VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2018 Lukáš Balaževič

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

BURZOVNÍ APLIKACE PRO SYSTÉM ANDROID
ANDROID STOCK MARKET APPLICATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Lukáš Balaževič**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Vojtěch Zvončák**
SUPERVISOR

BRNO 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Teleinformatika**
Ústav telekomunikací

Student: Lukáš Balaževič ***ID:*** 186416
Ročník: 3 ***Akademický rok:*** 2017/18

NÁZEV TÉMATU:

Burzovní aplikace pro systém Android

POKYNY PRO VYPRACOVÁNÍ:

Prostudujte Cryptonator API (Application Programming Interface). Vytvořte aplikaci pro operační systém Android. Aplikace bude notifikovat dle nastavení uživatele o aktuálním stavu vybraných kryptoměn a o jejich různých parametrech (například RSI). Dále do aplikace implementujte predikci ceny s využitím moderních metod strojového učení.

DOPORUČENÁ LITERATURA:

[1] DEITEL, Paul J. a Harvey M. DEITEL. Java how to program. Tenth edition. Boston: Pearson, 2015. ISBN 978-0-13-380780-6.

[2] Počítače a programování 2 [online]. [cit. 2017-09-15]. Dostupné z: https://moodle.vutbr.cz/

Termín zadání: 5.2.2018 ***Termín odevzdání:*** 29.5.2018

Vedoucí práce: Ing. Vojtěch Zvončák
Konzultant:

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Cieľom práce je navrhnúť a implementovať android aplikáciu ktorá bude zobrazovať parametre kryptomien, technické indikátory a bude predikovať cenu na základe moderných technológii strojového učenia. Okrem zobrazenia parametrov si používateľ bude môcť nastaviť notifikácie. Aplikácia zobrazuje dáta z Firebase databázy. Dáta sú nahrávané do aplikácie z dvoch služieb ktoré bežia na Ubuntu servery a sú napísané v jazyku Kotlin a Python. Služby majú na starosti získanie a spracovanie dát. Jedna zo služieb sa stará o predikciu ceny, cena sa predikuje pomocou (LSTM) neurónovej siete kde každá kryptomena má vytrénovaný vlastný model. Výsledkom práce je systém ktorý sa stará o celý životný cyklus dát od získania, transformovania až po prezentáciu dát používateľovi. Na základe informácii ktoré používateľ získa z aplikácie je schopný robiť štatistický správne rozhodnutia na trhu s kryptomenami bez nutnosti neustáleho sledovania trhu.

KĽÚČOVÉ SLOVÁ

mobilná aplikácia, Android SDK, kryptomeny, Kotlin, kryptografia, strojové učenie

ABSTRACT

The aim of the work is to design and implement an android application which will display parameters of cryptocurrencies, technical indicators and will predict the price based on the modern technology of machine learning. In addition to the displaying parameters, the user will be able to set up notifications. Application displays data from the Firebase database. The data are uploaded to an application from two services which run on Ubuntu server and are written in programming language Kotlin and Python. Services are in charge of obtaining and data processing. One of the services takes care of price prediction, the price is predicted using a LSTM neural network where each cryptocurrency has an own training model. The result of the work is a system that takes care of the whole lifecycle of the data from the acquisition, transformation and the presentation to the user. Based on acquired information from the application, the user is able to make statistically correct decisions in the cryptocurrency market without the need to constant market monitoring.

KEYWORDS

mobile application, Android SDK, cryptocurrencies, Kotlin, cryptography, machine learning

BALAŽEVIČ, Lukáš. *Burzovní aplikace pro systém Android*. Brno, 2018, 55 s. Bakalárska práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedúci práce: Ing. Vojtěch Zvončák,

VYHLÁSENIE

Vyhlasujem, že som svoju bakalársku prácu na tému „Burzovní aplikace pro systém Android“ vypracoval samostatne pod vedením vedúceho bakalárskej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej bakalárskej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto bakalárskej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákoníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora

POĎAKOVANIE

Rád bych poděkoval vedúcemu diplomové práce Ing. Vojtěch Zvončákovi za odborné vedenie, konzultácie, trpezlivosť a podnetné návrhy k práci.

Brno ....

podpis autora

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic
http://www.six.feec.vutbr.cz

POĎAKOVANIE

Výzkum popsaný v tejto bakalárskej práci bol realizovaný v laboratóriách podporených projektom SIX; registračné číslo CZ.1.05/2.1.00/03.0072, operačný program Výzkum a vývoj pro inovace.

Brno ....

podpis autora

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OP Výzkum a vývoj
pro inovace

Obsah

Zoznam obrázkov

Zoznam výpisov

Úvod

V poslednej dobe počujeme veľa o kryptomenách, čim ďalej tým viac ľudí sa zaujíma o kryptomeny. Niekto dáva kryptomenám veľkú budúcnosť. Veria, že novodobé platobné systémy postavené na kryptomenách postupne ovládnu trh, iný zase vravia, že kryptomeny nemajú budúcnosť a do pár rokov zaniknú. Na to, aby človek skutočne porozumel, ako kryptomeny fungujú, potrebuje vedieť, na akých základoch sú kryptomeny stavané. Potom bude chápať, aký je rozdiel v jednotlivých kryptomenách a ktoré meny by si mal vybrať pre svoje podnikanie, investovanie alebo inú aktivitu s kryptomenami.

Cieľom bakalárskej práce je vytvoriť mobilnú aplikáciu na platforme android, ktorá používateľom uľahčí prístup k aktuálnym informáciám ohľadne kryptomien. Cena a parametre kryptomien sa neustále vyvíjajú, aby používateľ dokázal dostatočne rýchlo reagovať na zmenu trhu, musí neustále sledovať stav kryptomien, čo je časovo náročné. Aplikácia CryptoStalker zjednodušuje tento faktor práve tým, že si používateľ v aplikácii môže nastaviť notifikácie na rôzne parametre kryptomien, ako sú napríklad cena, RSI, SMA a ďalšie. Tento spôsob sledovania trhu je pohodlnejší pre používateľa a dáva mu obchodnú výhodu na trhu tým, že používateľ okamžite dokáže reagovať na vývoj trhu. Používateľ by si taktiež mal dokázať prezerať rôzne technické parametre, aby vedel určiť dostatočne štatisticky presné odhady ceny. V tomto ohľade mu pomôže predikcia ceny, ktorá je implementovaná pomocou LSTM neurónovej siete.

Hlavný požiadavok na mobilnú aplikáciu je rýchlosť obnovy dát. Dáta sa musia obnovovať v reálnom čase, aby mal používateľ vždy najnovšie informácie. Ďalšia nutná funkcia je možnosť nastavenia notifikácie na jednotlivé kryptomeny podľa požiadaviek používateľa. Na používateľské rozhranie sú kladené nároky na jednoduchosť a intuitívnosť, preto sa riadi zásadami materiálového designu (Material Design).

Časť I

Teoretická časť

1 Kryptografia v kryptomenách

Všetky kryptomeny potrebujú spôsob ako kontrolovať vytváranie nových mincí (vytvorenie nového identifikátora ktorý predstavuje 1 mincu) a udržovať bezpečnosť aby sa predišlo podvodným transakciám. Klasické meny sú kontrolované a regulované organizáciami ako napríklad centrálna banka, ktorá zabezpečuje aj bezpečnostné mechanizmy pre fyzické bankovky. Tieto bezpečnostné opatrenia sťažujú útočníkom sfalšovanie, alebo iný útok na bankovky ale nie sú dokonalé a v určitých prípadoch môžu byť nedostatočné a hlavný problém je, že potrebujeme ďalšiu entitu ktorá v prípade pokusu o vytvorenie alebo použitie sfalšovaných bankoviek zakročí proti útočníkom. Kryptomeny tiež potrebujú mechanizmy na zabezpečenie, aby sa predišlo podvodným transakciám ktoré by rozhodili integritu systému čo by malo za následok, že rôzny ľudia by mali rôzne záznamy o množstve meny ktorú dané adresy obsahujú. Pri kryptomenách potrebujeme systém, ktorý sa nemusí spoliehať na ďalšie entity ktoré by spravovali bezpečnosť.

Práve z toho dôvodu sú kryptomeny závislé na kryptografických dátových štruktúrach a algoritmoch. Kryptografia poskytuje bezpečnostné mechanizmy pre zabezpečenie kryptomien. Na to aby sme správne pochopili ako kryptomeny fungujú, potrebujeme pochopiť kryptografické štruktúry a algoritmy na ktorých sú kryptomeny stavané.

1.1 Kryptografická hash funkcia

Hash funkcia je matematická funkcia ktorá ma 3 vlastnosti.

1. Vstup je reťazec akejkoľvek veľkosti
2. Výsledok má pevne danú dĺžku.
3. Má účinnú výpočtovú dobu, pri najhoršom prípade O(n).

Tieto tri vlastnosti určujú všeobecnú hash funkciu my sa zameriame na kryptografickú hash funkciu. Aby bola hash funkcia bezpečná pre kryptografické využite budeme od funkcie požadovať ďalšie tri vlastnosti:

1. Silná bezkolíznosť
2. Jednocestnosť
3. Slabá bezkolíznosť

Tieto vlastnosťi si popíšeme detailnejšie aby sme pochopili čo presne tieto vlastnosti znamenajú [9].

1.1.1 Silná bezkolíznosť

Ku kolízii dochádza ak hash funkcia vyprodukuje rovnaký výsledok pre dva rôzne vstupy.

Silná bezkolíznosť: Hash funkcia H má silnú bezkolíznosť ak je nemožné nájsť dve hodnoty, x a y pre ktoré platí:

$$x \neq y \wedge H(x) = H(y) \tag{1.1}$$

V skutočností vieme že existujú rôzne vstupy ktoré dokážu vyprodukovať rovnaký výsledok môžeme to aj dokázať úvahou. Výstup funkcie má pevne danú dĺžku zatiaľ čo vstup môže byť nekonečne dlhý vo všetkých kombináciach z toho nám vychádza že musia existovať dva rôzne vstupy ktoré vyprodukujú rovnaký výstup. V skutočnosti podľa Dirichletóvho princípu takých čísiel bude mnoho. Nie je to až taký problém ako by sa mohlo zdať v kryptomenách sa zväčša používajú hashe o veľkosti 256 bitov, aby sme pre taký hash našli kolíziu museli by sme ho porovnať s 2^{258+1} rôznymi vstupmi. Ak by sme mali stroj ktorý by dokázal počítať 10 000 hashov za sekundu, zabralo by to 10^{27} rokov[9].

1.1.2 Jednocestnosť

Jednocestnosť znamená že ak dostaneme výstup z hash funkcie y = H(x), tak neexistuje žiadny spôsob ako zistiť vstup x z výstupu y. Problém je že dané tvrdenie nemôže byť pravdivé samo o sebe. Predstavme si jednoduchý príklad, spravíme experiment kde budeme hádzať mincou. Ak padne rub zverejníme hash rubu, ak padne líc oznámime hash lícu. Ak niekto bude pozorovať experiment, ale nebude vedieť ktorá strana padla, ale uvidí len výsledný hash, tak môže vypočítať hash pre obe varianty a porovnať ich z našimi výsledkami. A iba v pár krokoch bude vedieť povedať čo bol vstupný parameter x. Pozorovateľ vedel odhaliť vstupné parametre iba preto lebo daný experiment má malý set vstupných dát. Aby sme dokázali získať vlastnosť jednocestnosti, nemôže sa v sete nachádzať x ktoré ma veľkú pravdepodobnosť výskytu. Inak povedané x musí byť zvolené zo setu dát ktorý je dostatočne rozložený. Aby sme dosiahli jednocestnosť aj na malom sete dát, vstupné dáta sa spájajú s ďalším parametrom ktorý má dostatočne rozloženie.

Jednocestnosť: hash funkcia H je jednocestná ak: keď tajná hodnota r je zvolená z pravdepodobnostného rozdelenia ktoré má vysokú minimálnu entropiu, potom z H(r || x) je nemožné nájsť x [9].

Minimálna entropia je meranie o tom aká je pravdepodobnosť predikcie výsledku, vysoká minimálna entropia je koncept veľmi veľkého rozloženia setu dát.

1.1.3 Slabá bezkolíznosť

Hash funkcia H má slabú bezkolíznosť ak pre každý n-bitový výstup y a ak k je vybrané zo setu dát s veľkou minimálnou entropiou, potom je nemožné nájsť x také kde H(k || x) = y v čase výrazne menšom ako 2^n [9].

To znamená že ak by dakto chcel aby výsledný hash mal určitú formu y, tak pokiaľ je časť vstupu zvolená v dostatočne náhodnom formáte tak je veľmi obtiažne nájsť ďalší vstup ktorý by mal danú výstupnú formu.

1.2 Hash ukazovateľ

Hash ukazovateľ je ukazovateľ na adresu kde sú dáta uložené spolu z cryptografickým hashom hodnoty dát v pevne danom bode v čase [9].

Pomocou tohto ukazovateľa môžeme vybudovať známe dátové štruktúry ako napríklad lineárny zoznam, binárny strom a mnoho ďalších.

