VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

INFORMAČNÍ SYSTÉM PRO AUTOMATIZOVANÉ OBCHODOVÁNÍ NA BURZE KRYPTOMĚN

INFORMATION SYSTEM FOR AUTOMATED TRADING AT CRYPTO-CURRENCY EXCHANGE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MIROSLAV KLOBÁSKA**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. VLADIMÍR BARTÍK, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

146134

Ústav:	Ústav informačních systémů (UIFS)
Student:	**Klobáska Miroslav**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**Informační systém pro automatizované obchodování na burze kryptoměn.**
Kategorie:	Informační systémy
Akademický rok:	2022/23

Zadání:

1. Prostudujte si základní způsoby obchodování na burzách kryptoměn a vyberte vhodnou burzu z pohledu vhodné výše poplatků a dostačujícího aplikačního rozhraní pro realizaci obchodních algoritmů.
2. Prostudujte principy tvorby informačních systémů a vhodná vývojová prostředí.
3. Analyzujte požadavky a navrhněte informační systém pro automatizované obchodování na burze, který umožní uživateli výběr a sledování chování obchodních algoritmů.
4. Implementujte navržený informační systém a otestujte jeho funkčnost.
5. Zhodnoťte dosažené výsledky a další možnosti rozšíření tohoto projektu.

Literatura:

- Bulkowski, T. N.: Encyclopedia of Chart Patterns, 3rd Edition. Wiley, 2021. ISBN 978-1-119-73968-5.
- Wilder, J. W.: New Concepts in Technical Trading Systems. Trend Research, 1978. ISBN 978-0-894-59027-6.
- *Connors*, L., Alvarez, C.: An *Introduction* to ConnorsRSI (*Connors Research Trading Strategy Series*), *2nd* edn. *Connors Research*, Jersey City (2014)

Při obhajobě semestrální části projektu je požadováno:
Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Bartík Vladimír, Ing., Ph.D.**
Vedoucí ústavu:	Kolář Dušan, doc. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	10.5.2023
Datum schválení:	24.10.2022

Abstrakt

Cílem této práce je vytvořit informační systém, který bude v reálném čase provádět obchody na kryptoměnové burze za účelem dosažení finančního zisku. Konkrétní algoritmy, které jsou navrženy tak, aby realizovaly zisk, v práci nejsou prezentovány.

Práce se zabývá návrhem a tvorbou programových modelů, které vymezují zaměření obecného obchodního algoritmu a dále integrací těchto modelů v rámci informačního systému.

Výsledkem této práce je informační systém, který provozuje konfigurovatelnou množinu duálních obchodních algoritmů s úplnou správou prostředků, resp. instance těchto algoritmů vytvořené konkrétními uživateli.

Jedná se přitom o jeden konkrétní pohled na automatizované obchodování. Nejde ani tak o ucelený produkt, jako spíše o infrastrukturu, která nabízí alternativu k veřejně dostupným, a často omezeným, systémům pro uživatele.

Abstract

The aim of this bachelor thesis is to create an information system that will execute trades on a cryptocurrency exchange in real-time, with the goal of achieving financial profit. Specific algorithms designed to generate profit are not presented in the thesis.

The thesis focuses on design and development of program models that are subset to presented general trading algorithm and on the integration of these models within the information system.

The result is an information system that operates users' trading algorithm instances of a configurable set of dual trading algorithms with complete asset management.

This is a specific view of automated trading, not a complete product, but rather an infrastructure that offers an alternative to publicly available, and often limited, systems for users.

Klíčová slova

Informační systém, Automatizované obchodování, Real-time obchodování, Kryptoměny, Obchodní algoritmus, Technická analýza

Keywords

Information system, Automated trading, Real-time trading, Crypto-currencies, Trading algorithm, Technical analysis

Citace

KLOBÁSKA, Miroslav. *Informační systém pro automatizované obchodování na burze kryptoměn.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vladimír Bartík, Ph.D.

Informační systém pro automatizované obchodování na burze kryptoměn

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Vladimíra Bartíka PhD. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Miroslav Klobáska
7. května 2023

Poděkování

Dovoluji si poděkovat panu Ing. Vladimíru Bartíkovi Ph.D. za velice cenné podněty a připomínky při tvorbě práce a zároveň za poskytnutí maximální volnosti, díky které jsem mohl k řešení práce přistupovat kreativně.

Obsah

Kapitola 1

Úvod

Cílem této práce je navrhnout a implementovat informační systém, který bude realizovat obchodní algoritmy na burze kryptoměn za účelem dosažení finančního zisku. Předpokládaná frekvence vykonávaných obchodů je 1 den až 1 hodina.

Smysl této práce se opírá můj soukromý výzkum obchodních algoritmů. Tato práce nicméně nebude prezentovat žádný obchodní algoritmus zaměřený na finanční zisk, ani jiné finanční či investiční rady, neboť:

1. se jedná o vysoce rizikovou oblast,
2. nejednalo by se o žádný veřejný či vědecký přínos a
3. výkonnost obchodních algoritmů zpravidla klesá se zvyšujícím se množstvím alokovaných prostředků.

Práce bude tedy popisovat především technickou část, kterou je zapotřebí vyřešit, aby bylo možné realizovat obchodní algoritmy v reálném čase.

V kapitole 2 představím obecné schéma, které si představuji pod pojmem *obchodní algoritmus*. Dále zavedu jednoduchou klasifikaci obchodních algoritmů, kterou budu využívat jako jistou formu omezení při tvorbě programových modelů pro realizaci konkrétních obchodních algoritmů.

V rámci kapitoly 3 krátce popíši data, která bude informační systém analyzovat. Představím obvyklé typy objednávek, které lze na kryptoměnových burzách vytvářet. Nakonec uvedu kritéria pro výběr burzy, kterou bude integrovat informační systém, a na základě aktuálně dostupných informací provedu její výběr.

Kapitola 4 se zabývá se jedním z možných způsobů, který je možné využít pro analýzu tržní situace – technickou analýzou. V rámci této kapitoly představím některé základní a obecně známé technické indikátory, a dále zavedu koncepty *kombinovaných technických indikátorů*. Všechny technické indikátory popisuji pomocí jednotné matematické notace, která přímo odpovídá jejich definici v rámci programového modelu pro výpočet indikátorů, který následně představím. Zamyslím se také nad *sítí technických indikátorů*, což je jakási struktura závislostí, která při kombinaci technických indikátorů přirozeně vzniká.

V kapitole 5 představím problematiku implementace programového modelu pro výpočet technických indikátorů po syntaktické a sémantické stránce a popíši vlastní implementované řešení.

V rámci kapitoly 6 dále nastíním problematiku simulace obchodních algoritmů. Zavedu zde dále *bezstavový model obchodního algoritmu*, který omezuje koncept obecného obchod-

ního algoritmu, čímž zároveň vytváří příhodné vlastnosti pro jeho realizaci. Zavedený model rozdělím na dílčí komponenty, které budou zároveň implementovány v informačním systému.

Kapitola 7 slouží jako krátké zamyšlení nad smyslem vytvářeného informačního systému, pokud bychom jej porovnávali s reálnými a veřejně dostupnými produkty, které se věnují stejné nebo obdobné problematice. V rámci této kapitoly specifikuji také požadavky na informační systém.

V kapitole 8 krátce popíši požadavky na výběr technologií s ohledem na charakter informačního systému. Na základě těchto požadavků uvedu nejdůležitější vybrané technologie, frameworky a knihovny, na nichž bude systém stavěn.

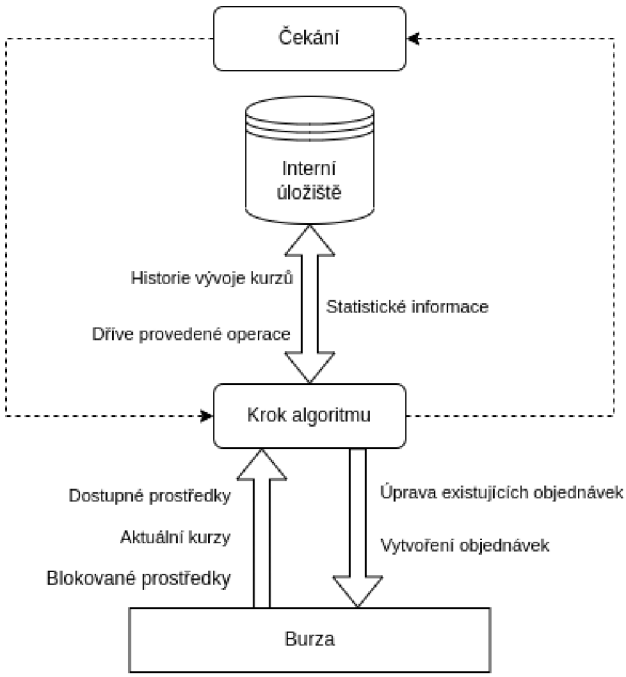
V rámci kapitoly 9 popíši návrh informačního systému, rozeberu jednotlivé moduly serverové aplikace a zmíním zajímavější části implementace. Stručně se zmíním také o verifikaci informačního systému, přestože se jednalo pravděpodobně o primární činnost při jeho realizaci. Dále popíši implementaci uživatelského rozhraní, přičemž se zaměřím především na výčet knihovních závislostí, které byly využity při implementaci.

Na závěr zhodnotím v kapitole 10 cíle, kterých se podařilo v rámci práce dosáhnout, a popíši aktuální omezení systému, včetně možností rozšíření.

Kapitola 2

Obchodní algoritmus

V rámci tého práce budu často používat termín „obchodní algoritmus“. Synonymem pro obchodní algoritmus by mohla být „obchodní strategie“ (z angl. „trading strategy“), pokud bychom odmysleli aspekty, které souvisí s technickou realizací automatizovaného obchodníka.



Obrázek 2.1: Schéma obecného obchodního algoritmu

Obecný obchodní algoritmus definuji jako plně specifikovaný deterministický nekonečný postup, který řídí finanční portfolio – rozhoduje o nákupu a prodeji omezeného počtu aktiv. Důležitou vlastností obchodního algoritmu je, že jeden jeho krok je specifikovaný konečným způsobem. Pro dosažení potřebného efektu algoritmu je poté zapotřebí tento krok pravidelně opakovat. V době provádění kroku musí mít obchodní algoritmus k dispozici:

- informaci o aktuálním množství spravovaných prostředků,
- informace o aktuálních převodních kurzech směnitelných dvojic aktiv a jejich historii,

- statistické informace, které vyžaduje algoritmus analýzu trhu[1],
- informace o případných závazcích, které omezují nakládání s prostředky (např. nedokončené burzovní objednávky).

Všechny tyto informace musí algoritmu poskytnout burza na vyžádání. Algoritmus si dále může vést interní evidenci výše obdržených informací, odvozených znalostí, a provedených operací.

Každý krok algoritmu analyzuje aktuálně dostupné informace a na jejich základě může rozhodnout o směně prostředků. Rozhraní pro směnu prostředků musí poskytnout burza. Směna prostředků se v burzovním prostředí nazývá objednávka a existuje celá řada typů objednávek, které lze ke směně využít. Součástí návrhu a definice algoritmu je výběr konkrétních typů objednávek, které budou v určitých situacích využívány.

2.1 Algoritmy podle počtu spravovaných aktiv

Obchodní algoritmy si dovolím rozdělit podle počtu spravovaných prostředků na *duální* a *vícenásobné*.

Duální obchodní algoritmy spravují vždy právě 2 prostředky. V případě modelování duálního obchodního algoritmu budu vybírat 2 různé prostředky takovým způsobem, aby hodnota jednoho prostředku se v čase často měnila a hodnota druhého prostředku byla pokud možno neměnná. Prostředek s nestabilní hodnotou budu poté nazývat *volatilní*, a prostředek se stabilní hodnotou budu nazývat *stabilní*. Příkladem může být algoritmus, který bude spravovat dvojici prostředků (BTC, USDT), kde BTC je *volatilní* prostředek a USDT je *stabilní* prostředek. Takový algoritmus bude využívat *volatilní* prostředek jako investiční médium a *stabilní* prostředek jako udržitel hodnoty.

Vícenásobné obchodní algoritmy spravují vždy minimálně 3 prostředky. Všimněte si, že modelování, simulace, analýza i realizace takového algoritmu bude zákonitě složitější. Příklad reálného algoritmu, se kterým jsem se setkal, může být tzv. *rebalancing strategy* [17], kterou často využívají investiční společnosti k řízení finančního portfolia. Tento algoritmus se stará o zachování požadovaného poměru hodnoty prostředků v čase.

V rámci této práce se budu věnovat realizaci výhradně duálních algoritmů, byť informační systém bude do budoucna rozšiřitelný o vícenásobné algoritmy.

2.2 Algoritmy podle způsobu správy prostředků

Obchodní algoritmy si dále dovolím rozdělit podle způsobu, který je využit pro správu prostředků.

Úplná správa prostředků spočívá uložení veškeré hodnoty, kterou algoritmus disponuje, v rámci maximálně 1 aktiva. Toto aktivum vybírá algoritmus na základě analýzy tržní situace. Úspěšnost takového algoritmu tedy závisí převážně na přesnosti, s jakou algoritmus dokáže odhadnout budoucí vývoj ceny[2]. Výhodou tohoto typu algoritmu je, že je možné od sebe maximálně oddělit modul vykonávající rozhodnutí a modul, který tato rozhodnutí realizuje, což umožňuje výrazným způsobem optimalizovat realizaci velkého množství instancí takového algoritmu a zároveň lépe verifikovat definici takového algoritmu.

[1] Informance, které nejsou zřejmé z historického vývoje cenových kurzů – např. počet provedených objednávek, objem obchodovaných prostředků, atd.

[2] Úspěšnost dále může záviset například na výši burzovních poplatků a na „nerozhodnosti“ algoritmu.

Částečná zpráva prostředků spočívá v přerozdělení hodnoty, kterou algoritmus disponuje, mezi více aktiv. Tato aktiva i poměr přerozdělení hodnoty určuje algoritmus na základě analýzy tržní situace. Tento způsob správy prostředků je konzervativnější než v předchozím případě a nachází časté využití u investičních produktů v reálném světě. Příkladem může být spoření (pravidelné vklady a nákupy *volatilního* aktiva) nebo rebalancování portfolia. Je zřejmé, že v případě ideální tržní analýzy nelze nikdy přesáhnout úspěšnost *úplné správy prostředků*.

V rámci této práce se budu věnovat realizaci výhradně algoritmu s *úplnou správou prostředků* a na jeho základě zavedu *bezstavový model algoritmu* v kapitole 6.2.

Kapitola 3

Kryptoměnová burza

Kryptoměnové burzy jsou soukromé společnosti, které poskytují prostředí pro směnu kryptoměn a fiat měn, a dále typicky řadu investičních či obchodních nástrojů.

Mezi hlavní úkoly kryptoměnové burzy patří zajištění bezpečného a stabilního prostředí pro uchování prostředků a jejich směnu. Toto prostředí by mělo být v ideálním případě neustále dostupné. Pro obchodníky je dále výhodné, pokud se na burze nachází dostatečné množství likvidity – dostatek rezervovaných prostředků pro realizaci obchodů snižuje náhlé lokální výkyvy tržní ceny, které mohou mít za následek ztrátu[1]. Poslední kritérium, které v rámci této práce budu zvažovat, je dostatečné aplikační rozhraní pro realizaci obchodního algoritmu.

Pro udržitělné fungování stanovují burzy poplatky. Mezi hlavní zdroj příjmu typicky patří poplatek z provedení obchodu (angl. „comission fee“). Dále se občas vyskytují poplatky například za vložení fiat prostředků na burzu či výběr fiat prostředků z burzy, poplatek za převedení kryptoměny na jiný účet, atd.