1.3 Block chain

Nazvime lineárny zoznam vytvorený pomocou hash ukazovateľov block chain. Pri klasickom lineárnom zozname máme sériu dát kde každý blok obsahuje dáta a ukazovateľ na predchádzajúci blok v zozname, v block chaine bude klasický ukazovateľ nahradený hash ukazovateľom. Takže z každého bloku máme informácie nielen o dátach a kde sa predchádzajúci block dát nachádza, ale tiež obsahuje informácie bloku samotnom to nám pomáha overiť či sa z daným blokom manipulovalo.

Block chain sa využíva v tamper-evident log. Cieľom tejto dátovej štruktúry je zostaviť zoznam, ktorý uchováva dáta a môžeme v ňom pridať dáta na koniec zoznamu. Ale ak sa dakto pokúsi upraviť dáta, ktoré sa nachádzajú na konci zoznamu budeme schopný to poznať.

Aby sme pochopili prečo potrebujeme práve toto správanie položme si otázku, čo sa stane ak by dakto chcel zmeniť dáta ktoré sa nachádzajú v strede zoznamu. Útočník chce pozmeniť dáta tak aby osoba ktorá ma odkaz len na koniec zoznamu nebola schopná povedať že dáta boli pozmenené. Aby sa mu to podarilo musel by zmeniť block z dátami na pozícii k, keďže pozmenil dáta na mieste k, hash na mieste k+1, čo je hash bloku k nebude súhlasiť. Nezabúdajme že štatisticky máme garanciu že nový hash nebude zhodný zo starým hashom keďže hash má silnú bezkolíznosť. Takže zaznamenáme ne-konzistentnosť v dátach. Samozrejme útočník môže pokračovať a meniť dáta až kým sa nedostane po koniec zoznamu ale posledný blok nemôže zmeniť bez toho aby bol detekovaný [9].

1.4 Digitálny podpis

Digitálny podpis slúži na podobné účely ako ručne písaný podpis. Iba vy sa môžte podpísať vašim podpisom ale ktokoľvek kto ho vidí, môže potvrdiť či je platný. Ďalej chceme aby daný podpis bol viazaný na daný dokument a aby daný podpis nebol znovu použitý pre iný dokument. V ručne písanom podpise by táto analógia odpovedala tomu, že dakto nemôže prísť, vystrihnúť váš podpis a použiť ho pre iný dokument.

Schéma digitálneho podpisu. Schéma digitálneho podpisu je tvorená tromi algoritmami.

1. **(*sk*, *pk*) := generateKeys(keysize)** Metóda generateKeys má jeden parameter a to je veľkosť klúča a generuje pár kľúčov. Tajný kľúč sk ostáva v privátny a používa sa na podpis správ. Pk je verejný kľúč ktorý slúži na overenie vášho podpisu. Verejný kľúč rozdáme každému kto chce overovať naše podpisy.
2. **sig := sign(*sk* , message)** Metóda sign má parametre správu a privátny kľúč *sk* ako vstupné parametre. Výstup z metódy je podpis pre správu pomocou kľúča *sk*.
3. **isValid := verify(*pk* , message , sig)** Metóda overuje platnosť podpisu. Ako vstupné parametere predávame správu, verejný kľúč *pk* a podpis. Návratová hodnota metódy je boolean ktorý určuje či daný podpis je platný[9].

Podpis musí mať nasledujúce dve vlastnosti:

1. Platný podpis bol overený metódou verify.
2. Podpisy sú existenčne zabezpečené proti falšovaniu.

1.5 Verejné kľúče ako identity

Predpoklad je že zoberieme verejný kľúč, verejný kľúč z digitálneho podpisu, a budeme s nim jednať ako s identitou ktorá predstavuje určitú osobu alebo skupinu v systéme. Ak sa ocitne nová správa v systéme, ktorá je podpísaná a je verifikovaná pravdivo pod verejným kľúčom, môžeme so správou nakladať ako správou ktorú poslala osoba alebo skupina pod identitou kľúča. Aby dakto mohol pod identitou verejného kľúča, musí poznať korešpondujúci privátny kľúč. Tým že jednáme s verejnými kľúčmi ako identitami tak ktokoľvek si kedykoľvek môže vytvoriť novú identitu. Verejné kľúče sú náhodne generované reťazce to znamená, že nikto nevie povedať kto vlastní danú identitu.

Manažment decentralizovaných identít. Namiesto centralizovanej autority ku ktorej by ste museli prísť, aby ste sa mohli zaregistrovať ako používateľ v systéme sa môžte do systému prihlásiť sami. Nepotrebujete používateľské meno alebo oznamovať dakomu že budete určité používateľské meno používať. Ak chcete identitu stačí si ju vygenerovať a môžete si vytvoriť ľubovoľný počet identít. Ak chcete vystupovať pod piatimi rôznymi identitami, nie je problém, vytvorte si päť identít. Chcete novú identitu pod ktorou vás nikto nepozná ? Vytvoríte si novú identitu a keď vás omrzí vytvoríte si novú. Všetky tieto veci sú možné pomocou manažmentu decentralizovaných identít. Tento systém využíva aj najznámejšia kryptomena bitcoin. Tieto identity sa volajú adresy v bitcoinovom ponímaní. Pojem adresa je často využívaný pojem v bitcoine a aj iných kryptomenách, v skutočnosti to je len hash verejného kľúča [9].

1.6 Zhrnutie

Uviedli sme si základné kryptografické dátové štruktúry a algoritmy ktoré sa používajú na zabezpečenie kryptomien. Identity v systéme sú osoby alebo skupiny ktoré si ktokoľvek a kedykoľvek môže vytvoriť v množstve jeho uváženom. Tieto identity v danom systéme medzi sebou komunikujú pomocou tvrdení, ktoré sú podpísané súkromným kľúčom danej identity. Podpis je možné overiť pomocou verejného kľúča a tým určiť či je tvrdenie pravdivé. Mnoho kryptomien využíva block chain ako zoznam kde sa uschovávajú práve tvrdenia ktoré boli vytvorené pomocou identít. Toto tvrdenie nie je pravdivé pre všetky kryptomeny ale väčšina využíva určitú formu block chainu najznámejší je bitcoin. Dôvod prečo sa dané tvrdenia uschovavajú práve do block chainu je aby sa predošlo duplikátnym tvrdeniam kde identita v systéme by sa pokúsila vytvoriť tvrdenie ktoré by sa vzťahovalo na určitý blok v systéme ktorý nie je možné aby vlastnili dve identity v tom istom čase. Práve za to sa všetky tvrdenia pre tým než sú potvrdené zapisujú do block chainu aby obe strany mohli potvrdiť že ide o platné tvrdenie.

2 Indikátory technické analýzy

2.1 Index relatívnej sily RSI

RSI bolo vytvorené J. Welles Wilderom. Index relatívnej sily RSI sa radí medzi momentum oscilátor indikátory ktorý meria rýchlosť zmeny a veľkosť ceny. RSI sa osciluje medzi hodnotami 0 až 100. Najčastejšie a aj podľa Wildera sa indikátor používa takým spôsobom, že hodnoty nad 70 bodov poukazujú na prekúpenosť a hodnoty pod 30 bodov poukazujú na prepredanosť podkladového aktíva. Wilder prvý krát uviedol tento indikátor v knihe "New Concepts in Technical Trading Systems"v roku 1978 [7].

2.1.1 Výpočet

Pre uľahčenie výpočtu bol rozdelený na 3 časti: RS, priemerný vzrast, priemerný pokles. RSI sa väčšinou počíta počas doby 14 periód, tento postup bol navrhnutý Wilderom [14]:

$$\text{RSI} = 100 - \frac{100}{1 + RS}, \tag{2.1}$$

$$RS = \frac{AGD}{ALD}, \tag{2.2}$$

$$AGD = \frac{1}{n} * \sum_{n=1}^{n} a_{gain}, \tag{2.3}$$

$$ALD = \frac{1}{n} * \sum_{n=1}^{n} a_{loss}. \tag{2.4}$$

Prvý výpočet pre priemerný vzrast a pokles je jednoduchý priemer za 14 periód v našom prípade dní:

$$AGD_{first} = \frac{1}{14} * \sum_{n=1}^{14} a_{gain}, \tag{2.5}$$

$$ALD_{first} = \frac{1}{14} * \sum_{n=1}^{14} a_{loss}. \tag{2.6}$$

Nasledujúce výpočty pre priemerný vzrast a pokles vychádzajú z predošlých výpočtov:

$$AGD = \frac{AGD_{first} * 13 + a_{gain}}{14}, \tag{2.7}$$

$$ALD = \frac{ALD_{first} * 13 + a_{loss}}{14}. \tag{2.8}$$

Výpočet pomocou predchádzajúcich hodnôt je technika pre spresnenie výpočtu. Čim väčší je daný set dát, tým je výpočet presnejši [7].

2.2 Pohyblivé priemery jednoduchý a exponenciálny

Najznámejšie a najjednoduchšie indikátory technickej analýzy sú pohyblivé priemery (Moving Averages). Je to priemer hodnôt za určitú periódu. Najčastejšie sa tento indikátor používa na uzatváracie ceny.

2.2.1 Jednoduchý pohyblivý priemer SMA

Jednoduchý pohyblivý priemer je jeden z najpoužívanejších označuje sa aj ako SMA (Simple Moving Average):

$$SMA = \frac{(P_1 + P_2 + ...P_n)}{n}, \tag{2.9}$$

kde P_n je hodnota ceny v bode x v danom intervale 1 až n,
n je koncový bod diskrétneho intervalu v ktorom priemer počítame.

Pohyblivý priemer je dôležitým analytickým nástrojom, ktorý slúži na identifikáciu súčasných cenových trendov a potencionálnu zmenu v stanovenom trende. Najjednoduchšia forma použitia jednoduchého kĺzavého priemeru pri analýze je jeho využitie na rýchle zistenie, či je aktuálna cena nad trendom alebo pod. Ďalším populárnym, hoci o niečo zložitejším analytickým nástrojom, je porovnanie páru jednoduchých kĺzavých priemerov, pričom každý zohľadní iný časový rámec. Ak je krátkodobý kĺzavý priemer nad dlhodobým priemerom, očakáva sa nárast hodnoty. Na druhú stranu ak je dlhodobý kĺzavý priemer nad krátkodobým očakáva sa pokles hodnoty [8].

2.2.2 Exponenciálny pohyblivý priemer EMA

Exponenciálny pohyblivý priemer alebo aj označovaný EMA (Exponential Moving Average), znižuje oneskorenie tým že dáva väčšiu váhu novším cenám. EMA sa líši od SMA tým že jeho výpočet je ovplyvnený predchádzajúcimi hodnotami. Zatiaľ čo pri SMA nám stačí perióda o 5 hodnotách pri EMA potrebujeme viac ako 10 aby sme dostali dostatočne presný pohyblivý priemer. Výpočet EMA je troj krokový proces. Ako prvé vypočítame SMA pre prvú hodnotu v EMA. Druhý výpočet je vážený multiplikátor. Posledný výpočet je exponenciálny pohyblivý priemer pre každý deň medzi prvým EMA výpočtom a posledným [7].

$$Mp = \frac{2}{n+1}, \tag{2.10}$$

$$EMA_{first} = \frac{1}{n} * \sum_{n=1}^{n} a, \tag{2.11}$$

$$EMA_n = \frac{price_{close}}{EMA_{n-1} * Mp + EMA_{n-1}}. \tag{2.12}$$

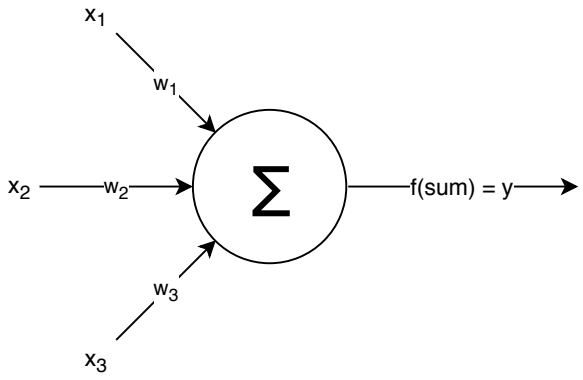
2.3 Pohyblivý priemer MACD

Pohyblivý priemer konvergencie a divergencie skratkou MACD (Moving Average Convergence/Divergence) vytvoril v 70. rokoch Gerald Appel. MACD je jeden z najefektívnejších indikátorov technickej analýzy. Vyjadruje vzťah medzi dvomi pohyblivými priemermi. Najčastejšie je to rozdiel 26 EMA s 12 EMA. 9 denný pohyblivý exponenciálny priemer sa používa ako signálová krivka. Histogram je pozitívny pokiaľ MACD krivka je nad signálovou krivkou a negatívny pokiaľ je signálová krivka nad MACD krivkou. Hodnoty 12, 26 a 9 sú typické hodnoty používané pre MACD [7].

3 Strojové učenie

3.1 Umelý neurón

Umelý neurón je abstraktný model biologického neurónu. Sila pripojenia vstupu je zakódovaná v synaptickej váhe. Intenzita vstupného signálu je modelovaná pomocou reálnych čísel. Umelý neurón pracuje v diskrétnych časových intervaloch, vstup je prečítaný a spracovaný krok po kroku synchrónne [10].



Obr. 3.1: Umelý neurón.

Umelý neurón na obrázku 3.1 je jednoduchá procesná jednotka. Neurón má pevne daný počet vstupov n, kde každý vstup je pripojený k neurónu pomocou synaptickej váhy w_i. Neurón sčíta vstupné hodnoty podľa rovnice 3.1. Aby neurón mohol vypočítať výslednú hodnotu musíme aplikovať aktivačnú funkciu. Aktivačná funkcia y môže byť jednoduchá hraničná funkcia alebo kontinuálna ne-lineárna funkcia [10].

$$y = S(\sum_{n=1}^{n} w_n * x_n). \tag{3.1}$$

3.2 Aktivačná funkcia

Čo je hlavnou úlohou neurónu ? Jednoducho povedané, vypočíta vážený súčet vstupov, pridá k nemu konštantu (bias) a potom rozhodne či má daný neurón byť aktivovaný (v skutočnosti má túto úlohu na starosti aktivačná funkcia). Predpokladajme neurón :

$$Y = \sum_{n=1}^{n} (w_n * x_n) + bias. \tag{3.2}$$

Hodnota Y môže byť v interlave $-\infty$ do $+\infty$. Neurón v skutočnosti nepracuje s intervalom vstupu z toho vyplíva, že priamo neovplivnuje aktiváciu. Pre túto funkcionalitu obsahuje neurón aktivačnú funkciu ktorá na základe hodnoty Y rozhoduje či sa daný neurón aktivuje alebo neaktivuje [12].

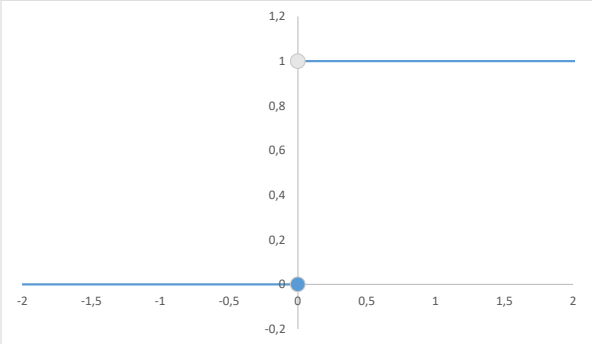
Aktivačná funkcia je jedna z najdôležitejších častí v neurónových sieťach. Konkrétne, ne-lineárne aktivačné funkcie sú základom aspoň z týchto dôvodov [11]:

- Pomáha dať zmysel neurónu pre zložité modely.
- Pridávajú ne-lineárne parametre do neuronovéj siete.
- Chceme aby malé zmeny vo váhe spôsobili malé zmeny na výstupe.

3.2.1 Kroková funkcia

Najjednoduchšia aktivačná funkcia je založená na hranici aktivácie pokiaľ hodnota presiahne túto hranicu neurón sa aktivuje ak túto hranicu nepresiahne neurón ostane neaktívny. Jednoduchý príklad môžme vidieť na rovnici 3.3 ktorá vyjadruje takúto krokovú funkciu y.

$$aky = \begin{cases} 0 & \text{ak } Y \leq 0. \\ 1 & \text{ak } Y > 0. \end{cases} \tag{3.3}$$



Obr. 3.2: Graf krokovej funkcie.

Predpokľadajme, že chceme vytvoriť binárny klasifikátor. Klasifikátor ktorý bude predpovedať hodnotu áno alebo nie (aktivované alebo neaktivované). Kroková funkcia toto správanie dokáže zreprodukovať. Je to presne to na čo je kroková funkcia vytvorená, vyprodukuje výstup ktorý je 0 alebo 1. Teraz si predstavme že chceme prípad použitia kde je viac takých neurónov prepojených aby sme mohli predikovať

viac tried. Problémom je ak viac ako jeden neurón bude aktívny, neurón bude mať nedeterministickú voľbu triedy výsledku.

Našim cieľom je aby iba jeden neurón bol aktivovaný a ostane boli ne-aktívne to je jediný spôsob ako môžme klasifikovať o akú triedu sa jedná. Týmto spôsobom by bolo príliš náročne vytrénovať danú sieť aby konvergovala správne. Lepšie by bolo ak by neuróny neboli binárne ale namiesto toho by neurón mohol byť aktivovaný na 50% alebo 20%. V tom prípade ak viac ako jeden neurón by bol aktivovaný by sme mohli zistiť ktorý neurón má najväčšiu hodnotu aktivácie a podľa toho rozhodnúť o ktorú triedu sa jedná [12].

Stále nastáva problém ak by viac ako jeden neurón bol aktivovaný na 100%. Ale tým že sieť má prechodnú hodnotu aktivácie na výstupe, trénovanie siete je jednoduchšie a šanca že dva neuróny budú na 100% aktivované je menšia ako pri binárnom neuróne. Funkcia ktorá nám ponúka kontinuálne hodnoty je lineárna funkcia [12].

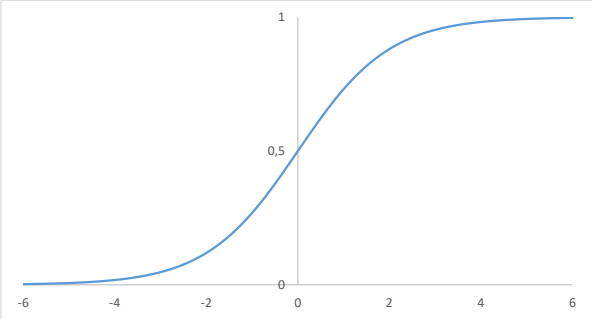
3.2.2 Sigmoid funkcia

Sigmoid je ne-lineárna funkcia. Kombinácia viacerých sigmoid funkcií je taktiež nelineárna. Takže môžeme vrstviť neuróny narozdiel od lineárnej funkcie. Aktivácia funkcie je ne-binárna takže nám ponúka viac informácii narozdiel od krokovej funkcie. Ako môžeme vidieť na grafe 3.3 vstupné hodnoty v intervale $< -2, 2 >$ majú rýchli nárast a pokles na Y osy, to znamená že zmena hodnoty X v tomto intervale, značne zmení výstupnú hodnotu Y [12]:

$$sigmoid(z) = \frac{1}{1 + e^{-z}}. \tag{3.4}$$

Podľa vlastností funkcie môžme určiť že sa jedná o dobrú klasifikačnú funkciu. Funkcia určuje jasné rozdiel pri predikcii. Ďalšia výhoda sigmoid fukcie je že na rozdiel od lineárnej funkcie, výstup bude vždy v intervale $(0, 1)$ v porovnaní z $(-\infty, +\infty)$ v lineárnej funkcii [12]. Sigmoid funkcia je jedna z najpoužívanejších aktivačných funkcii. Ale nie je dokonalá a má aj určité nevýhody.

V okrajových X hodnotách sa Y hodnoty takmer nemenia. To znamená že gradient v týchto intervaloch bude nadobúdať nízke hodnoty [12]. Tu vzniká nový problém takzvaný miznúci gradient (Vanishing Gradient). Ak sa hodnota výstupu dostane na tieto okrajové hodnoty, gradient je príliš malý alebo sa stratí (nedokáže určiť žiadnu zmenu kvôli nízkym hodnotám). Sieť sa nedokáže ďalej učiť alebo je príliš pomalá. Sú rôzne spôsoby ako sa s týmto problémom vysporiadať [12].

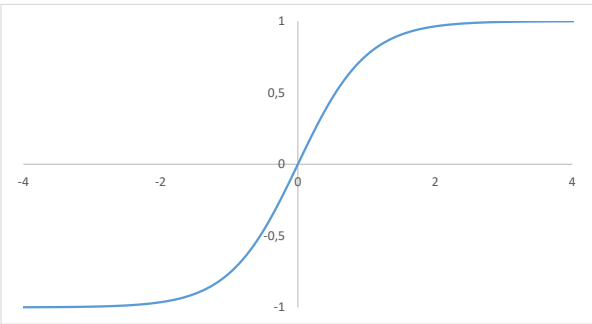


Obr. 3.3: Priebeh funkcie sigmoidu.

3.2.3 Tanh funkcia

Tanh alebo hyperbolický tangens 3.4 je ďalšia aktivačná funkcia ktorá sa bežne používa v neuronových sieťach hlbokého učenia. Vlastnosti tanh sú podobné ako sigmoid funkcie kde výstup je v pekvne danom intervale, výstup z tanh funkcie sa pohybuje v intervale (-1, 1) [12]:

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}. \tag{3.5}$$



Obr. 3.4: Tanh graf funkcie.

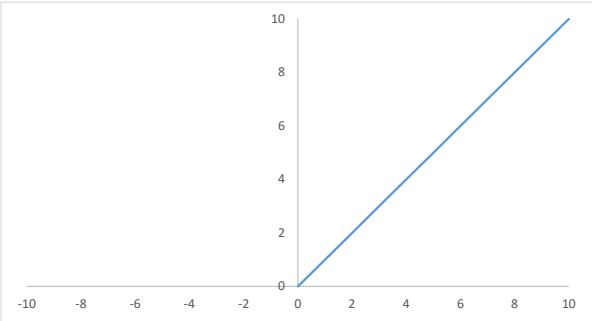
Tanh je ne-lineárna funkcia a výstup je v pevne danom intervale. Nevýhody tanh sú taktiež rovnaké ako pri signoid funkcii. Vzniká tu problém z miznúcim

gradientom, ako môžeme vidieť na rovnici 3.5 musíme počítať exponenciálu, čo je vo veľa prípadoch výpočetne ne-efektívne [12].

3.2.4 ReLu funkcia

ReLu funkcia 3.5 vracia na výstupe vstup x ak je kladný a inak 0. Na prvý pohľad to môže vyzerať že relu bude mať rovnaký problém ako lineárna funkcia, ale relu je ne-liíneárna funkcia a ich kombinácia je taktiež ne-lineárna. To znamená že môže vrstviť neuróny. Výstup je v intervale $(0, -\infty)$ [12]:

$$relu(z) = max(0, z). \tag{3.6}$$



Obr. 3.5: Graf ReLu funkcie.

Pri rozsiahlejšej neuronovéj sieti s vačším počtom neurónou, použitím sigmoid alebo tanh funkcie zapríčiní že takmer každý neurón bude aktivovaný kontinuálne. To znamená že každá aktivácia bude spracovaná na popísanie výstupu. Ideálne chceme dosiahnuť to aby iba pár neurónov bolo ne-aktivovaných a tým aktiváciu spraviť riedkou a efektívnou.

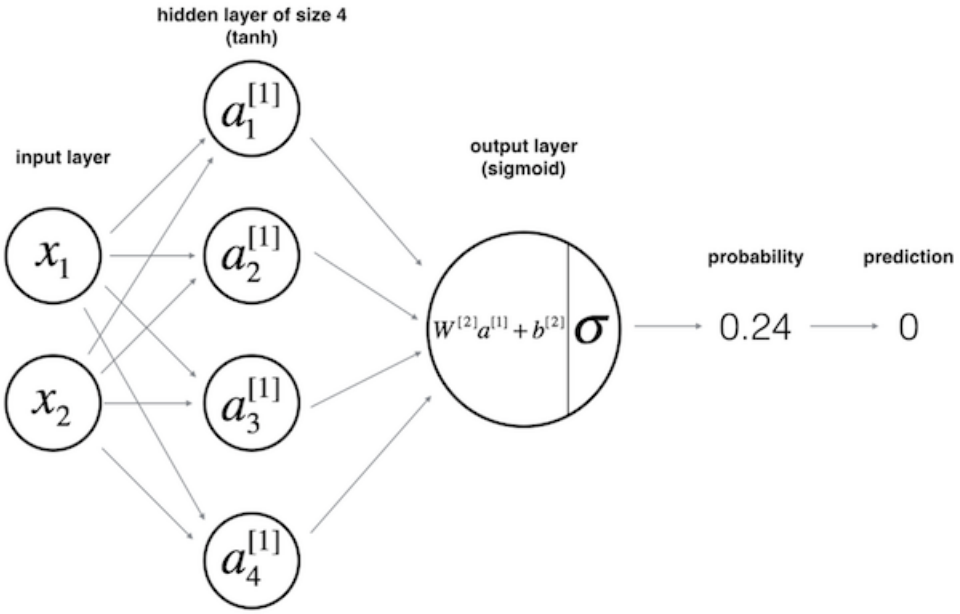
Relu nám dáva výhodu, v sieti s náhodnou váhou ako inicializátorom, takmer 50% siete bude mať výstup 0, kvôli charakteristike relu funkcie. To znamená že menej neurónou je aktívnych a sieť je výpočetne efektívnejšia [12].