3.1 Burzovní data

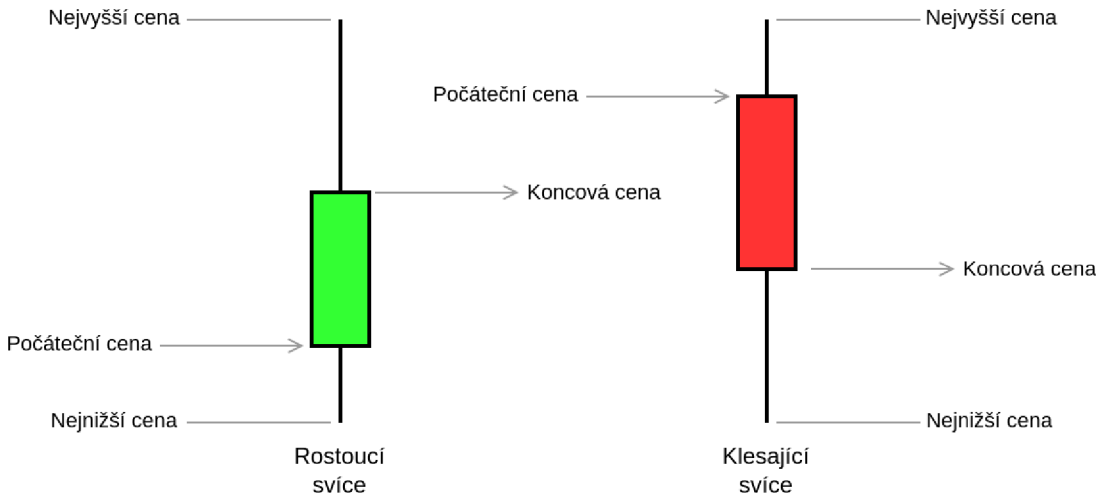
Hlavním zdrojem vstupních dat pro obchodní algoritmus, stejně jako pro reálné obchodníky, je vývoj cenového kurzu mezi dvěma prostředky v čase. Vzhledem k tomu, že se cenové kurzy mohou v čase rychle měnit, a prosté vzorkování kurzu by bylo náročné na objem dat a zároveň by způsobovalo nepřesnosti při analýze, ustálilo se v této oblasti využití tzv. *svícových grafů* (angl. „candlestick charts“).

Svícové grafy jsou tvořeny *svícemi*, přičemž každá svíce popisuje chování cenového kurzu v rámci vymezeného časového intervalu [15]. Datový model *svíce* sestává vždy z:

- počáteční ceny,
- koncové ceny,
- nejnižší ceny a
- nejvyšší ceny.

Důležitou informací je dále délka vymezeného časového intervalu a zároveň umístění svíce v čase.

[1] Příkladem ztráty by mohla být nežádoucí aktivace (či cena aktiva v době provedení obchodu) prodejní objednávky, jejímž cílem je omezit riziko v případě náhlého pádu ceny aktiva.



Obrázek 3.1: Vizualizace svíce

Obrázek 3.2: Příklad svícového grafu – vývoj kurzu BTC/USDT v říjnu až prosinci 2022[2]

Svícové grafy bývají často doplněny dalším grafem, který zobrazuje celkové obchodované množství prostředků (angl. „volume“) s ohledem na jednotlivé svíce. Burzovní data poté mohou obsahovat další dodatečné informace, jako například počet provedených obchodů a další. Všechy dostupné informace může využít obchodník k analýze situace na trhu.

3.2 Způsoby obchodování

Burzy obchodníkům zpravidla poskytují řadu nástrojů, které mohou využít k obchodování. Dostupné nástroje se navíc mohou lišit výběrem burzy či výběrem tržního prostředí (běžné obchodování, obchodování na páku, kontrakty, atd.). Těmto nástrojům se říká *objednávky*

[2]Obrázek byl pořízen na webové stránce služby Trading View.

nebo také *příkazy*. V rámci této kapitoly se zaměřím na nejběžnější typy objednávek na burzách, přičemž některé z nich bude využívat informační systém a některé budou moci být přidány později, budou-li to vyžadovat realizované modely obchodních algoritmů.

Společným znakem všech typů objednávek je tzv. *strana* (angl. „side“) – tj. volba, zda se jedná o nákup nebo prodej volatilního aktiva[3]. Objednávky pro nákup a prodej jsou navzájem *opačné* – aby mohl některý obchodník nakoupit volatilní aktivum, musí jej jiný obchodník prodat. Dalším společným znakem všech objednávek je specifikace objemu buď *volatilního*, nebo *stabilního* prostředku, který bude realizací objednávky odečten, nebo přičten[4] na burzovním účtu.

3.2.1 Market objednávka

Nejjednodušším typem objednávky je *market* objednávka, která umožňuje provést okamžitý nákup, resp. prodej za aktuálně nejvýhodnější cenový kurz stanovený opačnými *limit* objednávkami [4]. Tento kurz se bude blížit *aktuálnímu cenovému kurzu*[5].

Nevýhodou *market* objednávky je fakt, že negarantuje konkrétní cenový kurz – v případě nedostatku likvidity by mohlo dojít k výrazné změně kurzu nepříznivým směrem.

Burzy se zároveň typicky snaží omezit obchodníky v používání *market* objednávek. Děje se tak pomocí zvýšeného poplatku za realizaci objednávky, které spadají do tzv. kategorie *Taker* – třídy objednávek, které se nepodílí na tvorbě cenového kurzu.

3.2.2 Limit objednávka

Důležitým základním typem objednávky je *limit* objednávka. Jedná se o objednávku, která specifikuje minimální přijatelný kurz, za který může být objednávka provedena [4]. Tento fakt zakládá 2 nové vlastnosti:

1. objednávka může být provedena pouze částečně, což se může stát typicky u objednávek zahrnujících velký objem prostředků, pokud se *aktuální cena* stane nejvýhodnější pro opačné objednávky pouze na krátký časový okamžik; a

2. objednávka může být provedena i za výhodnější cenový kurz, než je stanovený kurz *limit*, v případě, že dojde ke vzájemnému překřížení *limit* kurzů *opačných* objednávek – cenový kurz v takovém případě určuje burzovní algoritmus a většinou se blíží *aktuálnímu kurzu*.

Výhodou tohoto typu objednávky je garance požadovaného kurzu. Objednávku lze například využít pro prodej *volatilního* aktiva za předem stanovený kurz, který je vyšší než *aktuální kurz*, nebo k nákupu *volatilního* aktiva za předem stanovený kurz, který je nižší než *aktuální kurz* – tímto způsobem je možné například optimalizovat časové možnosti obchodníka i výpočetní zátěž informačního systému pomocí předpřipravení obchodů.

[3] Pro vysvětlení typů objednávek budu využívat zavedené pojmy, které maximálně korespondují s účelem informačního systému, který budu realizovat. Terminologie z pohledu burzy však může být o něco složitější, neboť na burze lze obchodovat i dvojice typu (stabilní, stabilní) či (volatilní, volatilní) aktivum – můžeme poté hovořit o dvojicích (base, quote), tedy o základním aktivu a nabízené měně.

[4] Možnost specifikace objemu dle prostředku je typicky ovlivněna typem objednávky a konkrétní burzou. Například Binance umožňuje volit množství *volatilního* i *stabilního* prostředku pouze u Market objednávek. Ostatní objednávky vyžadují specifikaci objemu výhradně formou *volatilního* prostředku.

[5] Za *aktuální kurz* pro účely vysvětlení považujme průměr požadovaných cenových kurzů dvou *opačných limit* objednávek, které nejsou vzájemně obchodovatelné a jejich kurzy jsou vzájemně nejbližší.

Burzy se zpravidla snaží podporovat obchodníky ve využívání *limit* objednávek zařazením do tzv. kategorie *Maker*, neboť tím posilují likviditu trhu, zvyšují transparentnost, a omezují extrémní volatilitu trhu. Existence *limit* objednávek zároveň umožňuje obchodníkům analyzovat aktuální poptávku či nabídku a na základě toho činit obchodní rozhodnutí.

3.2.3 Stop-limit objednávka

Stop-limit objednávka je nástroj, který umožňuje reagovat na riziko, že se cenový kurz bude pohybovat opačným směrem, než předpokládáme.

Objednávka *stop-limit* [2] vyžaduje definici dvou cenových hladin:

1. *stop* ceny – tj. cena, která po dosažení v kontextu dané *strany* objednávky[6] aktivuje tuto objednávku; a

2. *limit* ceny – tj. minimální přijatelný kurz pro realizaci objednávky platný po její aktivaci.

Typické využití *stop-limit* objednávky spočívá ve specifikaci cenové hladiny (*stop*) pro ukončení aktuální pozice, přičemž je realizována ztráta. Tuto cenovou hladinu je tedy zapotřebí volit s rozvahou – aby v případě jejího dosažení existovala minimální pravděpodobnost, že by se cenový kurz vrátil v krátké době opět na předchozí úrovně, kde jsme stanovili náš předpoklad pro vývoj ceny.

Cenovou hladinu *limit* je také zapotřebí volit s rozvahou, a to především na trzích s nízkou likviditou – čím je kratší rozpětí mezi *stop* a *limit* cenou, tím je větší šance, že se nám nepodaří celou objednávku realizovat, a tedy budeme dále držet pozici, kterou již držet nechceme, zatímco se aktuální cenový kurz může vyvíjet nepříznivým směrem. Opačným problémem je příliš velké rozpětí mezi *stop* a *limit* cenou, kde naopak hrozí realizace objednávky za nepříznivý kurz. Závažnost obou problémů se pochopitelně prohlubuje se zvětšujícím se objemem vlastní objednávky.

3.2.4 Stop objednávka

Stop objednávka je nástroj, který slouží ke stejnému účelu jako *stop-limit* objednávka. Rozdílem zde je, že nedochází ke specifikaci *limit* ceny – po dosažení *stop* ceny dojde k aktivaci *market* objednávky [13].

Některé burzy tuto objednávku neposkytují, neboť při její aktivaci hrozí realizace objednávky za nevýhodný kurz. Zároveň v případě velmi prudkých výkyvů cenového kurzu na trhu s nižší likviditou dochází k dalšímu posílení takovýchto výkyvů.

3.2.5 OCO objednávka

One Cancels the Other objednávka [3] kombinuje vlastnosti *limit* a *stop-limit* objednávky. Tento nástroj vyžaduje specifikaci 3 cenových hladin:

1. *target* hladinu, která stanoví hladinu, na které dojde dojde k plnění *limit* objednávky;

2. *stop* hladinu, která při dosažení v případě nepříznivého vývoje trhu spustí příkaz *stop-limit*;

[6] Pro nákup *volatilního* aktiva je zapotřebí *stop* cenu volit vyšší než je aktuální kurz. Pro prodej je zapotřebí naopak volit *stop* cenu nižší než aktuální kurz.

3. *limit* hladinu, která specifikuje nejhorší možný přijatelný kurz v případě aktivace příkazu *stop-limit*.

Tento nástroj je možné využít například v rámci obchodních algoritmů, které potřebují co nejrychleji reagovat na vývoj kurzu oběma směry.

3.2.6 Trailing stop

Příkaz *trailing stop* [5] je podobný *stop-limit* objednávce. Specifikuje 3 parametry:

1. *aktivační* cenovou hladinu,
2. odchylku *trailing delta*,
3. cenovou hladinu *limit*.

Parametr *aktivační* cenové hladiny je volitelný. Pokud je tato hladina stanovena, k aktivaci *trailing stop* objednávky dochází teprve, pokud je *aktuální cenový* kurz shodný či na okamžik výhodnější než *aktivační* cenová hladina. V opačném případě k aktivaci dochází ihned po vytvoření objednávky na burze.

Po aktivaci *trailing stop* objednávky dochází k monitorování ceny trhu, přičemž burza eviduje nejvýhodnější cenový kurz od doby aktivace až do doby vyplnění či zrušení objednávky. Povinný parametr *trailing delta* stanovuje maximální procentuální odchylku nepříznivým směrem od evidovaného nejvýhodnějšího cenového kurzu. V případě, že aktuální odchylka dosáhne maximální stanovené odchylky, poté dojde k aktivaci *limit* objednávky na povinně stanovené cenové hladině *limit*.

Tento nástroj je možné využít například v rámci obchodních algoritmů, které se snaží zachytit prudké výkyvy trhu jedním směrem.

3.3 Výběr burzy

V rámci této kapitoly porovnám kryptoměnové burzy na základě aktuálně dostupných informací. Porovnání budu činit na základě:

1. co nejnižšího poplatku za realizaci objednávky,
2. dostatečného aplikačního rozhraní pro vytváření a správu všech výše uvedených typů objednávek,
3. výše dostupné likvidity na burze.

Burza	Poplatek Maker/Taker	Objednávky
Bitpanda Pro	0.1500 %/0.2500 %	Market, Limit, Stop Limit
Bitstamp	0.3000 %/0.4000 %	Neposkytuje OCO
HitBTC	0.2000 %/0.2000 %	Market, Limit, Stop Limit
Kraken	0.1600 %/0.2600 %	Neposkytuje OCO
Kucoin	0.1000 %/0.1000 %	Všechny uvedené
Binance	0.1000 %/0.1000 %	Všechny uvedené
Coinbase	0.6000 %/0.4000 %	Market, Limit, Stop Limit
Crypto.com	0.0750 %/0.0750 %	Všechny uvedené
Liquid	0.1000 %/0.1450 %	Neposkytuje OCO
OKX	0.0800 %/0.1000 %	Všechny uvedené
Poloniex	0.1450 %/0.1550 %	Market, Limit, Stop Limit

Tabulka 3.1: Srovnání kryptoměnových burz k datu 11.12.2022[7]

Z pohledu výše poplatku je aktuálně nejlepší volbou platforma Crypto.com. Nevýhodou je, že dle aktuální dokumentace API [11] neumožňuje tato burza vytvářet automaticky jiné než *market* a *limit* objednávky. Tento problém by bylo možné adresovat zavedením aktivního monitoringu aktuálních kurzů a speciálního typu objednávky na straně informačního systému a nebo integrovat existující řešení, které takovou funkcionalitu implementuje (například projekt Crypto.com Stop-Loss Bot [18]). Přestože je tento problém řešitelný, přenesl by důležité odpovědnosti a zátěž na stranu informačního systému, a tím by do systému vnesl potenciální chybovost. Zároveň by nebylo nikdy možné dosáhnout stejné rychlosti provedení objednávek. Z těchto důvodu burzu Crypto.com, ani další burzy s podobným nedostatkem, nebudu integrovat.

Druhou nejlepší burzou z pohledu výše poplatku je OKX. Tato burza zároveň umožňuje tvorbu všech výše uvedených typů objednávek prostřednictvím REST API. Tvorba objednávek konkrétně je v rámci rozhraní rozdělena na „standardní objednávku“ (*market, limit, stop-limit*) a „objednávku řízenou algoritmem“ (*oco*, *trailing order*, a další) [16].

Třetí až čtvrtou nejlepší burzou z pohledu výše poplatku je burza Kucoin a burza Binance. Burza Kucoin neumožňuje přes API vytvářet objednávky typu *oco*, tedy tuto burzu nebudu integrovat.

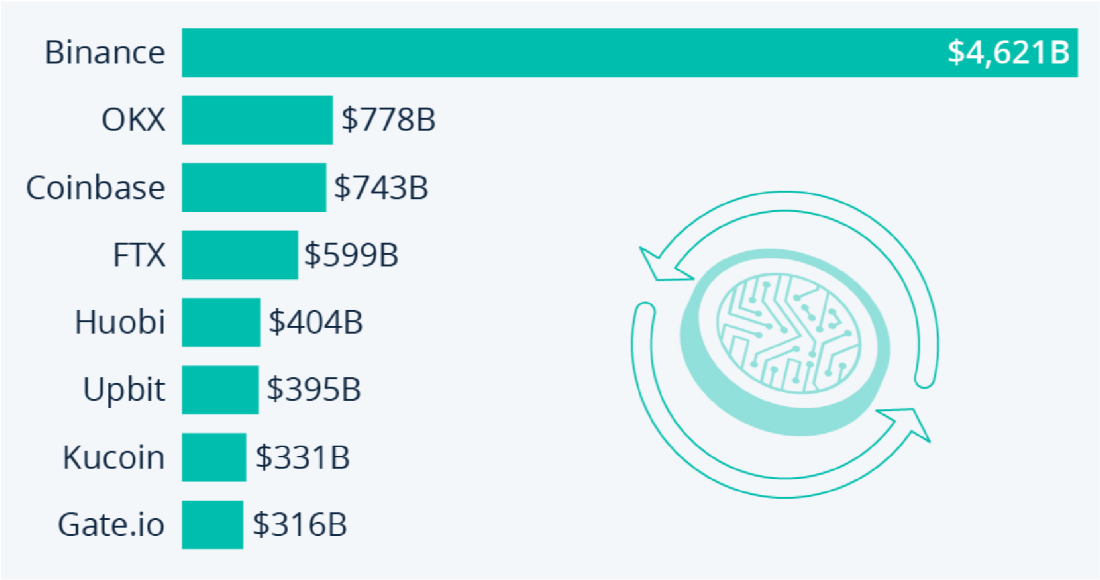
Burza Binance umožňuje dle dokumentace API [6] vytváření všech výše uvedených objednávek. Musím tedy rozhodnout mezi integrací burzy OKX, a nebo Binance. S ohledem na fakt, že burza Binance oproti OKX za poslední rok zahrnovala téměř 6násobek obchodovaného objemu aktiv, dávám zde přednost Binance. Z mojí strany se jedná z části o subjektivní rozhodnutí na základě předpokladů:

- že v časovém horizontu několika roků bude na Binance dostupná vyšší likvidita a
- že burza s větší kontrolou nad trhem dokáže zajistit bezpečnější prostředí pro uživatele.

Toto rozhodnutí činím v době, kdy před necelými 2 měsíci došlo ke krachu čtvrté největší kryptoměnové burzy z pohledu obchodovaných prostředků, FTX.

[7]Poplatky za provedení objednávky byly vybírány z ceníků, které jsou platné pro dvojici (BTC, USD) (a podobné) s ohledem hodnotu portfolia v rozsahu 100000-1000000 CZK, přičemž byla upřednostňována horší varianta poplatku a dočasné slevy nebyly brány v potaz.

Tabulka obsahuje výčet objednávek, které burzy inzerují a poskytují uživatelům prostřednictvím UI rozhraní. Podporu objednávek v rámci API burzy běžně neinzerují.



Obrázek 3.3: Obchodovaný objem na kryptomenových burzách za rok k datu 13.12.2022 [21]

V každém případě, informační systém bude vytvářen takovým způsobem, aby v budoucnosti bylo možné integrovat další burzu, například OKX, a poskytnout tak uživatelům výběr dle vlastní preference.

Kapitola 4

Technická analýza

Jedním způsobem, s jakým lze přistoupit k analýze situace na trhu, je využití tzv. „technické analýzy“. Jedná se o soubor metodik, které se snaží na základě historických burzovních dat odhadnout budoucí vývoj ceny aktiv. Technická analýza tedy nezahrnuje studium ekonomických, společenských a dalších aspektů, které mají na vývoj zásadní vliv [14].

Do technické analýzy lze zahrnout řadu metod analýzy a interpretace burzovních dat. Jedním z nich je využití tzv. „technických indikátorů“. Jedná se o způsob, který budu preferovat v rámci této práce, především z důvodu jednoduchosti implementace a jednoznačnosti, díky které lze realizovat maximálně objektivní obchodní rozhodnutí.

Alternativním prostředkem pro analýzu by mohla být rodina metod „Price Action“, kterou často využívají reální obchodníci. V tomto případě se jedná částečně o subjektivní přístup k analýze [7], ve kterém se obchodníci snaží identifikovat tvary na svícovém grafu [8], a pomocí nich následně predikovat budoucí vývoj. Její implementace na počítači by byla řádově složitější a důvody pro vykonání obchodních rozhonutí nemusí být tolik jednoznačné. Zároveň se kloním spíše k tomu názoru, že obchodní rozhonutí vykonávaná automatem by měla být pokud možno co nejvíce transparentní, což nemusí vždy dobře korelovat se subjektivními přístupy.

4.1 Technický indikátor

Technický indikátor je obecně matematický vzorec, který se snaží z historických burzovních dat (nebo jiných indikátorů) odvodit nějakou informaci.

V rámci této práce se budu dívat na technický indikátor jako na funkci. Vstupem pro výpočet této funkce bude nezáporný počet signálů. Výstupem této funkce bude signál.

Za *signál* budu považovat diskrétní posloupnost číselných hodnot datového typu s plovoucí řádovou čárkou (*double*), přičemž jedna hodnota posloupnosti vždy odpovídá jedné kompletní svíci předem zvoleného rozsahu časové periody. Za validní hodnoty signálu budu považovat konečná čísla a hodnotu *NaN*, která bude odpovídat neinicializované hodnotě. Bude-li do indikátoru vstupovat neinicializovaná hodnota signálu vyžadovaná pro výpočet, výstupní hodnotou indikátoru musí být opět neinicializovaná hodnota[1].

[1] Indikátory budou značeny tak, že budou začínat velkým písmenem (např. *SMA*, *S*, *Ternary* atd.). Parametr zastupující libovolný indikátor bude sestávat z jednoho velkého písmena (např. *S*). Konstantní parametry a proměnné budou značeny malými písmeny. *Dolní index* indikátoru bude značit výstupní hodnotu signálu indikátoru, která odpovídá vybrané svíci v minulosti: 0 aktuální svíci, 1 předchozí svíci, atd.

4.2 Základní technické indikátory

V rámci této kapitoly uvedu výčet některých základních a obecně známých technických indikátorů a nastíním jejich význam. Nejedná se však ani o úplný výčet, ani o výčet, který by obsahoval kompletní sadu nutnou pro co nejefektivnější analýzu trhu. Platí, že mnohem zajímavějšího chování lze docílit pomocí jejich kombinací, modifikací, a vytvořením zcela nových indikátorů, což je činnost, kterou se pokusím následně popsat.

4.2.1 Simple Moving Average

Simple Moving Average (SMA) je klouzavý průměr signálu S, který odpovídá aritmetickému průměru n posledních hodnot [12].

$$SMA_0(S, n) = \frac{S_0 + S_1 + \cdots + S_{n-1}}{n} \tag{4.1}$$

Tento indikátor vyhlazuje krátkodobé výkyvy vstupního signálu. Dále, v případě, že je vstupem zdrojový signál ceny svíce, indikátor lze využít:

- ke stanovení trendu signálu (pomocí porovnání či rozdílu aktuální hodnoty *SMA* a aktuální ceny),
- k identifikaci cenových hladin – *supportu* a *rezistence*[2].

4.2.2 Exponential Moving Average

Exponential Moving Average (EMA) je klouzavý průměr signálu S, který přisuzuje nejvyšší váhu hodnotě signálu aktuální svíce, a váha se s počtem svící do minulosti postupně snižuje. Nejvyšší váha je obvykle vypočtena na základě parametrů n a s (*smoothing*). Ke stanovení počáteční hodnoty se využívá typicky aritmetický průměr n posledních hodnot signálu [9].

$$EMA_0(S, n, s) = \begin{cases} SMA_0(S, n), & \text{pokud } EMA_1 = NaN \\ EMA_1 \cdot (1 - \alpha) + S_0 \cdot \alpha, & \text{jinak} \end{cases} \tag{4.2}$$

$$\alpha = \frac{s}{1 + n} \tag{4.3}$$

Tento indikátor lze využít obdobným způsobem jako *SMA*, přičemž díky váhování dokáže rychleji reagovat na prudké výkyvy signálu.

4.2.3 Moving Average Convergence Divergence

Autorem indikátoru *Moving Average Convergence Divergence (MACD)* je Gerald Appel [1].

Tento indikátor sleduje hybnost signálu S. Je vypočten jako rozdíl mezi krátkodobou křivkou *EMA* a dlouhodobou křivkou *EMA*.

$$MACD_0(S) = EMA_0(S, 12, 2) - EMA_0(S, 26, 2) \tag{4.4}$$

MACD je jeden z možných signálů, který může indikovat změnu trendu. Začne-li například platit $MACD_0(Close) > 0$, můžeme předpokládat, že dojde k růstu ceny. Naopak začne-li platit $MACD_0(Close) < 0$, můžeme předpokládat, že dojde k poklesu ceny.

[2] Tj. cenových hladin, u kterých při průrazu lze očekávat prudší vývoj cenového kurzu daným směrem [19].

4.2.4 Relative Strength Index

Relative Strength Index (RSI) je indikátor publikovaný v knize New Concepts in Technical Trading Systems. Jeho autorem je John Welles Wilder [20, s. 63–65].

Jedná se o oscilátorový indikátor nad signálem S, jehož hodnoty se pohybují v rozmezí $\langle 0, 100\rangle$. K výpočtu *RSI* je zapotřebí stanovit průměrný nárust a průměrný pokles signálu za počet svící n.

Existuje řada variant indikátorů, které se zde využívají ke stanovení průměru. Dovolím si zde zavést pomocný indikátor *RSI Moving Average (RMA)*, který je inspirován implementací v systému TradingView, a odpovídá původní publikaci. V rámci vlastního řešení však často používám modifikaci, která namísto *RMA* využívá *SMA*.

$$RSI_0(S, n) = 100 - \frac{100}{1 + \frac{\text{avggain}}{\text{avgloss}}} \tag{4.5}$$

$$\text{avggain} = RMA_0(MAX_0(S_0 - S_1, 0), n) \tag{4.6}$$

$$\text{avgloss} = RMA_0(ABS_0(MIN_0(S_0 - S_1, 0)), n) \tag{4.7}$$

$$RMA_0(S, n) = \begin{cases} SMA_0(S, n), & \text{pokud } RMA_1 = NaN \\ RMA_1 \cdot (1 - \frac{1}{n}) + S_0 \cdot \frac{1}{n}, & \text{jinak} \end{cases} \tag{4.8}$$

Indikátor *RSI* lze využít k identifikaci překoupení či přeprodání aktiva. Například, je-li $RSI_0(S, n) \leq 30$, pak můžeme předpokládat, že aktivum je přeprodané, a očekávat tak růst ceny. Naopak, je-li $RSI_0(S, n) \geq 70$, pak můžeme předpokládat, že aktivum je překoupené, a očekávat tak pokles ceny. Volba levelů překoupení a přeprodání je pochopitelně součástí obchodní strategie a takovou indikaci je zpravidla potřeba podrobit dalším testům.

4.2.5 Connors Relative Strength Index

Connors Relative Strength Index (CRSI) je indikátor představený v knize An Introduction to ConnorsRSI. Jeho autorem je Larry Connors [10].

Jedná se podobně jako u *RSI* o oscilátorový indikátor nad signálem S v rozmezí $\langle 0, 100\rangle$. Výpočet spočívá v aritmetickém průměru ze 3 oscilátorových složek, které se všechny pohybují ve stejném rozmezí.

UpDown je indikátor, který vyjadřuje počet po sobě rostoucích/klesajících hodnot signálu (např. hodnota 3 odpovídá třem růstům, hodnota -2 dvěma poklesům, atd.).

Indikátor *Rate of Change (ROC)* vyjadřuje procentuální změnu hodnoty signálu mezi aktuální a předchozí svící.

Indikátor *Percentage Rank (Perc)* vyjadřuje, jaký je počet n předchozích hodnot signálu S (bez zahrnutí aktuální hodnoty), které jsou nižší než aktuální hodnota signálu.

$$CRSI_0(S, l_1, l_2, l_3) = \frac{RSI_0(S, l_1) + RSI_0(UpDown(S), l_2) + Perc_0(ROC_0(S), l_3)}{3} \tag{4.9}$$

$$UpDown_0(S) = \begin{cases} \max(UpDown_1, 0) + 1 & \text{pokud } S_1 < S_0 \\ \min(UpDown_1, 0) - 1 & \text{pokud } S_1 > S_0 \\ 0 & \text{jinak} \end{cases} \tag{4.10}$$

$$ROC_0(S) = \frac{S_0 - S_1}{S_1} \cdot 100 \tag{4.11}$$

$$Perc_0(S, n) = \frac{\sum_{i=1}^{n} [S_i \leq S_0]}{n} \cdot 100 \tag{4.12}$$

Indikátor *CRSI* má podobné využití jako *RSI*, ale liší se výrazně vyšší frekvencí kmitání (pro běžné kombinace parametrů). Díky tomu je možné na jeho základě stavět například krátkodobé obchodní algoritmy.

Uvádím jej zde jako příklad složitějšího oscilátoru a také z toho důvodu, že jeho dílčí složky vykazují zajímavé vlastnosti, které jsou vhodné pro demonstrování funkčnosti informačního systému.

4.2.6 Average True Range

Average True Range (ATR) je indikátor publikovaný opět v knize New Concepts in Technical Trading Systems [20, s. 21–23].

Tento indikátor slouží k měření volatility ceny aktiva. Je vypočten jako průměr z pomocného indikátoru *True Range (TR)*, přičemž pro výpočet průměru se standardně využívá *RMA*, nicméně lze využít i modifikaci *SMA*.

$$ATR_0(n) = RMA_0(TR_0(), n) \tag{4.13}$$

$$TR_0() = \max \begin{cases} High_0 - Low_0, \\ ABS_0(High_0 - Close_1) \\ ABS_0(Low_0 - Close_1) \end{cases} \tag{4.14}$$

ATR je důležitá metrika, která umožňuje obchodníkům přizpůsobit rozhodnutí volatilitě trhu. Může pomoci například při stanovování cenové hladiny stop-loss či cílové cenové hladiny pro prodej (pochopitelně v těchto případech je typické například zavedení koeficientů pro násobení *ATR*). Zároveň tento indikátor lze využít k utlumení obchodů v případě nízké volatility tak, aby obchodní algoritmus neztrácel zbytečně prostředky na burzovních poplatcích.

4.3 Kombinované technické indikátory

Pokud se díváme na výstupy technických indikátorů jako na signály, dává smysl nad těmito signály tvořit další logiku – přivádět je na vstup dalších technických indikátorů. Pro tento účel si dovolím zavést jednoduchý koncept *kombinovaných technických indikátorů*.

Kombinovaný technický indikátor definuji jako jednoduchou matematickou funkci, která realizuje nějakou elementární operaci nad technickými indikátory, přičemž nepracuje s jejich historií, ale pouze poslední vypočtenou hodnotou.

Často je zapotřebí, aby vstupem pro kombinovaný technický indikátor byla také „konstantní hodnota“ – v takovém případě budeme hovořit o *konstantním indikátoru*, jehož výstupem je konstantní signál dané hodnoty.

4.3.1 Aritmetické indikátory

Aritmetické indikátory jsou velmi jednoduché indikátory, které realizují aritmetickou operaci nad aktuálními hodnotami dvou vstupních signálů.

$$Add_0(A, B) = A_0 + B_0 \tag{4.15}$$
$$Sub_0(A, B) = A_0 - B_0 \tag{4.16}$$
$$Mul_0(A, B) = A_0 \cdot B_0 \tag{4.17}$$
$$Div_0(A, B) = \frac{A_0}{B_0} \tag{4.18}$$
$$\tag{4.19}$$

Využití *aritmetických indikátorů* často spočívá v potřebě úpravy základních technických indikátorů. Můžeme například od *SMA* křivky odečíst *ATR* násobené konstantní hodnotou, abychom získali *stop-loss* křivku, u které lze předpokládat propad ceny aktiva, platí-li $stoploss_0 \leq Close_0$.