Kvôli horizontálnej línii v záporných hodnotách, gradient sa môže približovať k 0. Pre aktivácie v tejto oblasti ReLu bude gradient 0, kvôli čomu sa váhy nebudú upravovať počas zostupu. To znamená, že tie neuróny, ktoré idú do tohto stavu, prestanú reagovať na odchýlky v chybách/vstupoch (jednoducho preto, že gradient je 0, nič sa nemení). Toto sa nazýva problém zomierajúceho ReLu. Tento problém môže

spôsobiť, že niekoľko neurónov jednoducho zanikne a nereagujú, čo činí podstatnú časť siete pasívnou. Existujú varianty v ReLu, ktoré zmierňujú tento problém tým, že jednoducho urobia z vodorovnej línii, nehorizontálnu zložku. Hlavnou myšlienkou je nechať gradient ne-nulový aby sa obnovil počas tréningu [12].

3.3 Trénovanie a predikcia v neuronových sieťach

Architektúra zobrazená na obrázku 3.6 sa nazýva viacvrstvový perceptron. Ako naznačuje názov, sieť je zostavená z viacerých vrstiev ktoré sa skladajú z perceptronov. Sieť zobrazená na obrázku sa skladá z troch vrstiev: vstupnej, skrytej a výstupnej. Konvencia v hlbokom učení a komunite je nepočítať vstupnú vrstvu do počtu.



Obr. 3.6: Viacvrstvový perceptron [11].

Neurónová sieť generuje predikciu potom ako všetky vstupy prejdú všetky vrstvy až k výstupnej vrstve. Tento proces sa nazýva dopredné šírenie (Feedforward). Sieť spracuje vstup, vypočíta aktivačnú funkciu a výsledok pošle na ďalšiu vrstvu až po výslednú vrstvu. Pri úlohách s kontrolovaným nastavením, ako je klasifikačná úloha, zvyčajne používame sigmoid ako aktivačnú funkciu vo výstupnej vrstve, pretože môžeme preložiť jej výstup ako pravdepodobnosť [11].

Potom, ako v typickej klasifikačnej úlohe, budeme mať funkciu nákladu, ktorá meria, ako dobrý je náš model približný k skutočnej hodnote. Trénovanie neurónovej sieti znamená jednoducho zminimalizovať náklady čo najviac. Môžeme definovať našu nákladovú funkciu takto:

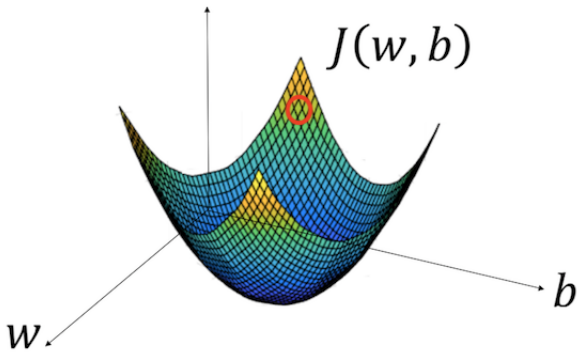
$$J(w,b) = \frac{1}{n}\sum \frac{1}{2}(Y - \hat{Y})^2, \tag{3.7}$$

kde $\hat{Y}$ je vektor n predikcií,
Y je vektor nadobudnutých hodnôt.

Takže cieľom je nájsť nejakú kombináciu w a b, ktorá by mohla spôsobiť, že naše náklady J budú čo najmenšie. Aby sme to urobili, spoliehame sa na dva dôležité algoritmy, ktorými sú gradient descent a spätná propagácia [11].

3.4 Gradient Descent

Trénovanie neurónovej siete nie je rozdielne od iných modelov strojového učenia ktoré sa učia pomocou algoritmu gradient descent. Jediným väčším rozdielom by bol vplyv nelinearít v našej sieti, ktorý robí našu nákladovú funkciu nekonvexnou [11].



Obr. 3.7: Algoritmus Gradient Descent [11].

Na diagrame 3.7 môžme vidieť že horizontálne osy vyjadrujú naše parametre. Zatiaľ čo naša nákladová funkcia, $J(w,b)$ sa nachádza na vertikálnej osy. Aby sme minimalizovali náklady, vieme, že musíme ísť na najhlbší bod. Ale otázkou je, ako vieme, ktorý smer vybrať? Mali by sme zvýšiť alebo znížiť hodnotu našich parametrov? Môžeme robiť náhodné vyhľadávanie, ale to bude trvať pomerne dlho a to je samozrejme výpočtovo drahé [11].

Existuje lepší spôsob, ako zistiť, akým smerom by sme mali ísť, aby sme upravili učiteľné parametre. Matematická analýza nás učí, že smer gradientového vektora v danom bode bude prirodzene smerovať k najstrmšiemu smeru. Preto budeme používať gradient našej nákladovej funkcie [11].

3.5 Neuronové siete Long Short-Term Memory

Long Short-Term Memory zvyčajne len nazývané LSTM, sú špeciálnym druhom RNN, schopnými učiť sa dlhodobé závislosti. Boli predstavené Hochreiterom a Schmidhuberom v roku 1997, a boli vylepšene a popularizované mnohými ľuďmi v nasledujúcich prácach.

Pracujú veľmi dobre na veľkom množstve problémov, a sú teraz široko používané. LSTM sú explicitne navrhnuté tak, aby sa vyhli dlhodobému problému závislosti. Pamätať si informácie na dlhú dobu je prakticky ich predvolené správanie, nie niečo, čo sa snažia naučiť.

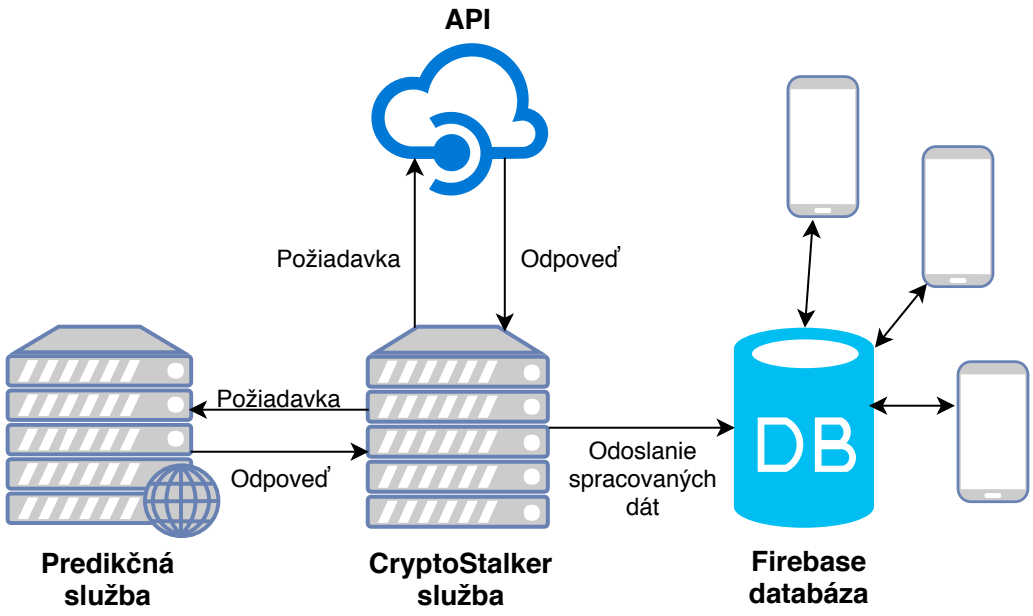
Všetky RNN majú formu reťazca opakujúcich sa modulov neurónovej siete. V štandardných RNN bude tento opakujúci sa modul mať veľmi jednoduchú implementáciu, ako je napríklad jedna tanhová vrstva. LSTM majú tiež podobnú reťazovú implementáciu, ale opakujúci sa modul má inú stavbu. Namiesto toho, aby mali len jednu vrstvu neurónovej siete, existujú štyri, ktoré spolu spolupracujú veľmi špecifickým spôsobom [13].

Časť II

Praktická časť

4 Model komunikácie

V tejto kapitole si popíšeme ako mobilná aplikácia, crypto stalker služba, predikčná služba a externé API spolu komunikujú. Na obrázku 4.1 môžeme vidieť diagram komunikácie. Iniciátorom komunikácie je crypto stalker služba ktorá sa ako prvé odosiela požiadavku na externé API. Požiadavka sa odosiela pomocou HTTP 1.1 protokolu konkrétne metódou GET. Odpoveď od API sú požadované dáta v JSON formáte. Crypto stalker služba tieto dáta spracuje a následne odosiela pomocou HTTP 1.1 metódy POST, potrebné dáta na predikciu ceny v JSON formáte na predikčnú službu ktorá spracuje tieto dáta a odošle predikciu v JSON formáte v odpovedi. V tento moment crypto stalker služba dopočíta indikátory technickej analýzy a odošle dáta do firebase databázy. Crypto stalker aplikácia v mobilných zariadeniach sa priamo pripája na firebase databázu. Pripojenie je na bázy TCP sieťových socketov (Network Socket). To znamená že pripojenie je obojstranné čo zaručuje obnovenie dát v reálnom čase na mobilných zariadeniach.



Obr. 4.1: Diagram komunikácie.

5 Crypto stalker služba

V tejto kapitole si popíšeme čo všetko má za úlohu crypto stalker služba a popíšeme si implementáciu úloh. Crypto stalker služba je hlavný prvok celého systému. Riadi komunikáciu medzi prvkami, spracúva a transformuje dáta ktoré následne odosiela do firebase databázy.

5.1 Diagram prípadov použitia

Táto sekcia sa zaoberá rozborom a popisom použitia crypto stalker služby 5.1 z hľadiska administrátora. Administrátor dokáže v behovom prostredí zmeniť konfiguráciu aplikácie ktorá sa následne prispôsobí novej konfigurácii. Parametre ktoré dokáže administrátor zmeniť sú primárne meny, sekundárne meny, zmena pomeru obnovy dát a cesta k súboru s právami na firebase databázu.

Administrátor môže službu spustiť v dvoch režimoch, prvý je ako služba na získavanie dát, transformáciu a odoslanie do firebase databázy. V tomto režime sa naplánujú dve úlohy. Prvá úloha je na získavanie aktuálnych dát, na ktorých sa počítajú technické indikátory a predikcia ceny. Úloha sa vykonáva podľa pomeru zadaného v konfiguračnom súbore. Druhá úloha je na získanie historických dát a technických indikátorov.

Druhý režim je na získanie historických dát ktoré sú následne uložené do CSV súborov. Tieto súbory slúžia na trénovanie LSTM modelu pre predikciu ceny.

Aktívny účastníci

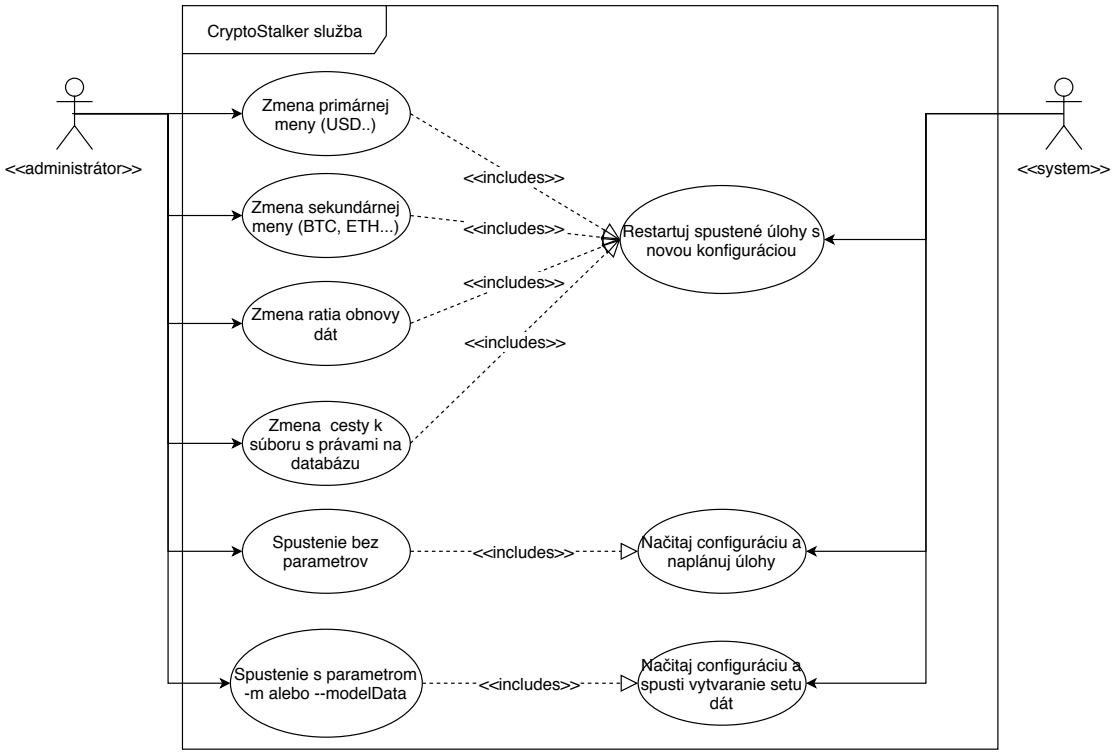
- Administrátor.
- Systém.

Spúšťače

- Administrátor pridá/zmení/odstráni primárnu menu.
- Administrátor pridá/zmení/odstráni sekundárnu menu.
- Administrátor zmení pomer obnovy dát.
- Administrátor pridá/zmení/odstráni cestu k súboru s právami.
- Administrátor spustí službu bez parametrov.
- Administrátor spustí službu s parametrom -m.
- Administrátor spustí službu s parametrom –modelData.

Predpoklady

- Stabilné pripojenie na Internet pri štarte aplikácie.
- Správna funkcia externého API a databázy.



Obr. 5.1: Diagram prípadov použitia crypto stalker služby

Ciele

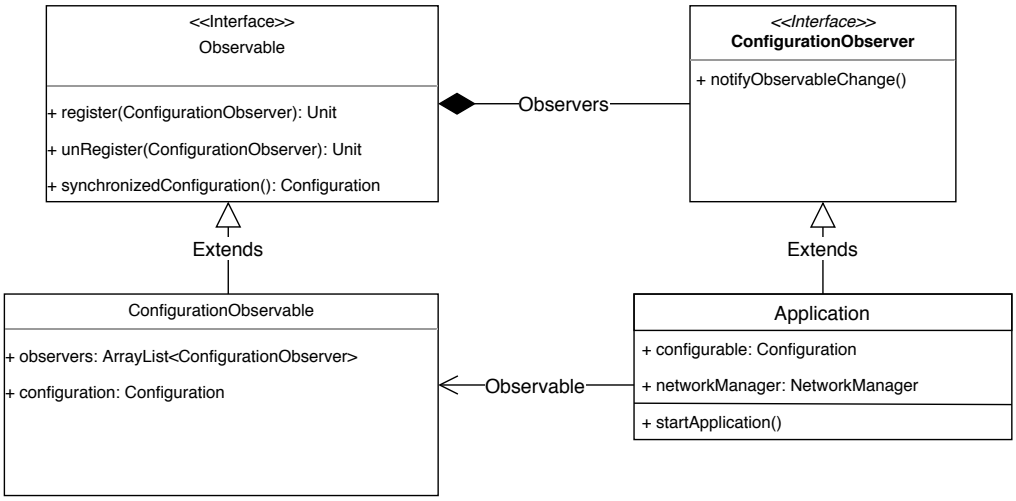
- Získavanie aktuálnych dát v určenom pomere obnovy.
- Výpočet indikátorov technickej analýzy.
- Odoslanie spracovaných dát do firebase databázy.
- Vytvorenie setu dát pre trénovanie LSTM modelu.

5.2 Zmena konfigurácie

Táto sekcia sa zaoberá rozborom a implementáciov zmeny konfigurácie v službe. Je dôležité aby aplikácia dokázala rozpoznať zmenu konfigurácie okamžite po jej zmenení administrátorom.

Aby služba dokázala túto úlohu efektívne plniť bola implementovaná na bázy návrhového vzoru pozorovateľ, diagram tried môžme vidieť na obrázku 5.2. Trieda ConfigurationObservable implementuje rozhranie Observable. To nám zaručí že daný pozorovatelia ktorý sa budú chcieť zaregistrovať na pozorovanie zmeny konfigurácie budú implementovať rozhranie ConfigurationObserver a tým budeme mať prístup k metóde notifyObservableChange() v triede ConfigurationObservable. V našom prípade má ConfigurationObservable iba jedného pozorovateľa ktorým je aplikácia.

Trieda ConfigurationObservable obsahuje objekt WatchService ktorý sleduje na novom vlákne zmenu súboru. Ak sa súbor zmení, prebehne spätné volanie do aplikácie, vytvorí sa nový konfiguračný objekt a zaregistrovaný pozorovatelia sú upozornený na zmenu konfigurácie zavolaním metódy notifyObservableChange(). Týmto spôsobom je zaručená okamžitá odozva aplikácie na zmenu konfigurácie. Aplikácia má prístup k objektu z triedy ConfigurationObservable, pri upozornení pozorovateľa si pozorovateľ pomocou tohto objektu a metódy synchronizedConfiguration() získa nový konfiguračný objekt.

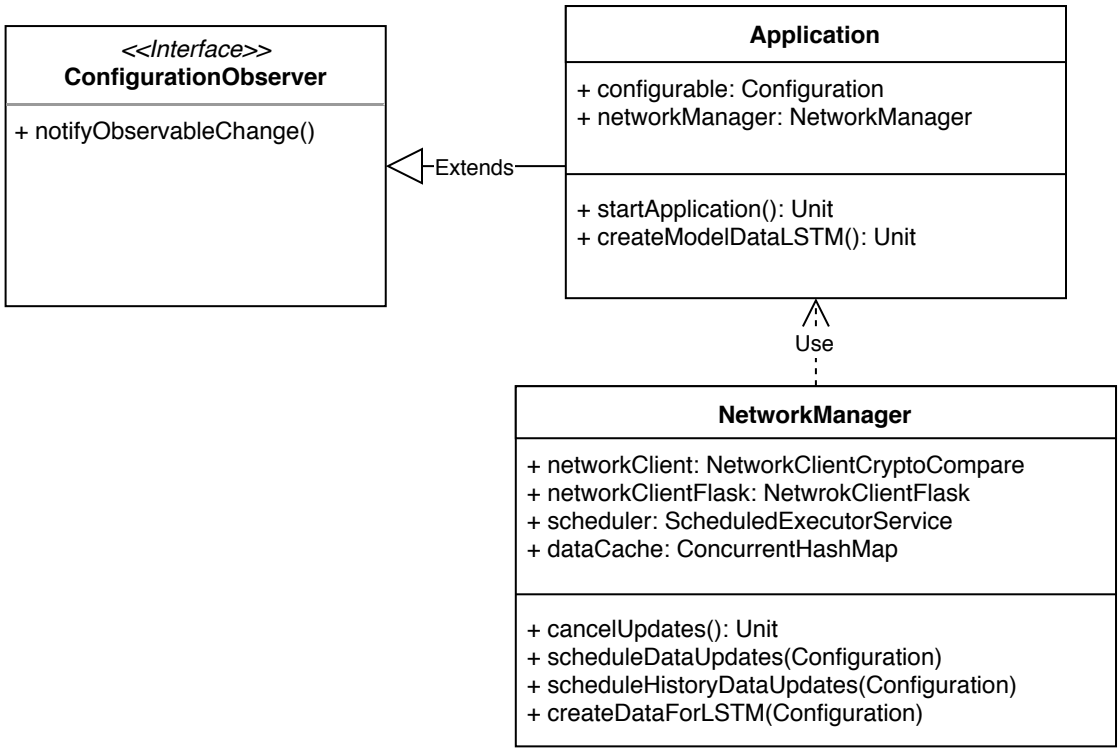


Obr. 5.2: Diagram zmeny konfigurácie.

5.3 Plánovanie úloh

Pri štarte aplikácie pomocou metódy startAplication() 5.3 v triede Application sa pomocou objektu z triedy NetworkManager, naplánujú dve úlohy. Ako prvá sa naplánuje úloha ktorá stiahne historické dáta a vytvorí históriu pre technické indikátory a následne úloha ktorá získava aktuálne dáta, indikátory a predikciu ceny. V NetworkManagery je vytvorený ScheduledThreadPool o velikosti jedna. Naplánované úlohy sú opakovane spúšťané v zadanom časovom pomere.

Pri zmene konfigurácie sa naplánované úlohy zrušia, pokiaľ niektorá z úloh práve prebieha Scheduler počká na dokončenie a následne úlohu ukončí. Následne sú úlohy naplánované podľa nového konfiguračného objektu.

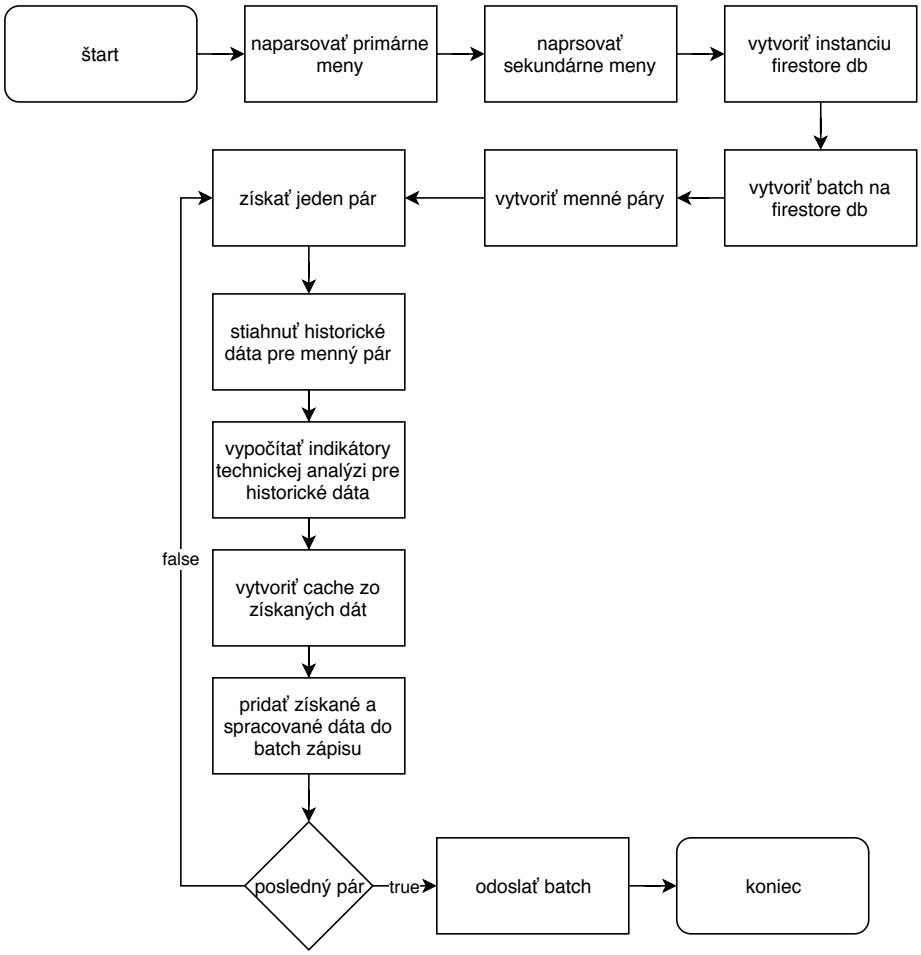


Obr. 5.3: Diagram tried plánovania úloh.

5.4 Úloha: Historické dáta

Hlavnou úlohou je zabezpečiť získanie historických dát pre menné páry. Pre zjednodušenie je na obrázku 5.4 uvedený vývojový diagram. Primárne a sekundárne meny sú získané z konfiguračného súboru. Úloha nepristupuje priamo ku konfiguračnému súboru, namiesto toho používa konfiguračný objekt ktorý obsahuje naformátované všetky údaje z konfiguračného súboru. Primárne a sekundárne meny sú transformované do párov. API nám nepovoľuje získať dáta pre všetky páry v jednej požiadavke. Postupne posielame požiadavky pre menné páry. V odpovedi dostávame historické dáta v JSON formáte. Dáta transformuje na nami vytvorenú objektovú štruktúru. Po získaní historických dát vypočítame pomocou nami vytvorenej triedy Analyzer, technické indikátory. Technické indikátory ktoré sa vypočítajú sú index relatívnej sily (RSI), jednoduchý kĺzavý priemer (SMA), kĺzavý priemer konvergencie / divergencie (MACD), ON BALANCE VOLUME (OBV). Ako sa jednotlivé indikátory počítajú bude vysvetlené v následnej sekcii. Posledná vec ktorú úloha robí pred zapísaním do batch zápisu je vytvorenie cache dát. Cache dáta sú použité hlavne na výpočet technických indikátorov, v úlohe ktorá získava aktuálne dáta pre menné páry. Všetky získané dáta sú zapísané do batch zápisu. Batch write nám povoľuje odoslať väčšie množstvo do databázy v jednom požiadavku. To nám znižuje záťaž

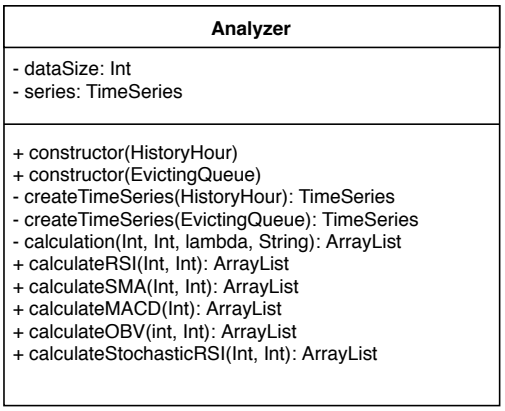
na databázu.



Obr. 5.4: Vývojový diagram - historické dáta.