4.3.2 Porovnávací indikátory

Porovnávací indikátory jsou velmi jednoduché indikátory, které realizují porovnání aktuálních hodnot dvou vstupních signálů. Je-li výsledek porovnání pravdivý, bude výstupem hodnota 1. Bude-li výsledek porovnání nepravdivý, bude výstupem hodnota 0. Výstup je číselný a jedná se opět o hodnoty, které budou v programu reprezentovány datovým typem *double* tak, aby nebyla porušena kompatibilita s ostatními indikátory[3].

$$GT_0(A, B) = \begin{cases} 1 & \text{pokud } A_0 > B_0 \\ 0 & \text{jinak} \end{cases} \tag{4.20}$$
$$GTE_0(A, B) = \begin{cases} 1 & \text{pokud } A_0 \geq B_0 \\ 0 & \text{jinak} \end{cases} \tag{4.21}$$
$$LT_0(A, B) = \begin{cases} 1 & \text{pokud } A_0 < B_0 \\ 0 & \text{jinak} \end{cases} \tag{4.22}$$
$$LTE_0(A, B) = \begin{cases} 1 & \text{pokud } A_0 \leq B_0 \\ 0 & \text{jinak} \end{cases} \tag{4.23}$$

Využití *porovnávacích indikátorů* je zřejmé – umožňují modelu se rozhodovat. Například je možné vytvářet jednoduché diskrétní signály pro nákup nebo prodej (je-li hodnota indikátoru vyšší než 0, může algoritmus provést danou akci). Dále umožňují tvořit mnohem komplexnější logiku i v kombinaci s rozhodovacími indikátory.

[3]Pozn. zatím jsem nenašel využití pro indikátor porovnání na rovnost signálů – z toho důvodu jej neuvádím. Implementace by byla problematická minimálně vzhledem k vlastnostem čísel s plovoucí řádovou čárkou.

4.3.3 Rozhodovací indikátory

Rozhodovací indikátory zpracovávají vstupní signály a na jejich základě provádí rozhodnutí o tom, který ze vstupních signálů bude přiveden na výstup v nezměněné podobě.

Indikátory *And* a *Or* jsou inspirovány stejnojmennými operátory ve skriptovacích jazycích tak, že jejich výstupem je první hodnota, pomocí které lze po převedení na logickou hodnotu určit platnost logického výrazu.

Indikátor *Ternary* umožňuje na základě aktuální logické hodnoty signálu C přepínat na výstup signály T a F.

$$And_0(D, F) = \begin{cases} D_0 & \text{pokud } D_0 = 0 \\ F_0 & \text{jinak} \end{cases} \tag{4.24}$$

$$Or_0(D, F) = \begin{cases} D_0 & \text{pokud } D_0 \neq 0 \\ F_0 & \text{jinak} \end{cases} \tag{4.25}$$

$$Ternary_0(C, T, F) = \begin{cases} T_0 & \text{pokud } C_0 \neq 0 \\ F_0 & \text{jinak} \end{cases} \tag{4.26}$$

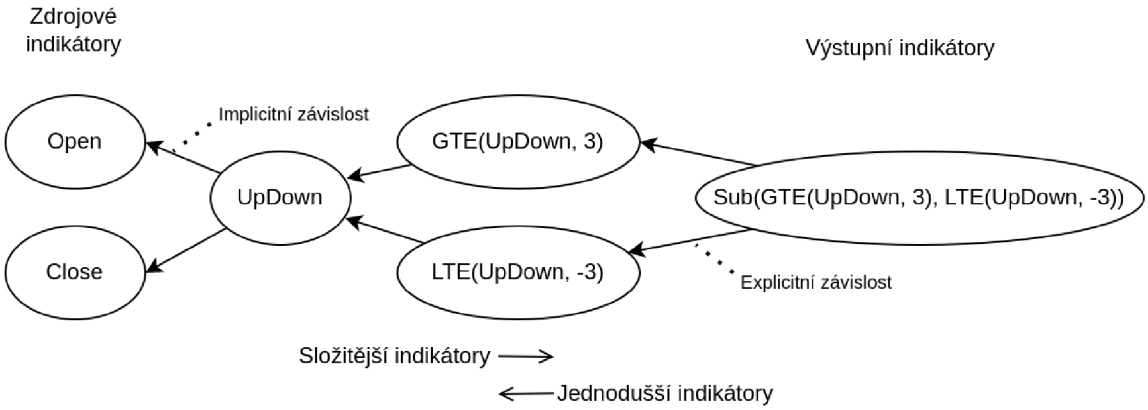
Rozhodovací indikátory umožňují realizovat mnohem komplexnější chování modelu. Můžeme například chtít, aby algoritmus vstupoval do dlouhé pozice pouze v rostoucím trhu s určitou volatilitou, kombinovat podmínky pro nákup a prodej, apod. Model, který budujeme, pochopitelně umožňuje i vyhlazovat výstup *rozhodovacího indikátoru*, např. pomocí $SMA(Ternary(\dots), n)$, čímž se můžeme vyhnout skokovým změnám signálu.

4.4 Síť technických indikátorů

Díky kombinovaným technickým indikátorům můžeme velkou část logiky *obchodního algoritmu* přesunout do indikátorů, které přivedeme na jeho vstup. Obchodní algoritmus bude moci poté vykonávat obchodní rozhodnutí na základě signálů. Výhodou takovéto implementace je bezesporu jednoduchost a transparentnost definice algoritmu. Samotné know-how rozhodování poté spočívá nejen v definici použitých indikátorů, ale především v jejich vzájemném propojení a v nastavení parametrů. Hovořme poté o *síti technických indikátorů*.

V kapitolách o základních a kombinovaných indikátorech jsem uváděl řadu matematických funkcí pro jejich definici. Některé indikátory přijímaly jako vstupní parametr jiný indikátor, jehož signál využívaly k vlastnímu výpočtu. Jiné (a zvláště základní indikátory) vyžadovaly pro vlastní výpočet signál konkrétního indikátoru, který nebylo možné zvenku ovlivnit. Indikátor vyžadující jiný indikátor k vlastnímu výpočtu nazvěme *složitějším indikátorem* a naopak vyžadovaný indikátor nazvěme *jednodušším indikátorem*. O vztahu mezi těmito indikátory říkejme, že složitější indikátor *je závislý* na jednodušším indikátoru. Pokud je jednodušší indikátor vložen jako parametr složitějšímu indikátoru, hovořme o *explicitní závislosti*. V opačném případě hovořmě o *implicitní závislosti*, a to i v případě, že jednodušší indikátor přijímá parametr složitějšího indikátoru.

Indikátory, které nemají žádnou závislost a nejsou *konstantními* indikátory, nazvěme *zdrojové indikátory*. Zdrojové indikátory extrahují nějakou konkrétní vlastnost svíce a přinášejí ji do sítě indikátorů (např. *Open* vybírá počáteční cenu svíce, *Close* koncovou cenu svíce, atd.).



Obrázek 4.1: Příklad jednoduché sítě technických indikátorů pro Demo algoritmu (kap. 6)

Zřejmou vlastností sítě je, že produkuje pro stejné zdrojové signály také stejné výstupní signály. Modelování sítě technických indikátorů je poté kreativní činnost, kterou je možné v určitých případech zjednodušit pomocí využití některých metod umělé inteligence.

Uvažujme *obchodní algoritmus*, který pro vykonání obchodního rozhodnutí vyžaduje aktuální hodnoty signálů indikátorů. Hodnota takového signálu poté může reprezentovat nějakou znalost založenou na historickém vývoji trhu, a tedy se může jednat o předpoklad pro následující vývoj, na základě kterého může obchodní algoritmus rozhodnutí provést. Zde se dostávám k důležité výhodě, kterou *síť technických indikátorů* poskytuje: všechny signály, se kterými síť pracuje, je možné vizualizovat, analyzovat a zdůvodnit i zpětně rozhodnutí provedená algoritmem. Narozdíl od využití neuronové sítě by tedy měla být všechna rozhodnutí transparentní (i díky menšímu množství výpočetních uzlů) a do jisté míry „předschválená“ člověkem.

Obrázek 4.2: Příklad signálů generovaných sítí technických indikátorů pro Demo algoritmus

Kapitola 5

Model technických indikátorů

V kapitole 4 jsem uvedl a definoval řadu technických indikátorů a zavedl jsem koncept *sítě technických indikátorů*. Jedná se o abstraktní model pro technickou analýzu trhu. Abychom mohli *síť technických indikátorů* realizovat, potřebujeme nyní vytvořit pokud možno izomorfní programový model.

Při experimentování s technickými indikátory jsem nejdříve pracoval s nejjednodušší možnou variantou – využil jsem návrhový vzor dekorátoru, v rámci kterého složitější indikátory zapouzdřovaly jednodušší, přičemž explicitní závislosti byly definovány pomocí parametrů konstruktoru. Pro výpočet hodnoty libovolného indikátoru byla volána metoda `Indicator.next(Candle)`, která přijímala svíci a vracela vypočtenou hodnotu (popř. hodnotu *NaN*, pokud nebylo možné výpočet provést). Pokud složitější indikátor vyžadoval výpočet jednoduššího indikátoru, volal nad ním opět stejnou metodu (v rámci vlastní implementace metody `Indicator.next(Candle)`). Pokud indikátor vyžadoval pro vlastní výpočet předchozí hodnoty některého signálu, musel implementovat vlastní buffer, do kterého historii (odpovídající potřebnému počtu svíček) daného signálu ukládal. Hlavní nevýhodou tohoto přístupu, na kterou jsem narazil, byla samotná složitost kódu – matematické výpočty, které bylo zapotřebí definovat, zahrnovaly mnoho příkazů *imperativního programování*, což v konečném důsledku znemožňovalo implementaci složitějších indikátorů. Dalším problémem bylo, že většina instancí indikátorů držela vlastní stav – bylo zapotřebí tak řešit problémy jako klonování instancí indikátorů explicitních závislostí, validace časového razítka vstupní svíčky na úrovni každé instance indikátoru, atd. Pochopitelně nebylo možné výpočet optimalizovat tím způsobem, aby každá hodnota unikátního indikátoru v síti byla vyhodnocena maximálně jednou.

Tyto problémy mě přesvědčily, že má smysl se zabývat návrhem programového modelu nejen ze sémantické podstaty, ale také po syntaktické stránce, neboť maximální přehlednost jak definice samotných indikátorů, tak celé sítě je klíčová.

5.1 Syntaxe

Syntaxe, kterou můžeme využít pro tvorbu programového modelu, je do značné míry omezena programovacím jazykem, který pro implementaci modelu vybereme. Z tohoto důvodu jsem se rozhodl programový model vytvořit hned ve dvou jazycích.

Programový model, který využívám pro simulování, jsem realizoval v jazyce Python3 především kvůli možnosti přetížit operátory. Tento model umožňuje velice jednoduchou de-

finici sítě indikátorů i výpočetného vzorce konkrétních indikátorů, což výrazným způsobem zefektivňuje interakci člověka s tímto modelem.

```
buyLen = 3
sellLen = 3

close = Close() # Input indicator
updown = UpDown(close)
buySignal = updown >= buyLen
sellSignal = updown <= -sellLen
buySellSignal = buySignal - sellSignal # Output indicator
```

Algoritmus 1: Příklad definice sítě indikátorů v Python3

```
def compute(self, sources: "tuple[IndicatorHistory, ...]"):
    source, updown = sources

    prev, current = source[1], source[0]

    if prev < current: # Rising
        return max(updown[1] or 0, 0) + 1
    elif prev > current: # Dropping
        return min(updown[1] or 0, 0) - 1
    else: # Equal
        return 0
```

Algoritmus 2: Příklad definice indikátoru UpDown v Python3

Informační systém však realizuji v jazyce TypeScript z objektivních důvodů, které budou uvedeny později v kapitole 8. V tomto jazyce jsem implementoval totožný model také. Tento model, narozdíl od modelu v Python3, vyžaduje explicitní využití konstruktorů a metod. Není proto tolik vhodný pro časté využívání člověkem, ale naopak slouží jako konfigurační prostředek, který svým charakterem omezuje výskyt nejednoznačností. Navíc, díky volbě jazyka, poskytuje statickou typovou kontrolu.

```
new Sub(
    new GTE(new UpDown(3), 3),
    new LTE(new UpDown(3), -3)
)
```

Algoritmus 3: Příklad definice sítě indikátorů v konfiguraci TypeScript

```
compute(sources: IndicatorHistory[]): IndicatorValue {
    const [open, close, upDown] = sources;

    const openVal = open.at();
    const closeVal = close.at();

    if (openVal < closeVal) {
      return Math.max(upDown.at(1) || 0, 0) + 1; // Rising
    } else if (openVal > closeVal) {
      return Math.min(upDown.at(1) || 0, 0) - 1; // Dropping
    } else {
      return 0; // Equal
    }
  }
```

Algoritmus 4: Příklad definice indikátoru UpDown v TypeScript

5.2 Sémantika

V rámci této kapitoly vysvětlím sémantickou část navrženého *výpočetního modelu indikátorů* implementovaného v jazyce TypeScript, který bude součástí informačního systému.

Každý indikátor je reprezentován třídou, která při tvorbě instance prostřednictvím konstruktoru obdrží *explicitní závislosti* (instance indikátorů) a vlastní konstantní parametry. Indikátor implementuje 2 nejdůležitější metody:

- `Indicator.sources(): Indicator[]`, která specifikuje všechny implicitní a explicitní závislosti instance indikátoru, a
- `Indicator.compute(dependencies: IndicatorHistory[]): number`, která provádí samotný výpočet na základě signálů, na kterých závisí.

Pro uchování signálů byla vytvořena třída `IndicatorHistory`. Tato třída zapouzdřuje klouzavé okno, které uchovává předchozí hodnoty signálu. K hodnotám signálu se přistupuje pomocí indexu v opačném pořadí, než jsou data uložena (index 0 odpovídá hodnotě signálu nad aktuální svící, index 1 hodnotě nad předchozí svící, atd.). Tato třída implementuje dvě klíčové metody:

- `IndicatorHistory.push(value: number)`, která přidá novou hodnotu signálu odpovídající nové aktuální svíci a upraví tak indexování,
- `IndicatorHistory.replace(value: number)`, která změní hodnotu signálu aktuální svíce.

Vždy, než je zavolána metoda `Indicator.compute(dependencies: IndicatorHistory[]): number` pro výpočet nové hodnoty indikátoru, musí být vypočteny všechny hodnoty jednodušších indikátorů nad aktuální svící, na kterých je daný indikátor závislý. V případě, že je indikátor závislý na vlastní předchozí hodnotě (jedná se o rekurentní výpočet, např. *EMA*), poté se zde počítá s doplněním nedefinované hodnoty signálu nad aktuální svící před provedením výpočtu (`IndicatorHistory.push(NaN)`). Po provedení výpočtu může být nově vypočtená hodnota dosazena na dané místo (`IndicatorHistory.replace(computedValue)`).

Platí tedy, že v době výpočtu nové hodnoty indikátoru nad svící C_t musí být všechny signály závislostí „zarovnané“ tak, že jejich `IndicatorHistory[0]` odpovídá hodnotě nad svící C_t.

Proces výpočtu zajišťuje třetí článek modelu indikátorů – `IndicatorEvaluator`. Jedná se o třídu, jejíž instance reprezentuje konkrétní síť indikátorů a její stav proměnlivý v čase. Pro vytvoření instance je zapotřebí definovat síť indikátorů. Síť indikátorů definuji nejjednodušším možným způsobem – specifikuji N-tici instancí *výstupních indikátorů* sítě, které je zapotřebí vypočítat. Se znalostí definic všech závislostí je tak možné síť celou zkonstruovat. `IndicatorEvaluator` nad sítí provede analýzu s pomocí grafových průchodů a připraví indikátory a jejich historie do takového pořadí, ve kterém je možné dále optimálním způsobem provádět opakovaná vyhonocení. Řazení indikátorů je provedeno vzestupně podle celkového počtu jednodušších indikátorů, na kterých je daný indikátor závislý, přičemž rekurentní závislost do tohoto počtu není zahrnuta[1].

Samotný výpočet je realizován prostřednictvím 2 veřejných `IndicatorEvaluator`:

- `IndicatorEvaluator.initialize(candles: Candle[])`, která provede počáteční výpočet sítě s předem známým minimálním počtem svící nutným pro konvergenci *výstupních indikátorů*, a
- `IndicatorEvaluator.push(candle: Candle)`, která přidá do modelu aktuální svíci a dopočitá aktuální hodnotu *výstupních indikátorů*.

5.2.1 Zdrojové indikátory

Při zavádění *výpočetního modelu indikátorů* jsem prakticky zamezil, aby mohla definice výpočtu indikátoru `Indicator.compute(dependencies: IndicatorHistory[])` pracovat přímo se svící na vstupu, neboť jsem definoval, že indikátor je vypočten na základě vstupních signálů, nikoli vstupních instancí struktur – svící.

Tento problém řeším jednoduchým způsobem: zavedením *zdrojových indikátorů*. Zdrojový indikátor v rámci metody `Indicator.compute(dependencies: IndicatorHistory[])` vyvolává instanci výjimky `ExtractCandleInstead`, která zapříčiní přepnutí výpočtu v rámci `IndicatorEvaluator` tak, aby pro vyhodnocení indikátoru byla využita metoda `Indicator.extract(candle: Candle)`[2]. V rámci této metody poté dochází k extrahování hodnoty vstupní svíce, díky čemuž vzniká *zdrojový signál*, který je dále již standardním způsobem přiváděn na vstupy *závislých* indikátorů.

Každý zdrojový indikátor extrahuje buď jednu hodnotu vstupní svíce (*Open*, *High*, *Low*, *Close*), a nebo může přistoupit k více hodnotám svíce a provést s nimi elementární výpočet, například[3]:

$$HL2_0 = \frac{High_0 + Low_0}{2} \tag{5.1}$$

$$HLC3_0 = \frac{High_0 + Low_0 + Close_0}{3} \tag{5.2}$$

[1] Pozn. funkčnost tohoto modelu jsem ověřoval experimentálně. Jistě by bylo možné vytvořit pro to formální důkaz, neboť se jedná o systém, který neumožňuje cyklické závislosti mezi různými indikátory na sobě.

[2] Jde o přehledný způsob, který díky početně malému zastoupení vstupních indikátorů pracuje rychleji, než kdyby při vyhodnocování každého indikátoru nad každou svící mělo docházet ke kontrole třídy instance indikátoru.

[3] Pozn. signály *HL2* a *HL3* jsou inspirovány jazykem Pine Script na platformě TradingView.

5.2.2 Konvergence technických indikátorů

Při běžném provozu informačního systému je instance `IndicatorEvaluator` uložena v paměti, a při kompletaci aktuální svíce (daného časového rozsahu) je volána metoda `IndicatorEvaluator.push(candle: Candle)` s argumentem poslední kompletní svíce. Nový výpočet tak naváže na dříve vypočtené hodnoty signálů.

Pokud je ale informační systém spuštěn, je zapotřebí provést inicializaci instance `IndicatorEvaluator`. Vzhledem k potřebě výpočtu indikátorů s konečným a předem známým počtem svící jsem model rozšířil o *minimální počet svící nutný pro konvergenci*, který musí specifikovat každá instance indikátoru. Nechť pro všechny indikátory S platí:

$$S.minInitLength = S.noncLength + 1, \tag{5.3}$$

kde *minInitLength* je minimální počet svící nutných pro výpočet první zkonvergované hodnoty signálu a *noncLength* je maximální počet nezkonvergovaných hodnot, které může při opakovaném výpočtu indikátor vyprodukovat. Za nezkonvergovanou hodnotu budu považovat také neinicializovanou hodnotu NaN.

Zřejmou vlastností všech *zdrojových indikátorů* je, že při inicializaci konvergují již nad první svící, tedy:

$$\forall I \ \in Sources : I.noncLength = 0 \tag{5.4}$$

Pro většinu *základních technických indikátorů* není definice délky konvergence složitá, ale je zapotřebí brát v potaz délku konvergence vstupních signálů – příkladem může být $SMA(S, n)$, kde:

$$SMA(S, n).noncLength = S.noncLength + n - 1 \tag{5.5}$$

Některé indikátory z analytického hlediska nemusí konvergovat nikdy. Z toho důvodu zde počítám s minimální vyžadovanou procentuální přesností, od které již hodnoty považuji za zkonvergované. Příkladem může být $EMA(S, n, s)$:

$$EMA(S, n, s).noncLength = SMA(S, n).noncLength + x \tag{5.6}$$

$$\alpha = \frac{s}{n+1} \tag{5.7}$$

$$x = \lceil \frac{\epsilon}{100 \cdot \log(1-\alpha)} \rceil, \tag{5.8}$$

kde ϵ [%] je maximální odchylka první vypočtené hodnoty $EMA(S, n, s)$, kterou budu považovat za zkonvergovanou, od skutečně zkonvergované hodnoty[4].

Poslední třídou indikátorů, na které je zapotřebí dát pozor, jsou indikátory příliš svázané s dříve vypočtenými hodnotami vlastního signálu, v důsledku čehož není možné analyticky určit minimální počet svící nutný pro konvergenci s předem danou přesností. Nazvěme je *stavovými indikátory*. Příkladem může být indikátor $UpDown(S)$:

$$UpDown_0(S) = \begin{cases} \max(UpDown_1, 0) + 1 & \text{pokud } S_1 < S_0 \\ \min(UpDown_1, 0) - 1 & \text{pokud } S_1 > S_0 \\ 0 & \text{jinak} \end{cases} \tag{5.9}$$

[4]Funkčnost vzorce pro výpočet $EMA(S, n, s).noncLength$ jsem ověřoval experimentálně s $\epsilon = 0.001$ % na testovacích datech – denních svících $(BTC, USDT)$ za rok 2020 na burze Binance, přičemž nově inicializované hodnoty indikátorů byly téměř totožné s hodnotami vypočtenými pomocí rekurentního výpočtu (kde inicializace byla provedena o rok dříve) a na funkčnost algoritmů měly stejný vliv.

Z jeho definice skutečně nelze určit minimální počet svící nutný pro vyhodnocení. Budeme-li například uvažovat posloupnost 100 po sobě rostoucích svící, nemůžeme provádět inicializaci pouze na 10 svících – hodnota signálu by tak byla inicializována s chybou 90%.

Tento konkrétní případ by bylo pochopitelně možné jednoduchým způsobem vyřešit pomocí *dvojité inicializace*, přičemž první inicializace by zahrnovala n svíci a druhá inicializace $n+1$ svící. Pokud by výsledkem obou inicializací byla stejná inicializovaná hodnota signálu, inicializace by byla úspěšná. V opačném případě by byla inicializace prováděna znovu, vždy s delším intervalem svící o 1, a hodnota signálu by byla porovnávána s hodnotou z předchozí inicializace. Takové řešení by však mělo řadu nevýhod:

- inicializace by mohla být teoreticky velmi výpočetně náročná,
- v krizovém případě by nám nemusel stačit předem omezený interval svící pro inicializaci (omezený historií svící poskytovaných burzou, ale také vlastními výpočetními prostředky) a
- především by se nejednalo o zcela univerzální řešení, které by bylo možné využít v případě všech *statových indikátorů*.

Toto je jedno z míst mojí práce, kde jsem se rozhodl problém řešit pomocí heuristik. Budeme-li uvažovat hodinové svíce historického vývoje páru $(BTC, USDT)$ na burze Binance mezi `1.1.2018` až `1.1.2023`, poté nejvyšší hodnota signálu $UpDown(Close)$ byla 12 (`4.11.2020 19:00`) a nejnižší hodnota signálu byla -11 (`28.2.2020 12:00`). Pokud inicializujeme signál $UpDown(Close)$, víme na základě historického vývoje trhu, že existuje značná pravděpodobnost, že signál inicializujeme správně na intervalu 25 svící. Nechť je tedy minimální počet svící pro inicializaci parametrem indikátoru:

$$UpDown(S, minInitLength).noncLength = minInitLength - 1 \tag{5.10}$$

Obdobný postup je možné využít i v případě ostatních *stavových indikátorů*, nicméně je zapotřebí vždy zvážit pravděpodobnost chybné inicializace a počítat s jejími důsledky na chování algoritmu.

Druhý způsob, který využívám pro řešení popsané situace, je samotný návrh *sítě indikátorů* tak, aby chybně inicializovaná hodnota signálu neměla na výstupní hodnoty sítě vliv s ohledem na použité indikátory. Příkladem může být vložení *UpDown* jako explicitní závislost indikátoru *RSI*, kde nezáleží na konkrétní hodnotě vstupního signálu[5], ale pouze na jeho změnách, a proto:

$$RSI(\,UpDown(S, n), n) = RSI(\,UpDown(S, \infty), n), \tag{5.11}$$

dále pokud budeme signál *UpDown* například porovnávat, zajímá nás pouze hodnota signálu inicializovaná do výše porovnání[6]:

$$UpDown(Close, 3) \geq 3 \iff UpDown(Close, \infty) \geq 3 \tag{5.12}$$

$$UpDown(Close, 3) \leq -3 \iff UpDown(Close, \infty) \leq -3. \tag{5.13}$$

[5]Pozn. indikátor *RSI* pochopitelně bude definovat také vlastní počet svící nutných pro inicializaci, protože sám vyžaduje výpočet klouzavého průměru ze změn vstupního signálu pomocí indikátoru *RMA*. Reálně tak bude k inicializaci využito $RSI(UpDown(S, n), n).minInitLength$, což je více než $UpDown(S, n).minInitLength$.

[6]Tato metoda se používá v síti indikátorů Demo algoritmu.

5.2.3 Přenositelnost a zabezpečení

Poslední aspekty *výpočetního modelu indikátoru*, o kterých bych se rád zmínil, jsou přenositelnost a zabezpečení.

Model je navržen tak, aby průběžně uvolňoval staré hodnoty signálů, které nebudou již potřeba pro vyhodnocení nových hodnot výstupních indikátorů. Díky tomu je možné model provozovat v rámci dlouho běžící serverové aplikace, přičemž nedochází k vyčerpání paměti.

V rámci informačního systému budeme ale chtít, aby si administrátor mohl zobrazit signály sítě indikátorů v čase, a mohl tak ověřit správnost rozhodování algoritmů. Při počátečních návrzích informačního systému jsem brzy dospěl k závěru, že není udržitelné hodnoty signálů ani ukládat do databáze, ani posílat po síti, a to zejména u algoritmů pracujících nad grafy s krátkodobými svícemi. Vzhledem k tomu, že hlavní prioritou systému je velmi rychlé rozhodování, nechceme ani provádět jiné než nezbytné výpočty při obsluze požadavků uživatele.

Řešením je přenos *sítě indikátorů* a vstupních svící k uživateli tak, aby klientský program mohl sám indikátory zpětně dopočítat a dále graficky zobrazit. Z tohoto důvodu je *výpočetní model indikátorů* implementován jako balíček v jazyce TypeScript a je součástí instalace jak serverové aplikace, tak uživatelské aplikace. Součástí instalace je definice jednotlivých indikátorů, ale nikoli *sítí indikátorů*, které jsou využívány jednotlivými algoritmy.

Model indikátorů umožňuje síť *serializovat* tak, aby mohla být přenesena po síti, a dále ji umožňuje *deserializovat*, aby mohly být totožné výpočty prováděny na straně klienta. Jediný rozdíl nastavení modelu spočívá ve vypnutí možnosti použití klouzavého okna – uživatel si tak může zobrazit i historické hodnoty signálů sítě indikátorů, které na straně serverové aplikace již byly uvolněny.

Přenos probíhá v rámci HTTPS/GraphQL spojení, ve formátu JSON s předem nespecifikovanou strukturou, a tato služba je dostupná pouze administrátorům informačního systému.

Kapitola 6

Model obchodního algoritmu

V rámci kapitoly 2 jsem uvedl pohled na koncept *obecného obchodního algoritmu*. Pokud budeme chtít obchodní algoritmus realizovat, budeme vždy pracovat s *homomorfním modelem* obecného obchodního algoritmu, tedy ze dříve popsaného konceptu vybereme pouze některé vlastnosti, které budou v novém modelu zohledněny. Nazvěme takový model *(výpočetním) modelem obchodního algoritmu*.

6.1 Simulační model obchodního algoritmu

Před samotnou realizací výpočetního modelu bychom se nejdříve měli zabývat realizací *simulačního modelu obchodního algoritmu*. Snažíme se vytvořit model, pomocí kterého budeme schopni ověřit chování výpočetního modelu.

Takový model by v ideálním případě měl být *izomorfní* s výpočetním modelem, ale této vlastnosti 1.) není možné nikdy plně docílit a 2.) nechceme jí vždy docílit. Pokud algoritmus navrhujeme, zabýváme se nejdříve simulací rozhodování, abychom zjistili míru správných a špatných rozhodnutí, které by algoritmus v ideálním systému provedl. Potřebujeme tedy, aby simulace byla plně *deterministická*.

Teprve po nalezení vhodného rozhodovacího modelu může mít smysl se zabývat simulací vlastností, které se vyskytují při provozu výpočetího modelu:

- závislost na API burzy – výskyt náhodných chyb;
- závislost na kvalitě síťového připojení – zpoždění prováděných operací;
- práce v reálném čase – doba nutná pro realizaci rozhodnutí algoritmu;
- poptávka/nabídka na daných cenových hladinách – odlišná cena realizace objednávek;
- dostupnost odlišných informací oproti simulačnímu modelu, atd.

Jistě bude platit, že se zvyšující se frekvencí prováděných operací algoritmem se stává simulace těchto vlastností důležitější. Zároveň některé z uvedených vlastností lze simulovat teprve po provedení vlastních měření či na základě dat vyprodukovaných při provozu takového výpočetního modelu algoritmu, což je v částečném rozporu s tím, že bychom chtěli pomocí simulačního modelu nejdříve ověřit chování výpočetního modelu, a teprve poté výpočetní model spouštět.

6.2 Bezstavový model algoritmu

Při experimentování s obchodními algoritmy, které analyzují trh pomocí indikátorů technické analýzy, jsem došel k některým modelům, které by bylo možné pro realizaci využít.

Model, který budu prezentovat v rámci této práce je vhodný pro realizaci jednoho předpisu obchodního algoritmu nad více jeho instancemi. Výhodou tohoto modelu je bezesporu rychlost, s jakou lze vytvořit nový předpis obchodního algoritmu, jednoduchost samotného předpisu obchodního algoritmu, a možnost parametrizovat chování obchodního algoritmu pomocí *sítě technických indikátorů*. Tento model je zároveň velice praktický pro testování. Nevýhodou tohoto modelu je poté řada omezení, která model zavádí.

Nechť je *bezstavový model obchodního algoritmu* výpočetním modelem duálního obchodního algoritmu s úplnou správou prostředků. Bezstavový model je tvořen následujícími prvky:

1. bezstavovým rozhodovacím modelem algoritmu,
2. výpočetním modelem technických indikátorů,
3. synchronizačním algoritmem,
4. konfigurací algoritmu,
5. běhovým prostředím a
6. rozhraním pro komunikaci s burzou.

Jedná se o obecný koncept, s pomocí krerého ale potřebujeme realizovat konkrétní algoritmy. K definování konkrétního algoritmu slouží *konfigurace algoritmu*, která definuje:

- *pár* prostředků, nad kterým bude algoritmus vykonáván,
- *frekvenci* provádění algoritmu, která odpovídá také délce svící, nad kterými bude pracovat *model technických indikátorů* a
- *rozhodovací model* včetně vstupních indikátorů (výstupní signály *sítě indikátorů*).

Uživatelé systému si poté budou moci založit *instance algoritmů*, přičemž každá instance odpovídá právě jedné konfiguraci.