5.5 Výpočet technických indikátorov

Technické indikátory sa počítajú pomocou ta4j knižnice ktorá je najpoužívanejšia knižnica v jave na počítanie technických indikátorov. Diagram triedy je na obrázku 5.5. Dáta potrebné na výpočet technických indikátorov sú získané z HistoryHour objektu alebo EvictingQueue<Pair<Long, Double». Dáta sú spracované a je pomocou nich vytvorený TimeSeries. TimeSeries je základná dátová štruktúra použitá v ta4j knižnici. Funkcia calculation() slúži ako generická funkcia ktorú používajú všetky nami počítané technické indikátory, jeden z parametrov je lambda funkcia ktorá je individuálna pre každý technický indikátor. Metódy calculateRSI(), calculateSMA() a ďalšie ktoré sú uvedené v diagramu triedy, volajú funkciu calculation() a predávajú jej potrebné parametre pre výpočet parametru.



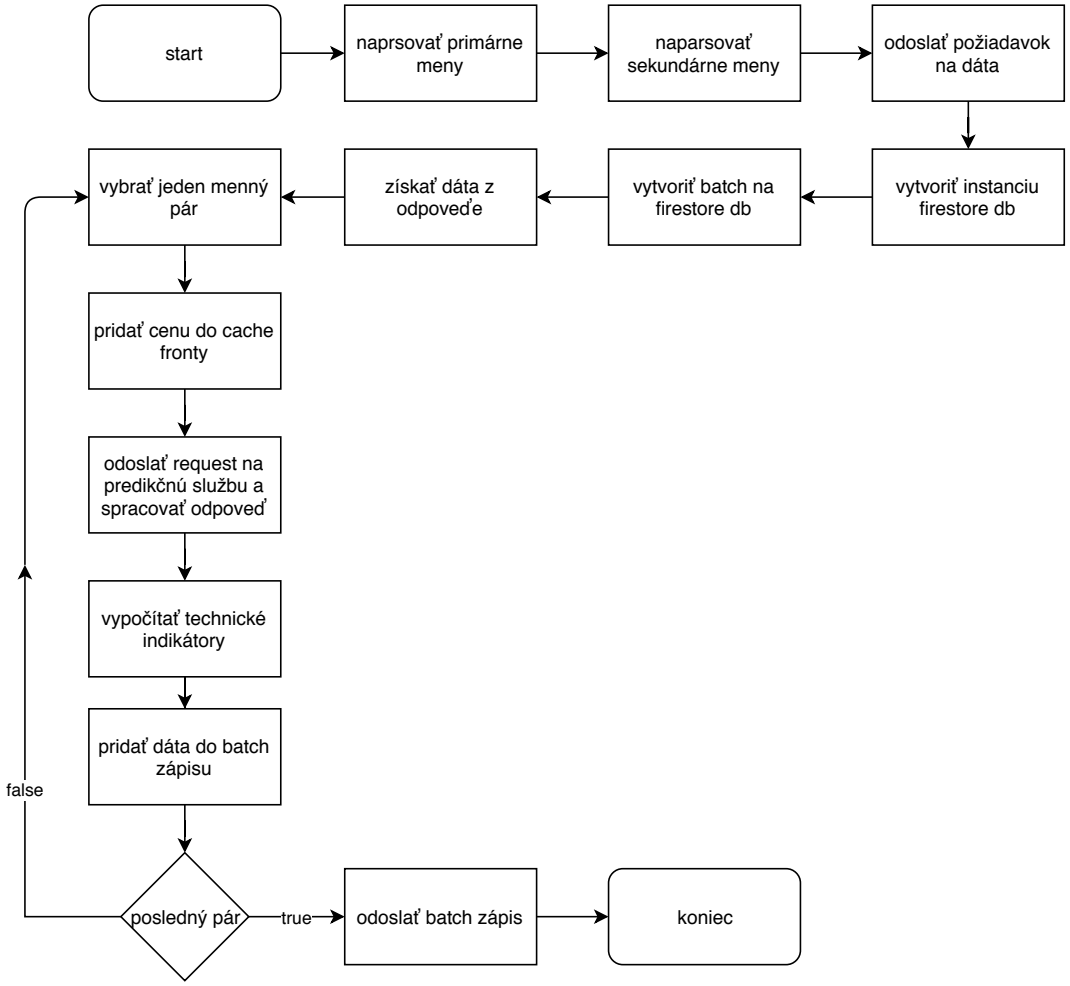
Obr. 5.5: Diagram triedy - Analyzer.

5.6 Úloha: Aktuálne dáta

V poradí druhá vykonávaná úloha, má za úlohu získať aktuálne dáta o menných pároch uvedených v konfiguračnom objekte. Na obrázku 5.6 je uvedený vývojový diagram. Na API je odoslaný jeden požiadavok ktorý obsahuje všetky sekundárne a primárne meny pre ktoré požadujeme dáta. API skombinuje meny a ako návratovú dátovú štruktúru sme definovali triedu Data. Získané menné páry sú iterované v cykluse. Na mennom páre je počítaná predikcia ceny a indikátory technickéj analýzy.

5.7 Monitorovanie a chybové stavy

Aby sme mohli efektívne riešiť chybné stavy a zistiť kde nastal problém musíme, zaviesť protokol na logovanie a riešenie chybných stavov. Pre to nám slúži trieda Log. Trida Log je založená na princípe návrhového vzoru jedináčik (Singleton). Pri inicializácii sa vytvorí súbor do ktorého sú zapisované všetky logované udalosti. Súbor je nastavený na limit 1 MB. Po prekročení tohto limitu sa súbor premaže novým súborom. Aby sme rozlíšili o aký chybový stav od stavu programu tak môžme logovať na dvoch úrovniach závažnosti. Prvá úroveň je informačná aby sme vedeli v akom stave sa aplikácia momentálne nachádza. Druhá úroveň je chybná. Pri chybnej úrovni sa do logu vypíše informácia o probléme, detailná správa výnimky a návratový chybový kód. Chybné stavy sú stavy pri ktorých program nemôže pokračovať v činnosti. Riešenie je reštart aplikácie. Všetky chybné návratoví kódy sú popísané v výpočtovej (enum) triede ErrorStatus.



Obr. 5.6: Vývojový diagram - Aktuálne dáta.

6 Predikčná služba

Predikčná služba slúži na predikciu zmeny ceny. V tejto kapitole bude popísané ako predikčná služba funguje. Implementacia HTTP Serveru 6.1

```
import glob
import os
import numpy as np
from flask import Flask, jsonify
from flask import request
from core import Predict

def create_app():
    def initModels():
        localcache = dict()
        dir_path = os.path.dirname(os.path.realpath(__file__))
        models = glob.glob(f"{dir_path}/models/*.json")
        for model in models:
            loaded_model = Predict.open_model(model)
            key = model.replace(f'{dir_path}/models/', '').
   replace('_model.json', '')
            print(key)
            localcache[key] = loaded_model
        return localcache
    app = Flask(__name__)
    cache = initModels()

    @app.route('/predict', methods=['POST'])
    def predict():
        data = request.get_json()
        price = data['price']
        pair = data['pair']
        model = cache.get(pair)
        data = Predict.get_predicted_price(model, price)
        return jsonify(
            data=np.reshape(data, len(data)).tolist()
        )
    return app
if __name__ == "__main__":
    app = create_app()
    app.run()
```

Výpis 6.1: HTTP Server.

Predikčná služba je funguje ako REST API. Na koncový bod /prediction je poslaný POST požiadavok s parametrami 'price' a 'pair'. Pair je reťazec ktorý obsahuje primárnu a sekundárnu menu ktoré tvoria pár na ktorom chceme predikovať cenu. Price sú historické dáta ceny. Tento požiadavok služba spracuje a na základe dát predikuje cenu. Pred štartom HTTP serveru sa do pamäte načítajú vytrénované LSTM modeli. Aplikácia podporuje 200 kryptomien, každá kryptomena ma osobitne vytrénovaný LSTM model čo nám zaručuje väčšiu presnosť pri predikcii. Hlavný dôvod prečo modeli načítame do pamäte je kvôli rýchlosti serveru. Dynamické načítanie zo súboru je príliš pomalé pre naše požiadavky. HTTP server je vytvorený pomocou Flask knižnice. Flask je python knižnica na vytvorenie REST API ktorá obsahuje zabudovaný HTTP server.

6.1 Dáta

Dáta sú získané z cryptocompare API pomocou crypto stalker služby. Posledných dvetisíc zmien je spracovaných do štruktúry ktorá ma dva dátové polia, cenu a časovú značku. Dáta sú uložené do CSV súboru ktorého názov odpovedá mennému páru dát ktorý je v ňom uložený. Pri trénovaní modelu sa dáta načítajú z CSV súborov.

```
def train_model(path, name):
 df = pd.read_csv(path)
 group = df.groupby('Timestamp')
 real_price = group['price'].mean()
 df_train = real_price[:len(real_price)]

 training_set = df_train.values
 training_set = np.reshape(training_set, (len(
  training_set), 1))

 sc = MinMaxScaler()
 training_set = sc.fit_transform(training_set)
 x_train = training_set[0:len(training_set) - 1]
 y_train = training_set[1:len(training_set)]
 x_train = np.reshape(x_train, (len(x_train), 1, 1))
 ......
```

Výpis 6.2: Príprava dát pre model.

Dáta sú z CVS súboru načítané pomocou knižnice pandas a sú normalizované. Pojmom normalizované myslíme upravenie dát tak, ak sa v jednej časovej značke nachádza viac ako jedna hodnota je pole pretransformované do dvoj dimenzionálneho

poľa kde druhá dimenzia sú hodnoty na danej časovej značke. Následne je na časovej značke spočítaný priemer hodnôt a ten je uložený ako hodnota pre danú časovú značku. Tento mechanizmus normalizácie slúži pre elimináciu chybných dát a udržuje integritu kódu bez nutnosti ho upravovať pri zmene predikcie napríklad na hodiny alebo dni.

Časovú značku potrebujeme len na vizualizácia výsledných dát. Na riadku sedem sa odstráni časová značku a dáta budú mať formát 2D homogénneho poľa. Kde prvá dimenzia bude mať iba jeden index. V druhej dimenzii bude cena v zachovanom časovom poradí. V ďalšom kroku meníme tvar poľa pričom tvar poľa sa mení ale dáta zostávajú nezmenené. V našom prípade meníme X a Y osu. Pomocou MinMaxScaler dáta transformujeme medzi hodnotu 0 až 1. Ďalej si na definujeme dáta na X a Y osu pre model. Našim cielom je predikovať nasledujúcu hodnotu pomocou predchádzajúcej hodnoty, ciže:

$$y(x+1) = f(x). \tag{6.1}$$

6.2 Model

Ako prvé inicializujeme rekurentnú neurónovú sieť 6.3. A pridáme LSTM vrstvu ktorá ma 50 výstupných uzlov, ako aktivačnú funkciu používa sigmoid. Ďalej pridáme výstupnú vrstvu, použijeme Dense kde výstup má 1 uzol. Ako optimalizér použijeme Adam optimalizačný algoritmus, algoritmus obnovuje vzťahy medzi dátami iteratívne podľa tréningových dát.

```
.....
# Initialising the RNN
regressor = Sequential()

# Adding the input layer and the LSTM layer
regressor.add(
     LSTM(
        units=50,
        activation='sigmoid',
        input_shape=(None, 1)))

# Adding the output layer
regressor.add(Dense(units=1))

# Compiling the RNN
regressor.compile(optimizer='adam', loss='mean_squared_error')

# Fitting the RNN to the Training set
regressor.fit(x_train,
         y_train,
         batch_size=5,
         epochs=100,
         verbose=0)

model_json = regressor.to_json()
with open(f'models/{name}_model.json', "w") as json_file:
   json_file.write(model_json)

regressor.save_weights(f'models/{name}_model.h5')
.....
```

Výpis 6.3: Vytvorenie LSTM modelu.

Aby sme zistili ako presné náš model predikuje a mohol porovnávať pri tréningu ako sa zlepšuje zadefinujeme loss funkciu ktorá bude určovať ako sa model zlepšuje,

použijeme MEAN SQUARED ERROR. Týmto máme model zadefinovaný, posledná vec musíme modelu definovať z kade má získať dáta na trenovanie a porovnávanie, v akých celkoch môže obnovovať gradient a na koľko epoch sa model bude trénovať. Toto všetko je definované na riadku 19 vo výpise 6.3. Vytrénovaný model sa uloží do JSON súboru a vzťahy medzi dátami do binárneho súboru.

6.3 Predikcia

Pri predikcii je spracovanie dát podobné ako pri trénovaní modelu 6.4. Tvar poľa je zmenený na 2D homogénne pole a dáta sú transformované pomocou MinMaxScaler medzi hodnoty od 0 po 1. Takto zmenené dáta sú predané ako parameter modelu vo funkcii predict(). Návratový typ je predikcia ktorú musíme spätne transformovať pomocou MinMaxScaler.

```
1  def get_predicted_price(model, data):
2      if model is None:
3          return [0.0]
4      sc = MinMaxScaler()
5      size = len(data)
6      data = np.ndarray((size,), buffer=np.array(data))
7      inputs = np.reshape(data, (len(data), 1))
8      inputs = sc.fit_transform(inputs)
9      inputs = np.reshape(inputs, (len(inputs), 1, 1))
10     predicted_price = model.predict(inputs)
11     predicted_price = sc.inverse_transform(predicted_price)
12     return predicted_price
```

Výpis 6.4: Predikcia ceny.