6.2.1 Bezstavový rozhodovací model

Bezstavový rozhodovací model definuje chování obchodního algoritmu. Parametry pro vytvoření instance rozhodovacího modelu jsou výstupní indikátory definující síť *síť technických indikátorů*. Rozhodovací model definuje metodu `decide(indicators: number[], currentCandle: Candle)`, která na základě aktuálních hodnot technických indikátorů (vypočtených nad poslední kompletní svící) a aktuální (nekompletní) svíce vydává posloupnost obchodních rozhodnutí. Aktuální (nekompletní) svíce slouží především pro zjištění aktuálního kurzu v době rozhodnutí. Posloupnost obchodních rozhodnutí odpovídá seřazenému poli o 0-2 prvcích[1]. Každé obchodní rozhodnutí odpovídá některému konkrétnímu typu objednávky, kterou musí podporovat burza, přičemž definuje *stranu objednávky* a případně

[1]Pokud je výsledkem více rozhodnutí, očekává se, že první rozhodnutí je okamžitého charakteru (*Market* objednávka) a druhé je dlouhodobého charakteru (např. *Limit* objednávka).

všechny další parametry, které jsou potřebné pro vytvoření daného typu objednávky. Žádné obchodní rozhodnutí naopak nestanovuje objem prostředků, který je zahrnut při realizaci objednávky, neboť model algoritmu je založen na *úplné správě prostředků* a díky této vlastnosti jsou vydaná rozhodnutí aplikována na všechny prostředky přidělené instanci algoritmu.

Příkladem rozhodovacího modelu může být definice *Market Buy Market Sell* algoritmu. Jedná se o velice jednoduchý koncept, který se rozhoduje na základě hodnoty 1 vstupního signálu:

- pokud je hodnota signálu kladná, model vydává rozhodnutí k okamžitému nákupu,
- naopak pokud je hodnota signálu záporná, model vydává rozhodnutí k okamžitému prodeji.

```
function decide(indicators: number[], currentCandle: Candle):
    StatelessDecision[] {
  const [buySellSignal] = indicators;

  const decisions: StatelessDecision[] = [];
  if (Number.isFinite(buySellSignal)) {
    if (0 < buySellSignal) {
      decisions.push(new MarketBuyDecision());
    } else if (buySellSignal < 0) {
      decisions.push(new MarketSellDecision());
    }
  }

  return decisions;
}
```

Algoritmus 5: Jednoduchý rozhodovací model *Market Buy Market Sell*

Složitější modely poté mohou zohledňovat více signálů a vydávat rozhodnutí k přípravě objednávek, které budou na burze realizovány pouze při splnění podmínek daných parametry objednávky, a tyto parametry mohou být rovněž stanoveny na základě signálů.

6.2.2 Synchronizační algoritmus

Synchronizační algoritmus zajišťuje promítnutí aktuálních rozhodnutí, vydaných rozhodovacím modelem, na konkrétní instancí algoritmu. Jeho vykonání spočívá 1.) ve zrušení všech otevřených objednávek na burze a 2.) v postupném vykonání obchodních rozhodnutí, pro které má instance algoritmu přidělený dostatek prostředků.

Pokud synchronizace instance algoritmu proběhne úspěšně, je to zaevidováno pomocí časového razítka. Při synchronizaci může dojít ale také k chybě, například při pokusu o otevření objednávky na burze. Synchronizační algoritmus v takovém případě chybu neodchytává, ale je na *běhovém prostředí*, aby po nějaké době opět spustilo synchronizaci. Synchronizační algoritmus i v případě chyb ponechává lokální systém v konzistentním stavu, a tedy i v krizových situacích by mělo docházet k postupné realizaci rozhodnutí.

```
async function performDecisions(algorithm, decisions) {
    let [assignedVolatile, assignedStable] = algorithm.assignedAssets();
    const currentCandle = await fetchCurrentCandle(algorithm);

    for (const [idx, decision] of enumerate(decisions)) {
      const enoughAssets = algorithm.hasEnoughAssets(decision);

      if (enoughAssets) {
        const order = decision.createOrder();
        await openOrder(order);

        if (idx < decisions.length - 1) {
          [assignedVolatile, assignedStable] = algorithm.assignedAssets();
        }
      }
    }

    await markAlgorithmEvaluationSuccess(algorithm);
  }
```

Algoritmus 6: Synchronizační algoritmus, zjednodušeno

6.2.3 Běhové prostředí

Úkolem *běhového prostředí* je v následujícím pořadí zajistit:

1. synchronizaci svící sledovaných párů a délky do lokálního systému,
2. zajistit vydání rozhodnutí nad každou konfigurací algoritmu v rámci každé periody (s ohledem na frekvenci konfigurace algoritmu) a
3. zajistit synchronizaci všech instancí algoritmů, pro které bylo v rámci dané periody vydáno obchodní rozhodnutí, ale nebyly v rámci periody zatím úspěšně synchronizovány.

Běhové prostředí budu realizovat v rámci serverové aplikace Nestjs, kde s pomocí knihovny @nestjs/schedule bude spouštěna popsaná úloha každých 10 sekund. Výhodou plánování úlohy v rámci dlouhodobě běžícího procesu, který je založen na událostmi řízené architektuře, je, že všechny komponenty informačního systému jsou po celou dobu běhu aplikace inicializované, a systém tak může naplánovanou úlohu velmi rychle vykonat.

6.3 Demonstrační algoritmus

Pro demonstraci funkčnosti systému jsem na základě *bezstavového modelu* vytvořil jednoduchý algoritmus nad hodinovými svícemi páru $(BTC, USDT)$[2]. Tento algoritmus nakupuje volatilní aktivum v případě 3 po sobě jdoucích rostoucích svící a prodává volatilní aktivum v případě 3 po sobě klesajících svící.

[2]Pozn. nejedná se o žádnou formu rady pro správu prostředků, tento algoritmus nebyl vytvořen za účelem generování zisku.

K realizaci je využit rozhodovací model Market Buy Market Sell, který se rozhoduje na základě výstupního signálu dříve uvedené sítě indikátorů v kapitole 4.4.

Obrázek 6.1: Průběh Demo algoritmu v čase

Kapitola 7

Smysl informačního systému

V rámci této kapitoly bych se rád zamyslel nad užitkem vytvářeného informačního systému a porovnám vznikající koncept s existujícími řešeními.

7.1 Podobná řešení

Existuje celá řada veřejně dostupných a zpoplatněných informačních systémů, které provozují obchodní algoritmy na kryptoměnových burzách. Příkladem mohou být systémy Coinrule a Pionex, které v rámci této kapitoly pro představu popíši, ale také celá řada dalších: 3Commas, Quadency, Cryptohopper, Haasonline atd.

Všechny tyto systémy řeší za jejich uživatele potřebu automatizovaně obchodovat. Hlavním zdrojem financování takového systému je tedy jejich uživatel, který je buď motivován ke koupi tarifu pro používání vylepšených funkcí těchto systémů, a nebo mu jsou účtovány poplatky za realizaci objednávek nad výši standardních burzovních poplatků.

Systém tvořený v rámci této práce nemá za cíl konkurovat těmto veřejným řešením, ale naopak nabídnout alternativní privátní řešení, které bude schopno financovat sebe sama, a to na základě:

- minimálních nákladů nutných pro provoz systému (v porovnání s placenými tarify veřejných systémů),
- co nejvyšší efektivitě využití výpočetních prostředků,
- hostování dobře prověřených konceptů algoritmů s předem vyhodnocenými riziky, a
- schopnosti poskytnout privátní a dobře zabezpečené prostředí pro subjekty, které budou systém využívat.

7.1.1 Coinrule

Informační systém Coinrule umožňuje uživateli vytvářet předpisy (angl. „rules“) prostřednictvím uživatelského rozhraní. Předpis se skládá zpravidla ze stavové podmínky, která v případě splnění aktivuje stavovou událost. Stavové podmínky a události na sebe mohou sekvenčně navazovat, přičemž celý předpis lze opakovat či jeho aktivaci načasovat. Uživatel propojí předpis se zvolenou burzou prostřednictvím API klíčů, samotné obchody potom systém realizuje prostřednictvím API na burze.

Stavová podmínka se může týkat aktuální situace na trhu jednoho či více aktiv (např. aktuální kurz, obchodovaný objem, tržní kapitalizace) nebo omezeného počtu poskytovaných technických indikátorů. Stavová událost může vytvořit *market* a *limit* objednávku na aktuálním cenovém kurzu dvojice aktiv a nebo notifikovat uživatele.

Slabinou tohoto systému může být právě jeho otevřenost uživatelům – uživatel se základní znalostí obchodování sice může vytvořit v uživatelském prostředí obchodní strategii, nicméně volba parametrů stavových podmínek je značně omezená, neboť se nejedná o programovací jazyk, a tedy není možné vytvářet specializované obchodní algoritmy, které by se mohly řídit méně častými nebo na míru vytvořenými technickými indikátory. Prostředí jako takové zároveň nepomáhá uživatelům s volbou vhodné obchodní strategie, naopak spoléhá na ochotu a kreativitu uživatele, který do systému musí vložit vlastní strategii.

7.1.2 Pionex

Kryptoměnová burza Pionex umožňuje provádět základní *market* a *limit* objednávky a dále poskytuje několik specializovaných objednávek a investičních nástrojů. Burza dále poskytuje několik obchodních robotů (Grid Trading Bot, Dollar Cost Averaging Bot, Rebalancing Bot, Spot-Futures Arbitrage Bot), které si uživatel může nastavit dle vlastních preferencí a ochotě riskovat. Dostupné je zároveň doporučené nastavení pro uživatele, kteří se podrobně nezabývají obchodováním.

Obchodní roboty burzy Pionex již nějakou dobu testuji, neboť se jedná minimálně na první pohled o zajímavý finanční produkt. Problém, který zde vidím, je nižší výnosnost oproti mnou testovaným obchodním algoritmům a dále fakt, že roboti nejsou optimalizovaní například pro dlouhodobě klesající trh. Z velké části se jedná o koncepty, které fungují za ideálních podmínek trhu, ale odpovědnost za řešení krizových scénářů zůstává již na jejich uživateli. Z mého pohledu se tedy nejedná o plnohodnotné implementace obchodních strategií.

7.2 Požadavky na informační systém

Hlavním účelem informačního systému, který budu vytvářet, je realizace *obchodních algoritmů* na základě konceptů, které jsem definoval v této práci. Jedná se tedy především o *duální* obchodní algoritmy s *úplnou správou prostředků*.

Obchodní algoritmy budou interně definovány v informačním systému, přičemž jejich definice nebude přístupná uživateli.

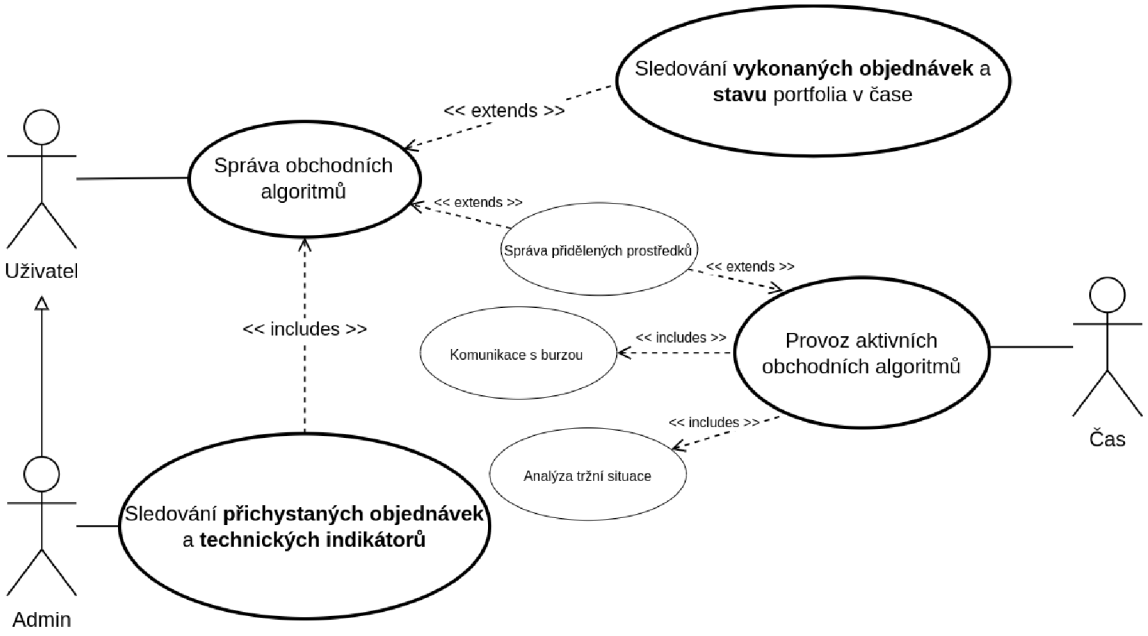
Informační systém umožní uživateli propojit vlastní účet s účtem na burze Binance pomocí API klíčů. Následně bude uživatel moci založit instanci *obchodního algoritmu* vybraného typu a přidělit mu prostředky dostupné na burze. U aktivního obchodního algoritmu bude moci:

- uživatel:
  - sledovat sledovat historii kurzu obchodované dvojice spolu s historií hodnoty přidělených prostředků, které byly algoritmu přiřazeny;
  - vidět, kdy došlo k provedení objednávek na burze;
  - pozastavit algoritmus a nebo upravit hodnotu přidělených prostředků;
- administrátor:

– provádět všechny akce, které může provádět uživatel;
– vidět přichystané, dosud neprovedené objednávky;
– vidět technické indikátory, na základě kterých došlo k vytvoření objednávek.

V rámci informačního systému bude kladen velký důraz na automatizaci, prevenci chyb a jejich reportování. Menší důraz bude kladen na kvalitu uživatelského rozhraní, neboť se bude jednat o soukromou část systému, která bude dostupná pouze vybraným uživatelům.

Systém neumožní uživatelům registrovat se zvenčí a nebude obsahovat veřejnou část, která by byla zaměřena například pro prezentaci systému veřejnosti – jedná se o možná budoucí rozšíření, která budou realizována později, pouze pokud se potvrdí funkčnost, spolehlivost a výnostnost systému a budou vyřešeny všechny právní a ekonomické aspekty jeho provozu.



Obrázek 7.1: Use case diagram shrnující hlavní požadavky na systém

Informační systém by měl být do budoucnosti rozšiřitelný:

- o další kryptoměnové burzy či brokery;
- o další moduly pro analýzu tržní situace, které budou moci být založeny i na jiných principech, než na technické analýze;
- a o další modely obchodních algoritmů, které nebudou *duální* nebo nebudou založeny na *úplné správě prostředků*.

Kapitola 8

Technologie

V rámci této kapitoly uvedu požadavky na vlastnosti informačního systému, na základě kterých vyberu technologie pro jeho realizaci.

8.1 Požadavky

Programové modely definované v kapitolách 5 a 6.2 by neměly být příliš výpočetně náročné, zvláště přihlédneme-li k faktu, že mým aktuálním cílem je realizovat obchodní algoritmy, které se budou rozhodovat na frekvenci 1 hodiny až 1 dne. V případě, že by informační systém v budoucnosti realizoval algoritmy na vyšších frekvencích a nebo by vyžadoval integraci nějakého výpočetně náročného modelu pro analýzu trhu, mělo by být možné uvažovat i o řešení paralelně vykonávaných procesů, jejichž programy mohou být vytvořeny případně i v jazycích nižší úrovně (C++, C), které je možné přeložit do jazyků symbolických instrukcí.

Prioritou informačního systému je naopak přehledná implementace, které může být často dosaženo i pomocí prvků funkcionálního programování. Definice informačního systému by nicméně měla být explicitní, a tedy spíše vylučuji možnosti využití přetěžování operátorů.

Přestože by systém neměl být příliš výpočetně náročný, mohou zde vznikat krátkodobé špičky při obsluze více obchodních algoritmů současně. Za klíčovou vlastnost systému bych tedy označil co nejvyšší propustnost a co nejnižší latenci v těchto případech. Beru v úvahu, že systém bude závislý na síťovém připojení k databázi a k API burzy. Jistě by bylo nežádoucí, pokud by celý informační systém byl vždy blokován při čekání na výsledek nějakého dotazu. Po srovnání možností paralelismu na úrovni procesů, paralelismu či konkurentního zpracování na úrovni vláken a konkurentního zpracování na událostmi řízené architektuře mi jako nejvhodnější způsob vychází událostmi řízená architektura. Díky této volbě odpadá celá řada problémů, která se týká synchronizace procesů, vláken a také řízení přístupu ke sdíleným zdrojům.

Problém, který bych dále rád eliminoval, je inicializace systému při příchodu každého požadavku (ať už síťového nebo lokální plánované události). Bylo by vhodné, aby proces informačního systému byl po celou dobu běhu inicializovaný, a byl tak schopen okamžitě na požadavky reagovat.

Posledním důležitým aspektem pro výběr technologií je čas nutný pro tvorbu systému. Tuto dobu je možné efektivně zkrátit výběrem dynamicky typovaného jazyka se slabou typovou kontrolou. Zároveň ale vnímám potřebu kód verifikovat, a tedy by bylo vhodné, aby

jazyk umožnil definici strukturovaných datových typů a zajistil statickou typovou kontrolu na všech místech, kde kontrola není z objektivních důvodů potlačena.

8.2 Backend

Na základě uvedených požadavků jsem se rozhodl serverovou aplikaci informačního systému realizovat v prostředí Node.js. Systém je implementován v jazyce TypeScript a je překládán do jazyka JavaScript, přičemž překlad zahrnuje statickou typovou kontrolu.

Pro realizaci jsem vybral framework Nestjs, který umožňuje tvorbu serverových aplikací. Framework zahrnuje vnitřní plánovač @nestjs/schedule, který umožňuje v rámci běžícího procesu plánovat akce pomocí stejného formátu, jaký známe ze souborů `crontab`.

Aplikace dále naslouchá na síťovém portu a přijímá HTTP dotazy. Pro komunikaci s uživatelskou aplikací jsem definoval schéma a obsluhu GraphQL rozhraní, které stanoví formát dotazů a strukturu odpovědi ve formátu JSON. Důvodem volby tohoto rozhraní je:

- častý problém s dokumentací čistých REST rozhraní, který je zde vyřešen pomocí GraphQL playground prostředí;
- možnost definice datových typů, které jsou validovány při vstupu do systému a při výstupu z něj.

Pro ukládání a zpětné prohledávání dat je využita relační databáze MariaDB. Tuto databázi vybírám jako logické open-source pokračování MySQL, které obsahuje řadu vylepšených funkcí a optimalizací. Relační databázi, omezení nad daty a transakční zpracování považuji za absolutní nutnost v systému této aplikace.

K interakci s databází je využita knihovna TypeORM, která zajišťuje objektově relační mapování a poskytuje celou řadu dalších pokročilých funkcí.

8.3 Frontend

Pro realizaci uživatelského rozhraní informačního systému jsem vybral framework React, který umožňuje deklarativním způsobem definovat jednotlivé komponenty uživatelského rozhraní formou stromu JSX. Každá komponenta zároveň může zahrnovat asynchronně vykonávanou aplikační logiku. Tento framework zajišťuje konverzi aplikačního stromu na strom DOM ve webovém prohlížeči. Framework dále zajišťuje aktualizaci stromu DOM v případě, že se změní data vykreslovaná aplikací.

Aplikace samotná je implementována opět v jazyce TypeScript. Díky tomu je možné sdílet implementaci některých částí kódu mezi frontendem i backendem.

Nejen z bezpečnostních důvodů bych rád aplikaci, alespoň nyní, distribuoval jako desktopový balíček, který bude vyžadovat instalaci. K tomu si dovolím využít framework ElectronJS, který dokáže aplikaci spustit v rámci okna integrovaného prohlížeče Chromium.

Pokud by měl být systém v budoucnu otevřen veřejnosti, nebude téměř nutné upravovat aplikační logiku. Z aplikace by v takovém případě byl odstraněn framework ElectrojJS a aplikace by byla zpřístupněna na veřejné URL adrese tak, aby ji mohl načíst libovolný webový prohlížeč.

Kapitola 9

Návrh a implementace informačního systému

V této kapitole popíši návrh informačního systému, který odpovídá jeho implementaci. Návrh i implementaci systému jsem řešil iterativním způsobem. Uvedené kapitoly tedy již zahrnují změny návrhu, ke kterým bylo nutné přistoupit v důsledku okolností, na které jsem narazil při implementaci a testování. Jedná se tedy o jakousi vysokoúrovňovou dokumentaci, přičemž detailní záležitosti jsou dokumentovány na úrovni zdrojového kódu.

Programové modely definované v kapitolách 5 a 6.2 jsou pochopitelně v rámci informačního systému také zahrnuty. Jedná se spíše o obecné koncepty a v rámci této kapitoly popíši tedy jejich integraci.

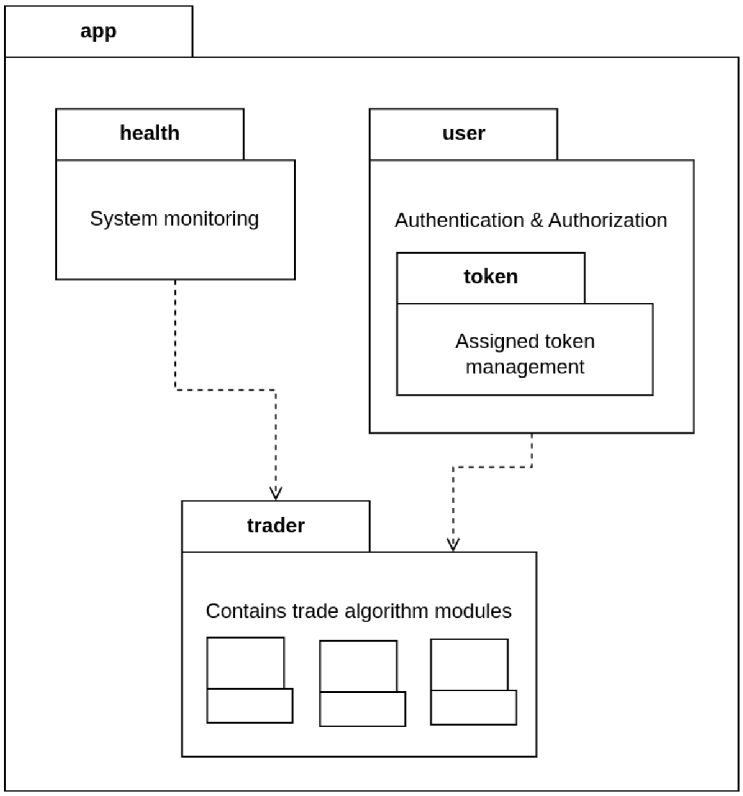
9.1 Backend

Aplikace vytvářené ve frameworku Nestjs mají přirozeně hierarchickou modulární strukturu. V rámci této kapitoly popíši návrh jednotlivých modulů, které jako celek budou fungovat jako jeden monolitický informační systém.

Hlavním modulem systému je modul automatizovaného obchodníka, `trader`. Tento modul poskytuje veškeré nutné komponenty pro realizaci *bezstavového modelu obchodního algoritmu*.

Vedlejší modul uživatele, `user`, řeší autentizaci a autorizaci uživatelů. Po přihlášení pomocí e-mailu a hesla obdrží uživatel náhodně vygenerovaný token s omezenou dobou platnosti pro prokázání vlastní identity. Jeho uložení a správu zajišťuje submodul `token`. Rozhraní GraphQL pro komunikaci s klientskou aplikací striktně rozlišuje roli uživatele, přičemž jedinou veřejnou operací je mutace `login(email, password)`.

Vedlejší modul `health` zajišťuje veřejný monitoring chodu systému.

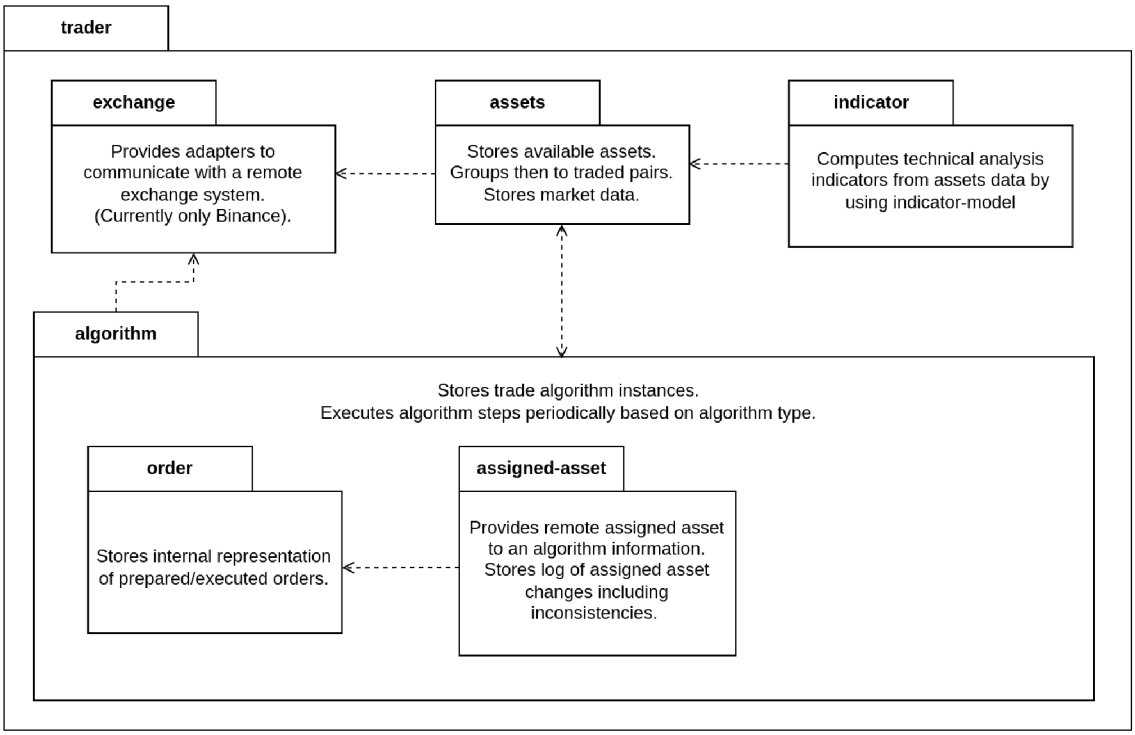


Obrázek 9.1: Package diagram reprezentující rozložení modulů na nejvyšší úrovni

9.1.1 Modul automatizovaného obchodníka

Modul automatizovaného obchodníka, `trader`, obsahuje všechny potřebné komponenty pro realizaci *bezstavového modelu obchodního algoritmu*, a to ve formě služeb pro zajištění běhového prostředí a synchronizaci objednávek, a dále ve formě submodulů, které řeší dílčí potřebné úkony, týkající se zejména:

- komunikace s burzou,
- ukládání burzovních dat,
- inicializace a zajištění kontinuity výpočtů v rámci *modelu technických indikátorů*;
- ukládání všech změn množství přidělených prostředků v čase, které mohou být způsobeny manuálním zásahem uživatele, a nebo v důsledku detekce chybějících lokálně přidělených prostředků ve vzdáleném systému burzy;
- ukládání všech připravených, zrušených a provedených objednávek.



Obrázek 9.2: Package diagram reprezentující rozložení submodulů modulu `trader`

Submodul `exchange` poskytuje jednotné rozhraní pro asynchronní komunikaci se vzdáleným systémem burzy. K tomuto je využit návrhový vzor adaptér, přičemž konečné rozhraní pro komunikaci s burzou stanoví burza – může se jednat například o HTTP rozhraní, knihovní třídu či funkční rozhraní, které burza poskytne. Rozhraní je zatím, v počáteční fázi projektu, implementováno pro burzu Binance. Ke komunikaci s burzou je využita oficiální knihovna @binance/connector, kterou Binance poskytuje pro Node.js (přestože by bylo možné využít existující HTTP rozhraní). Modul zároveň definuje pro burzu Binance slabou entitní množinu `BinanceSettings`, která se váže na uživatele a umožní uchování klíčů uživatele tak, aby automatizovaný obchodník mohl v systému burzy uživatele zastupovat.

Submodul `assets` definuje entitní množiny: `Asset`, `Pair` a `Candle`. Entita `Asset` reprezentuje obecně aktivum, které má nějakou hodnotu. Entita `Pair` reprezentuje vzájemně směnitelnou dvojici aktiv, přičemž pokud by došlo k rozšíření systému o další burzu, musela by tato entita být navíc identifikována burzou, kde ke směně dochází. Entita `Candle` se váže na konkrétní časové razítko počátku svíce; časovou jednotku, kterou svíce popisuje; a konkrétní `Pair`. Reprezentuje jednu jednotku burzovních dat odpovídající jedné svíci ve svícnovém grafu, která je obohacená o případné další informace poskytované burzou (např. obchodovaný objem aktiv, počet provedených obchodů, aj.). Do lokálního systému budou pravidělně synchronizovány svíce z burzy 1.) z důvodu optimalizace pro častější využití a 2.) jako důkaz zpětné neměnnosti burzovních dat.

Submodul `indicator` integruje model indikátorů uvedený v kapitole 5. Zajišťuje inicializaci a kontinuitu výpočtu sítí indikátorů nad aktuálně dostupnými kompletními svícemi modulu `assets`.

Modul `trader` dále obsahuje submodul obchodních algoritmů (`algorithm`). Ten zajišťuje evidenci obchodních algoritmů a dalších informací, které se týkají provozu instancí algoritmů.

9.1.2 Modul obchodních algoritmů

Modul `algorithm` definuje entitní množinu `Algorithm`, jejíž odvozené entity reprezentují instance obchodních algoritmů. Eviduje se časové razítko poslední úspěšné synchronizace instance. Každý algoritmus je možné pozastavit. Modul dále obsahuje submoduly `order` a `assigned-asset`.

K aktivaci algoritmu je nutné přidělit algoritmu vzdálené prostředky na burze. K tomu slouží submodul `assigned-asset`. Tento modul definuje dvě entitní množiny – `AssignedAsset` a `AssignedAssetChange`. Entita odvozená od množiny `AssignedAsset` reprezentuje aktuální množství jednoho přiděleného prostředku jedné instanci algoritmu. Entita odvozená od množiny `AssignedAssetChange` reprezentuje změnu v čase jednoho přiděleného prostředku u jedné instance algoritmu, kterou může v systému zařídit uživatel manuálním způsobem a nebo k ní může dojít automaticky v důsledku nekonzistence množství přiděleného prostředku a jeho skutečného stavu na burze. Algoritmus pro detekci nekonzistencí a automatickou úpravu přidělených prostředků nebudu uvádět z důvodu jeho složitosti, nicméně spočívá v:

- identifikaci aktiv, jejichž volný objem v lokálním systému přesahuje volný objem na burze, a to nad všemi algoritmy uživatele,
- pokud existuje alespoň jeden takový případ, poté je stanoven celkový rozdíl volných prostředků na burze a těch v lokálním systému, a
- v poměru dle celkového množství přiděleného prostředku všem instancím algoritmů a individuálnímu množství přidělenému každé instanci jednotlivě je stanoveno množství, které je každému algoritmu odebráno,
- a to vše je řešeno v rámci jedné databázové transakce.

Tento algoritmus tedy odebírá prostředky všem instancím obchodních algoritmů v lokálním systému rovnoměrně a týká se pouze algoritmů a prostředků, které nemají prostředky vázané ve formě otevřených objednávek na burze.