7 CryptoStalker aplikácia

CryptoStalker aplikácia je mobilná aplikácia pre systém android. Slúži ako analytický nástroj ktorý pomáha obchodníkom s kryptomenami spraviť správne rozhodnutia v správnom čase. Kryptomeny majú vysokú kolísavosť ceny, aplikácia umožňuje si nastaviť sledovanie na rôzne parametre kryptomeny a pokiaľ sa parametre zmenia podľa používateľových nastavený je informovaný prostredníctvom android notifikácie. Ďalej aplikácia obsahuje technické indikátory ktoré pomáhajú používateľovi sa rozhodnúť či ma danú menu kúpiť alebo predať. Aplikácia podporuje android 5.0 a vyššie a je napísaná v kotline.

7.1 Diagram prípadov použitia

Táto sekcia sa zaoberá rozborom a popisom použitia crypto stalker mobilnéj aplikácie z hľadiska používateľa 7.1. Používateľ sa môže v aplikácii pohybovať medzi tromi hlavnými obrazovkami. Na prvej obrazovke je list všetkých menných párov zoradených podľa množstva meny v obehu za 24h. Toto zoradenie je možné zmeniť na zoradenie podľa ceny. Používateľ môže položky pridávať do zoznamu obľúbených a monitorovaných mien. Druhá obrazovka je zoznam mien ktoré si používateľ pridal do svojich obľúbených. Tretia obrazovka je zoznam monitorovaných mien používateľom. Po kliku na ikonku srdiečka je položka pridaná alebo odobraná zo zoznamu obľúbených. Aký stav nastane zaleží od toho či sa daná položka už v obľúbených nachádza. Po kliku na ikonku oka sa otvorí monitor aktivita kde si používateľ môže nastaviť parametre ktoré si želá sledovať, hodnotu zmeny pri ktorej chce byť používateľ upozornený android notifikáciou môže byť zadaná sklárne alebo percentne. Po nastavení parametrov používateľ potvrdí tlačidlom voľbu a položka bude pridaná do monitovaných. Pri kliku na ikonu oka, ak sa položka v zozname už nachádza otvorí sa dialóg kde si používateľ môže vybrať upravenie monitorovaných parametrov alebo odstránenie zo zoznamu monitorovaných mien. Ak používateľ klikne na položku v zozname je otvorená nová detailná aktivita kde sú uvedené detailné informácie o položke.

Aktívny účastníci

- Používateľ.
- Systém.

Spúšťače

- Klik udalosť na domovský fragment.
- Klik udalosť na obľúbený fragment.

- Klik udalosť na monitorovaný fragment.
- Klik udalosť na položku v zozname.
- Zmeniť dotaz na zoradenie.
- Vyhľadať menný pár.
- Používateľ pridá/odstráni položku do obľúbených.
- Používateľ pridá/odstráni položku do monitorvaných.

Predpoklady

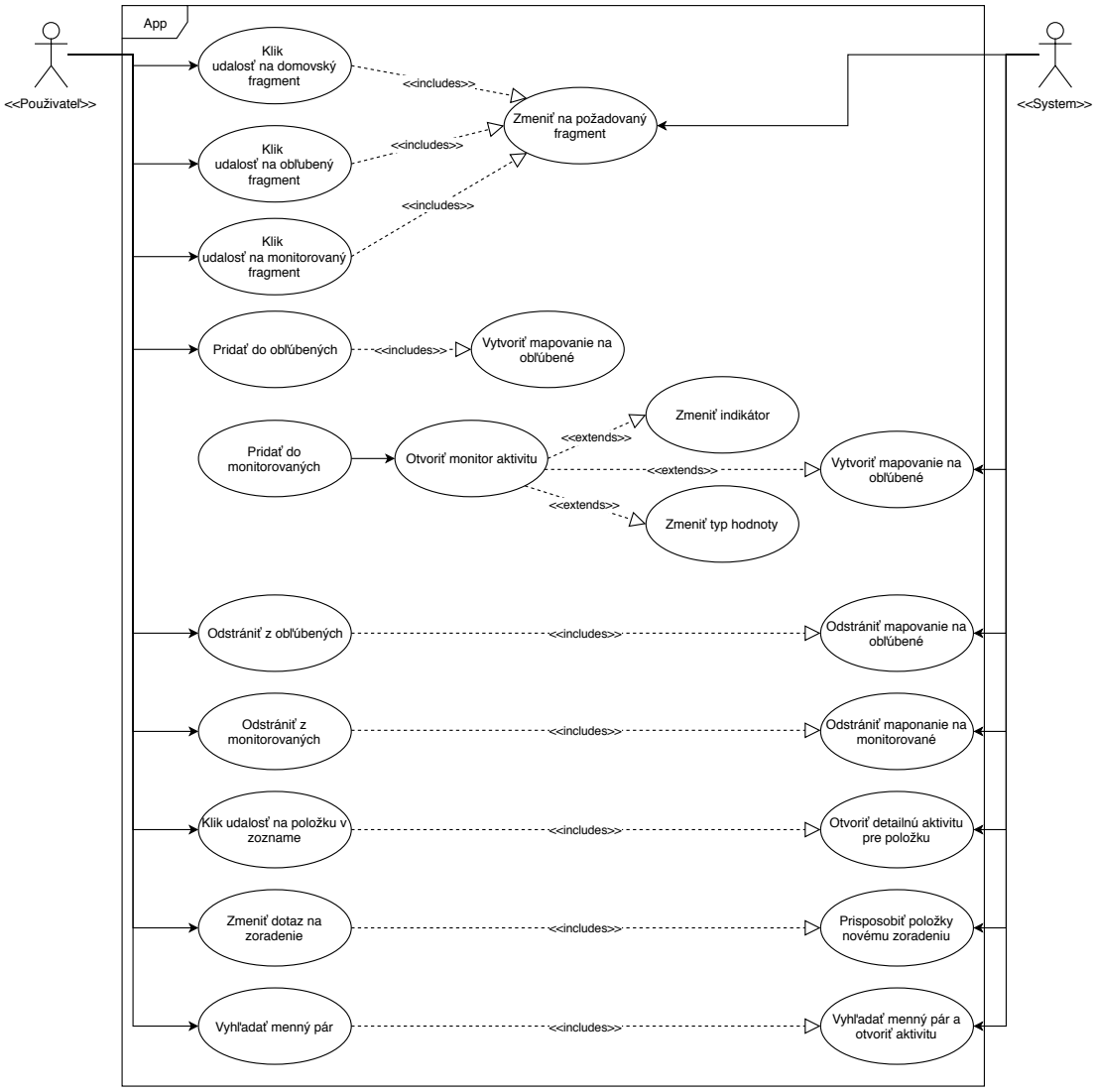
- Stabilné pripojenie na Internet pri štarte aplikácie.
- Správna funkcia firestore databázy.

Ciele

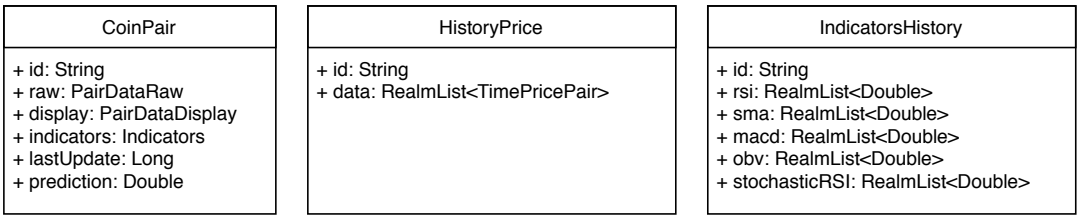
- Získať aktuálne informácie o kryptomenách.
- Byť informovaný pri zmene nastavených indikátorov.

7.2 Dáta

Aplikácia získava dáta 7.2 priamo z firestore databázy. V databáze sú uložené 3 dátové štruktúry CoinPair, HistoryPrice, IndicatorsHistory. O synchronizáciu dát sa stará synchronizačný manažér. Ak nastane zmene v dátach v databáze, aplikácia je o zmene informovaná a sú jej odoslané nové dáta. Ako internú databázu používa aplikácia realm databázu, do ktorej sa ukladajú spracované dáta. Aby bolo aplikácia informovaná o zmene dát, musíme na tabuľky vytvoriť poslucháča (listener). V pozadí databázy sa napájame na sieťový TCP socket a reagujeme na určitý typ udalosti. Ak daná udalosť nastane zavolá sa náš poslucháč ktorý sme za-definovali.



Obr. 7.1: Diagram prípadov použitia - mobilná aplikácia.



Obr. 7.2: Diagram tried dát v aplikácii.

7.3 Hlavná obrazovka

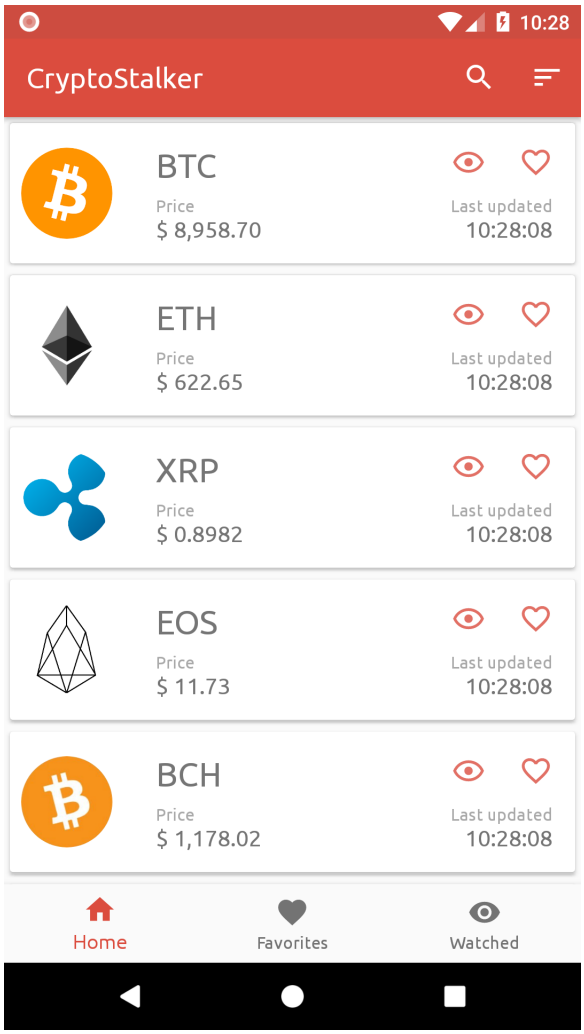
Hlavná aktivita obsahuje kontajner na fragmenty a navigačný bar. Pomocou navigačného baru môžme meniť fragment ktorý sa nachádza v kontajnery. Môžme meniť medzi domovským, obľúbeným a monitorovaným fragmentom. Všetky fragmenty dedia z abstraktného CryptoListFragment 7.1. CryptoListFragment obsahuje dve abstraktné metódy ktoré si každý fragment na hlavnej obrazovke prepíše. Ako viewholder používajú všetky fragmenty CryptoViewHolder s tým rozdielom že obľúbený a monitorovaný fragment používajú viewholder ktorý slúži ako obálka okolo CryptoViewHoldru. CryptoViewHolder obsahuje metódu onBind() práve túto metódu prepisujú WatchedViewHolder a FavoriteViewHolder. Viewholdre rozbalia dáta do formy ktoré dokáže spracovať CryptoViewHolder a zavolajú rodičovskú metódu onBind(). Týmto spôsobom eliminujeme redundantný kód.

```
abstract class CryptoListFragment : Fragment() {
  protected var sort : Sortable? = null
  private var adapter: CryptoListAdapter? = null
  abstract fun getData(): RealmResults<RealmModel>
  abstract fun getViewHolder(): (view: View) ->
  CryptoViewHolder

  override fun onCreate(savedInstanceState: Bundle?)
  {...}
  override fun onCreateView(inflater: LayoutInflater,
                            container: ViewGroup?,
                            savedInstanceState: Bundle?)
  : View? {...}
  override fun onAttach(context: Context?) {
       super.onAttach(context)
       sort = context as Sortable
   }
```

Výpis 7.1: CryptoListFragment.