Submodul `order` dále slouží k evidenci připravených, vykonaných a zrušených objednávek na straně lokálního systému. K tomu slouží entitní množiny `Order` a `OrderFill`. Objednávky budou vždy nejdříve vytvořeny v lokálním systému a uloženy. Následně budou vytvořeny v příslušném systému burzy prostřednictvím modulu `exchange` a synchronizovány do lokálního systému. Pokud není objednávka dokončena ihned po vytvoření, bude ve stavu *otevřena (open)* a bude pravidelně do lokálního systému synchronizována, dokud nebude dokončena nebo zrušena. Vzhledem k tomu, že objednávky mohou být v systému burzy Binance rozděleny na více částí realizovaných za odlišný kurz a zároveň mohou být pouze částečně dokončené, entity odvozené od `OrderFill` slouží k evidenci jednotlivých převedených objemů prostředků za konkrétní kurzy a váží se vždy na konkrétní objednávku.

9.2 Testování systému

Testování systému byla činnost, která tvořila nadpoloviční část mojí práce, a tedy si zaslouží alespoň krátkou kapitolu. Automatizované testovací sady jsem vytvářel pouze pro backend systému. Funkčnost frontendu jsem ověřoval pouze manuálním způsobem při implementaci, neboť se 1.) nejedná o kritickou část infrastruktury a za 2.) se nepředpokládá se využití uživatelského rozhraní větším množstvím (neinformovaných) uživatelů.

K testování backendové aplikace jsem využil prostředí Jest a další podpůrné knihovny jako jest-extended a @nestjs/testing. Zaměřil jsem se přitom na testování sémantiky jednotlivých modulů a služeb systému, s vlastní znalostí vnitřní struktury, a na ověření funkčnosti očekávaných případů užití. Na základě měření dosahují testy následujícího průměrného pokrytí celého systému.

Pokrytí výrazů programu	79.74 %
Pokrytí větví programu	66.22 %
Pokrytí funkcí	66.95 %

Tabulka 9.1: Celkové pokrytí systému testy

Kritické části systému poté jsou pokryty zpravidla mezi 90–100 %, přičemž výjimky z nich tvoří zejména služby, které jsou příliš svázány s reálným časem či externími závislostmi, a tedy u nich bylo upřednostněno manuální testování v kombinaci s logováním činnosti za běhu. Výskyt chyb v celém systému za běhu je dále monitorován pomocí služby Sentry.

Velká část testů vyžaduje pro ověření správnosti systému využití databáze včetně některých konkrétních entit. Pro tento účel systém zahrnuje infrastrukturu databázových migrací, které se liší s ohledem na prostředí, ve kterém je systém spuštěn. Samotná databáze MariaDB pro testování je dále realizována v prostředí docker containeru tak, aby její verze byla shodná s produkčním verzí.

Velká část testů vyžaduje síťové připojení a komunikaci s burzou. Komunikace s burzou je rovněž testována, a to jak po sémantické stránce, tak pomocí porovnání výstupů komunikace pomocí metody *Snapshot Testing*. Pokud je zapotřebí nasimulovat výstup síťové komunikace, který s ohledem na časově omezenou dobu testu či omezení testovacího API burzy není možné obdržet, využívají se zde *Mock* implementace.

9.3 Frontend

Strukturu uživatelského rozhraní jsem navrhnul formou pracovního wireframe ve veřejně dostupném nástroji Figma a následně jsem jej konzultoval s vedoucím. S ohledem na uzavřenost systému uživatelům jsem se rozhodl nerealizovat grafický návrh aplikace a také nerealizovat jakékoli průzkumy, které by měly za cíl optimalizovat použitelnost UI. Díky zvoleným UI nástrojům, které jsem využil při implementaci, by přesto grafická stránka projektu neměla limitovat uživatele v použití systému.

Frontend informačního systému je implementován v jazyce TypeScript s použitím frameworku React pro definici dynamického uživatelského rozhraní v rámci webového prohlížeče. Z aplikace je vytvořen balíček s pomocí knihovny a pluginů Webpack. Konfigurace balíčku je inspirována konfigurací react-scripts@5.0.1 a je upravena tak, aby mohla být aplikace integrována v rámci frameworku ElectronJS. Díky tomu může být desktopová aplikace distribuována na většině operačních systémů.

Frontend aplikace komunikuje se serverovou aplikací prostřednictvím aplikačního protokolu GraphQL nad HTTPS. Tato komunikace, včetně slučování více požadavků do jediného HTTPS dotazu, kešování výsledků odpovědí a následné revalidace dat formou pollingu, je řešena s pomocí knihoven Apollo GraphQL.

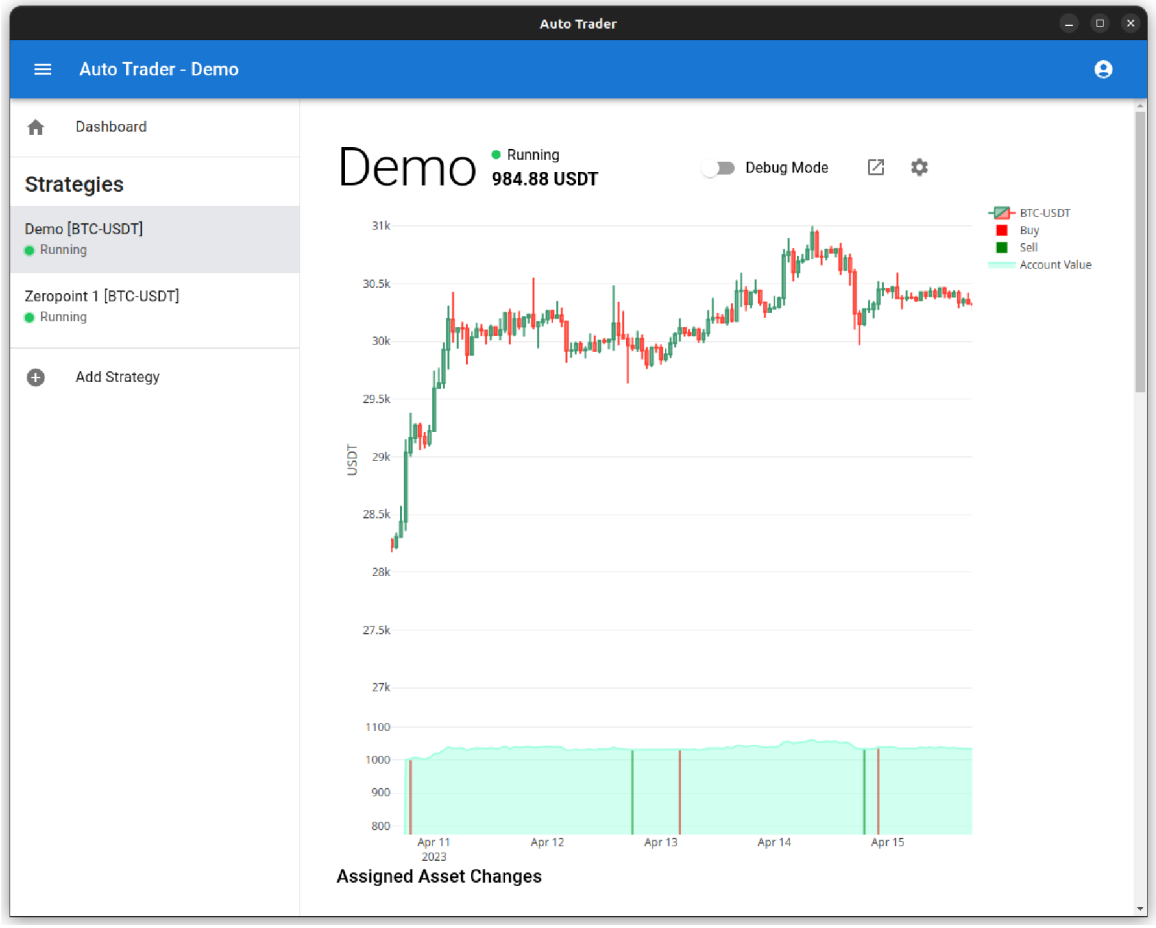
Uživatelské rozhraní je poté implementováno za pomocí UI frameworku Material UI. Aplikace implementuje případy užití uživatelů definované v kapitole 7.2, přičemž jejich

implementace ve formě komponent frameworku React a s využitím konceptu React Hooks je spíše přímočará. Sdílené funkcionality mezi větším počtem komponent aplikace, jako je například přihlášení uživatele a přizpůsobení aplikace jeho roli v systému, jsou řešeny pomocí konceptu React Context.

K rozlišení více stránek aplikace je využita knihovna react-router. Vzhledem k výchozímu omezení frameworku ElectronJS, které se týká provozu aplikace na jediné cestě URL adresy (např. `/main_window/index.html`), je zde využit koncept ukládání cesty stránky aplikace jako parametr *hash* URL adresy, a tedy výsledná cesta po zahrnutí parametru může vypadat následovně: `/main_window/index.html#/strategy/demo-1`.

Vizualizace burzovních dat a informací týkajících se obchodních algoritmů je řešena s pomocí knihovny Plotly.js, přičemž:

- pro vizualizaci cenových kurzů je využit Candlestick graf,
- pro vizualizaci provedených objednávek je využit Sloupcový (Bar) graf a
- pro vizualizaci hodnoty přidělených prostředků v čase je využit Bodový (Scatter) graf.



Obrázek 9.3: Příklad části uživatelského rozhraní – stránka instance Demo algoritmu

Kapitola 10

Závěr

V rámci práce jsem zavedl některá omezení obecných obchodních algoritmů tak, aby následně bylo možné vytvořit programový model pro realizaci širokého spektra obchodních algoritmů založených na technické analýze tržního prostředí. Hlavním omezením obchodního algoritmu je práce pouze nad jedním, předem zvoleným, párem prostředků. Druhým zásadním omezením je úplná správa prostředků, která vychází z faktu, že navrhnutý rozhodovací model pracuje nezávisle na stavu jednotlivých instancí algoritmů.

Přesto tento model umožňuje realizovat poměrně komplexní obchodní algoritmy, neboť se ukazuje, že síť technických indikátorů v kombinaci s obecným předpisem rozhodovacího modelu funguje jako mocná konfigurační složka. Koncept výpočetního modelu indikátorů by bylo vhodné do budoucna rozšířit o možnost výpočtu indikátorů na základě vstupních svící různé délky, což je problém, který je zapotřebí vyřešit ze sémantického, syntaktického i výkonnostního hlediska. V budoucnosti také plánuji ověřit kompatibilitu sítě indikátorů a neuronových sítí.

Kromě popsaných modelů systém také zahrnuje částečně upozaděnou infrastrukturu, která umožňuje monitorování přidělených prostředků, cenových kurzů a komunikaci s burzou. Tuto infrastrukturu bude možné v budoucnosti využít i pro realizaci odlišných modelů obchodních algoritmů, pro které již nebudou platit uvedená omezení, a které budou moci být založeny i na jiné než technické analýze trhu.

Nakonec, přirozeným rozšířením systému by mohla být integrace více kryptoměnových burz, či případně systémů brokerů pro automatizované obchodování na akciových či forexových trzích.

Výsledkem této práce je informační systém, který v reálném čase provádí analýzu trhu. Z burzovních dat se snaží získat znalosti, které mají charakter předpokladů pro následující vývoj, a na jejich základě se rozhoduje a provádí obchody na burze nad instancemi obchodních algoritmů v zastoupení uživatelů.

Přestože cílem bylo původně realizovat systém, který bude vydávat a realizovat obchodní rozhodnutí nad periodou 1 hodiny až 1 dne, systém by měl bezpečně zvládnout i periodu 1 minuty. Efekty vysokofrekvenčního obchodování budu nicméně teprve zjišťovat a je možné, že realizace takových algoritmů bude vyžadovat i řešení detailů, jako je geografické umístění serveru, kde je systém provozován.

Literatura

[1] Appel, G. *Technical Analysis: Power Tools for Active Investors*. FT Press, 2005. 165–169 s. ISBN 0131479024.

[2] Binance. *What Is the Stop-Limit Function and How to Use It?* [online]. Prosinec 2017 [cit. 2023-01-03]. Dostupné z: https://www.binance.com/en/support/faq/what-is-the-stop-limit-function-and-how-to-use-it-115003372072.

[3] Binance. *What Is an OCO (One Cancels the Other) Order and How to Use It?* [online]. Srpen 2019 [cit. 2023-04-28]. Dostupné z: https://www.binance.com/en/support/faq/what-is-an-oco-one-cancels-the-other-order-and-how-to-use-it-360032605831.

[4] Binance. *What Are Market Order and Limit Order, and How to Place Them?* [online]. Červenec 2021 [cit. 2023-01-03]. Dostupné z: https://www.binance.com/en/support/faq/what-are-market-order-and-limit-order-and-how-to-place-them-12cba755d6334ad98ced0b66ddde66ec.

[5] Binance. *How to Use Spot Trailing Stop Order* [online]. Duben 2022 [cit. 2023-04-28]. Dostupné z: https://www.binance.com/en/support/faq/how-to-use-spot-trailing-stop-order-339635f6260d43c5aefa4c3c921728ec.

[6] Binance. *Binance API Documentation* [online]. Leden 2023 [cit. 2023-01-11]. Dostupné z: https://binance-docs.github.io/apidocs/spot/en/#new-order-trade.

[7] Brooks, A. *Trading Price Action Trends: Technical Analysis of Price Charts Bar by Bar for the Serious Trader*. Wiley, 2011. 33 s. Wiley Trading. ISBN 9781118166253.

[8] Bulkowski, T. N. *Encyclopedia of Chart Patterns, 3rd Edition*. Wiley, 2021. ISBN 9781119739685.

[9] Chen, J. *What is EMA? How to Use Exponential Moving Average With Formula* [online]. Investopedia.com, březen 2023 [cit. 2023-04-16]. Dostupné z: https://www.investopedia.com/terms/e/ema.asp.

[10] Connors, L. a Alvarez, C. *An Introduction to ConnorsRSI (Connors Research Trading Strategy Series)*. 2nd. Jersey City: Connors Research, 2014. 7–9 s.

[11] Crypto.com. *Crypto.com Exchange API v1 Documentation* [online]. Duben 2020 [cit. 2023-01-10]. Dostupné z: https://crypto.com/exchange-docs-v1.

[12] Hayes, A. *Simple Moving Average (SMA): What It Is and the Formula* [online]. Investopedia.com, únor 2022 [cit. 2023-04-16]. Dostupné z: https://www.investopedia.com/terms/s/sma.asp.

[13] Majaski, C. *Limit Order vs. Stop Order: What's the Difference?* [online]. Investopedia.com, duben 2023 [cit. 2023-04-24]. Dostupné z: https://www.investopedia.com/ask/answers/04/022704.asp.

[14] Murphy, J. J. *Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications*. 2nd. Prentice Hall Press, 1999. 5–6 s.

[15] Nison, S. *Japanese Candlestick Charting Techniques*. 2nd. Prentice Hall Press, 2001. 23–25 s. ISBN 978-0735201811.

[16] OKX. *OKX REST API Documentation* [online]. Leden 2023 [cit. 2023-01-10]. Dostupné z: https://www.okx.com/docs-v5/en/#rest-api-trade-place-order.

[17] Pinkasovitch, A. *Types of Rebalancing Strategies* [online]. Feb 2022 [cit. 2022-11-13]. Dostupné z: https://www.investopedia.com/articles/stocks/11/rebalancing-strategies.asp.

[18] Salas, G. *Crypto.com Stop-Loss Bot* [online]. Github.com, Jul 2021 [cit. 2023-01-10]. Dostupné z: https://github.com/giansalex/crypto-com-stoploss.

[19] Thomsett, M. *Support and Resistance Simplified*. Traders' Library: Marketplace Books, 2003. 17–21, 115, 117–118 s. ISBN 978-1592800674.

[20] Wilder, J. W. *New Concepts in Technical Trading Systems*. Trend Research, 1978. ISBN 978-0-894-59027-6.

[21] Zandt, F. *Binance Dominates Crypto Exchange Landscape* [online]. Nov 2022 [cit. 2022-12-13]. Dostupné z: https://www.statista.com/chart/28721/cryptocurrency-exchanges-with-the-highest-trading-volume-year-to-date/.