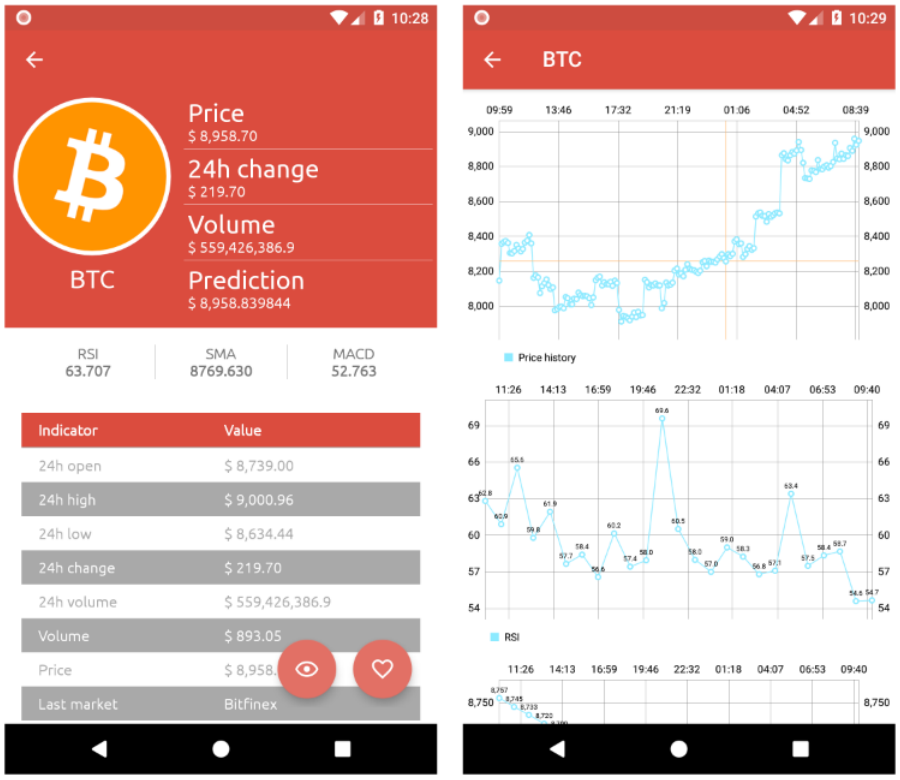
Fragment obsahuje recycleview ktorý obsahuje adaptér. Adaptér spája viewholder s recycleview. Viewholder je bunka jednej položky ktorá je zobrazená v liste. Na obrázku 7.3 je obrazovka ktorú používateľ vidí po zapnutí aplikácie.



Obr. 7.3: Hlavná obrazovka v aplikácii.

7.4 Detail kryptomeny

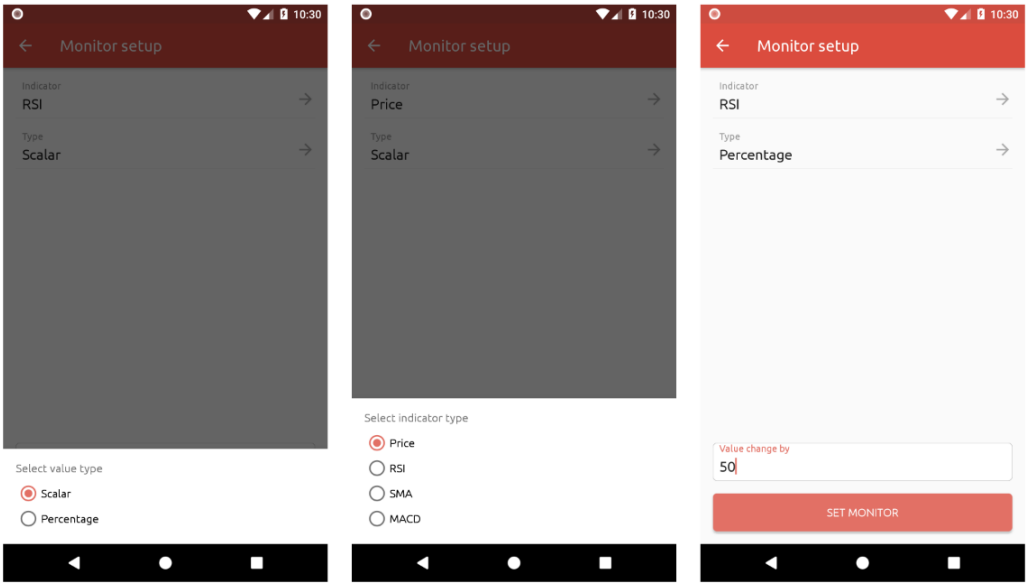
Po kliknutí na položku v zozname sa používateľ dostane na detailnú aktivitu položky 7.4. Pohľad je rozdelený na tri logické celky hlavička, tabuľka, grafy. V hlavičke sú najdôležitejšie informácie o kryptomene a aktuálne technické indikátory. V tabuľke sú uvedené všetky informácie ktoré aplikácia poskytuje. Posledná časť sú grafy kde sú uvedené grafy ceny, RSI, SMA, MACD, OBV. Kryptomenu je možné pridať do obľúbených a monitorovaných aj z detailu pomocou okrúhlych tlačidiel v spodnej časti obrazovky, tieto tlačidla sa pri skrolovaní smerom dole skrývajú aby používateľovi neprekážali pri čítaní informácii a opätovne sa objavujú pri skrolovaní smerom hore.



Obr. 7.4: Detailná obrazovka v aplikácii.

7.5 Monitorovanie kryptomeny

V monitor aktivite 7.5, si používateľ môže nastaviť notifikácie na jednotlivé parametre, má na výber z ceny, RSI, SMA, MACD. Hodnotu zmeny môže nastaviť skalárne alebo percentne. Po vybratí indikátoru a typu používateľ zadá hodnotu a tlačidlom potvrdí nastavenie monitoru. Tým sa vytvorí objekt typu ObservableInterest ktorého štruktúru vidíme na výpise 7.2. Pokiaľ zadaná hodnota nie je platné číslo vypíše sa upozornenie používateľovi aby zadal validné číslo. Objekt sa uloží do realm databázy a ukončí sa monitor aktivita. Pri štarte aplikácie sa inicializuje ObservableManager ktorý dohliada na zmeny ceny a porovnáva ich s ObservableInterest objektmi. Pokiaľ zmena ceny vyhovuje používateľovým nastaveniam je vytvorená notifikácia pomocou NotificationService objektu. NotificationService objekt má za úlohu vytvárať android notifikácie.



Obr. 7.5: Monitor obrazovka v aplikácii

```
@RealmClass
open class ObservableInterest : RealmModel {
    @PrimaryKey
    var id: String? = ""
    var currentValue: Double = 0.0
    var change: Double = 0.0
    var indicator: String = ""
    var type: String = ""
}
```

Výpis 7.2: Trieda ObservableInterest

8 Záver

V práci boli preskúmané témy kryptomien, technických indikátorov, strojového učenia a Android aplikácií. Pomocou znalostí z daných oblastí bola vytvorená mobilná aplikácia s podpornými službami ktoré spolu tvoria systém. Systém je navrhnutý tak, aby odľahčil požívateľské zariadenia, tým šetril batériu a výpočetný výkon mobilného zariadenia. Systém sa skladá z troch hlavných komponentov, ktorými sú Android aplikácia, predikčná služba a Cryptostalker služba. Cryptostalker služba sa stará o získanie dát, spracovanie, transformáciu a komunikáciu systému, služba je napísaná v jazyku Kotline. Služba je vstupným bodom systému. Ako prvé služba odošle požiadavok na externé API, dáta získané z požiadavku sa používajú naprieč celým životným cyklusom aplikácie. Konkrétne výpočty a transformácie na dátach, riadi Cryptostalker služba, ktorá dáta odosiela na predikčnú službu. Po skončení všetkých výpočtov do Firebase databázy. Produkčná verzia je spustená na Ubuntu servere. Okrem hlavného módu, kde je Cryptostalker služba spustená ako hlavná služba v systéme, môže byť Cryptostalker služba spustená v móde, kde vytvorí trénovacie dáta pre predikčnú službu. Predikčná služba je HTTP REST API napísané v jazyku Pythone 3.6 pomocou vývojového balíčka FLASK. REST API má jeden koncový bod, ktorý slúži na predikciu ceny, podľa prijatého typu a predchádzajúcich cien. Predikcia je implementovaná pomocou LSTM neurónovej siete, ktorá na základe predchádzajúcej ceny, predikuje novú cenu. Každá kryptomena má vlastný vytrénovaný model tak, aby bola zaručená integrita predikcie. Produkčná verzia je spustená na Ubuntu servere. Android aplikácia je napísaná v jazyku Kotline a jej úlohou je zobrazovať dáta, ktoré služby nahrali do databázy. Mobilná aplikácia dáta iba zobrazuje a žiadnym spôsobom ich ďalej netransformuje. Technické parametre zobrazované v aplikácií sa radia do dvoch kategórií, a to v reálnom čase a historické dáta. Aplikácia je implementovaná spôsobom, aby používateľovi poskytla dostatok informácii ohľadom, ako aktuálnych dát, tak aj dát historických, ktoré dávajú používateľovi väčší rozhľad, ako sa daná kryptomena vyvíjala. Android notifikácie je možné nastaviť na cenu, RSI, SMA, MACD. Hodnota zmeny môže byť zadaná percentne, alebo na pevne danú hodnotu. Aby bolo ovládanie aplikácie intuitívne, sú dodržiavané zásady materiál designu. V aplikácii je možné jednotlivé zoznamy triediť podľa ceny a volume. Aplikácia obsahuje 200 kryptomien, ktoré sú v aplikácii z databázy obnovované v reálnom čase. Hodnoty sú do databázy nahrávané v konštantnom časovom intervale, ktorý je nastavený na cryptostalker službe. Interval tejto obnovy môže byť od troch sekúnd do pár minút. Pôvodná hodnota je implementovaná na 60 sekúnd. Okrem hlavných parametrov kryptomeny sú obnovované aj technické parametre a predikcia ceny. Pomocou implementovaných funkcií, aplikácia ponúka informácie na uskutočnenie štatisticky správnych rozhodnutí na trhu.

Literatúra

[1] MASSÉ, Mark. *REST API design rulebook*. Sebastopol, CA: O'Reilly, 2012. ISBN 1449310508.

[2] *Realm Database* [online]. San Francisco: Realm, n.d. [cit. 2017-12-13]. Dostupné z: https://realm.io/docs/

[3] GRANT ALLEN AND MIKE OWENS. *The definitive guide to SQLite: [take control of this compact but powerful tool to embed sophisticated SQL databases within your applications!]. 2nd*. ed. New York: Apress, 2010. ISBN 9781430232254.

[4] FREEMAN, Eric, Elisabeth. ROBSON, Kathy. SIERRA a Bert. BATES. *Head First design patterns*. Sebastopol, CA: O'Reilly, c2004. ISBN 0596007124.

[5] *Activity* [online]. California: Google, n.d. [cit. 2017-12-08]. Dostupné z: https://developer.android.com/reference/android/app/Activity.html

[6] *Fragments* [online]. California: Google, n.d. [cit. 2017-12-08]. Dostupné z: https://developer.android.com/guide/components/fragments.html

[7] *Technical Indicators and Overlays* [online]. Washington: StockCharts.com, n.d.[cit. 2017-12-12]. Dostupné z: http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators

[8] *Simple Moving Average - SMA* [online]. New York: investopedia.com, n.d.[cit. 2018-7-5]. Dostupné z: https://www.investopedia.com/terms/s/sma.asp

[9] NARAYANAN, Arvind, et al. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction.* Princeton University Press, 2016.

[10] SCHMIDT, Albrecht. A modular neural network architecture with additional generalization abilities for high dimensional input vectors. Manchester Metropolitan University, Department of Computing, 1996.

[11] CHRIS, Adi. From Perceptron to Deep Neural Nets [online]. 25 December 2017 [cit. 2018-05-13]. Dostupné z: https://becominghuman.ai/from-perceptron-to-deep-neural-nets-504b8ff616e

[12] SHARMA V, Avinash. Understanding Activation Functions in Neural Networks [online]. 30 March 2017 [cit. 2018-05-14]. Dostupné z: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

[13] OLAH, Christopher. Understanding LSTM Networks [online]. 27 August 2015 [cit. 2018-05-16]. Dostupné z: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[14] EDWARDS, Robert D.; MAGEE, John; BASSETTI, WH Charles. Technical analysis of stock trends. CRC press, 2007.

Zoznam symbolov, veličín a skratiek

API	Rozhranie pre programovanie aplikácií
HTTP	Hypertextový prenosový protokol
JSON	JavaScriptový objektový zápis
TCP	Protokol riadenia prenosu
REST	Representational State Transfer
RSI	Index relatívnej sily
SMA	Jednoduchý pohyblivý priemer
MACD	Pohyblivý priemer konvergencie a divergencie
OBV	On-Balance Volume
EMA	Exponenciálny pohyblivý priemer
LSTM	Long Short-Term Memory
RNN	Rekurentná neurónová sieť
CSV	Súbor s hodnotami

Zoznam príloh

A Obsah priloženého DVD

Na priloženom DVD je elektronická verzia bakalárskej práce, zdrojové kódy, dáta použité pri trénovaní neurónovej sieťe, programy na spustenie.

- `/` ..... koreňový adresár priloženého DVD
  - `1. Praktická časť` ..... zdrojový kód a dáta
    - `Data`
      - `LSTM-Models` ..... vytrénované LSTM modely
      - `LSTM-TrainingCSV-Data` ..... csv dáta použité pri trénovaní
    - `Release`
      - `CryptoStalkerAndroidApp` ..... android aplikácia určená na inštalovanie
      - `CryptoStalkerService` ..... cryptoStalker služba na spustenie
      - `PredictionService` ..... predikčná služba na spustenie
    - `Source-code` ..... zdrojový kód k programom
  - `2. Elektronická verzia` ..... elektronická verzia bakalárskej práce
    - `bakalárska-práca.pdf`
  - `3. Návod`