

UNICORN VYSOKÁ ŠKOLA S.R.O.

Informačné technológie



BAKALÁRSKA PRÁCA

Technická analýza automatického obchodného systému

Autor BP: Jozef Randjak

Vedúci BP: Ing. Pavel Bory

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Jméno a příjmení:	Jozef Randjak
Studijní program:	Informační technologie
Studijní obor:	Informační technologie
Název práce:	Technická analýza automatického obchodního systému

CÍL

Cieľom projektu je implementovať a zanalyzovať automatický obchodný systém popísaný v knihe Short Term Trading Strategies That Work od Larryho Connorsa. Na základe historických dát akciových titulov, uskutočniť backtest a analýzu výkonnosti a rizikovosti obchodného systému a zhodnotiť zmysel použitia obchodného systému v praxi. Daný systém bude obohatený o obchodovanie nie na jeden akciový titul, ale na vybraný koš akciových titulov. Takýto prístup zahrňuje simultánne vyhodnotenie stovky akciových titulov, kde sa z vybraného koša vždy vyberie taký titul, ktorý najviac vyhovuje podmienkam daného systému.

Na základe historických výsledkov z obchodného systému, bude realizovaná analýza jeho výkonnosti, v porovnaní s akciovým indexom SPY a voľne dostupnými investičnými nástrojmi bank. Takisto posúdime vhodnosť použitia obchodného systému v reálnom prostredí, najvhodnejšieho času nasadenia systému na akciový trh a objasníme rizika, ku ktorým môže dojsť obchodovaním pomocou daného systému. Systém bude implementovaný v .NET Core a poskytne API s výstupnými dátami ktoré budú následne analyzované v MS Excel.

OSNOVA

ÚVOD
2.1.1 Motívacia
TEORETICKÁ ČASŤ
2.1.2 Akciové trhy
2.1.3 Úvod do obchodovania akcií
2.1.4 Popis analyzovanej stratégie
PRAKTICKÁ ČASŤ
2.1.5 Návrh dátového modelu
2.1.6 Implementácia API
2.1.7 Získanie historických burzových dát
2.1.8 Implementácia obchodného systému

DOPORUČENÁ LITERATURA

- Larry Connors, C. A. (2009). Short Term Trading Strategies That Work. United States of America: ISBN 978-0-9819239-0-1.
- Lock, A. (2018). ASP.NET Core in Action. Chicago: ISBN 9781617294617.
- McConnell, S. (2004). Code Complete Second Edition. Redmond, Washington: Microsoft Press.
- Skeet, J. (2019). C# in Depth. Manning Publications.

Vedoucí bakalářské práce: Pavel Bory

Adresa pracoviště: V Kapslovně 2767/2, 130 00 Praha 3

Datum zadání bakalářské práce: 07.08.2019

Termín odevzdání bakalářské práce: Podle rozhodnutí rektora

V Praze dne 14.07.2021



.....
doc. Ing. Jan Čadil, Ph.D.
Rektor

Čestné prehlásenie

Prehlasujem, že som svoju bakalársku prácu na tému Technická analýza automatického obchodného systému vypracoval/a samostatne pod vedením vedúceho bakalárskej práce a s použitím výhradne odbornej literatúry a ďalších informačných zdrojov, ktoré sú v práci citované a sú taktiež uvedené v zozname literatúry a použitých zdrojov.

Ako autor/ka tejto bakalárskej práce ďalej prehlasujem, že v súvislosti s jej vytvorením som neporušil/a autorské práva tretích osôb a som si plne vedomý/á následkov porušenia ustanovení §11 a nasledujúcich autorského zákona č. 121/2000 Sb.

Ďalej prehlasujem, že odovzdaná tlačená verzia bakalárskej práce je zhodná s verziou, ktorá bola odovzданá elektronicky.

V dňa

(Jozef Randjak)

Pod'akovanie

Rád by som sa touto cestou pod'akoval vedúcemu bakalárskej práce Ing. Pavlovi Borymu za odbornú pomoc a ďalšie cenné rady pri spracovávaní mojej bakalárskej práce.



Technická analýza automatického obchodného systému

Technical analysis of automatic trading
system

UNICORN
UNIVERSITY

Abstrakt

Hlavným cieľom práce je technická analýza a implementácia automatického obchodného systému.

V teoretickej časti práca vysvetľuje základnú problematiku burzy, obchodovanie na burze, technickú analýzu obchodných systémov, analyzovanú obchodnú stratégiu a jej využitie. V závere teoretickej časti práca popisuje návrh automatického systému, použité technológie a metódy pomocou ktorých je takýto systém implementovaný.

V praktickej časti je realizovaná implementácia navrhnutého systému. Podrobne rozoberá spracovania dát, počítanie obchodných indikátorov a implementáciu obchodnej stratégie. Integruje potrebné nástroje pre analýzu obchodnej stratégie. Praktická časť na základe dosiahnutých výsledkov z reportov vygenerovaných implementovaným systémom odpovedá na otázku výkonnosti jednotlivých prístupov. Porovnáva výkonnosti stratégie pomocou stanovených základných ukazovateľov, zamýšľa sa nad najvhodnejším časom nasadenia stratégie na trh, objasňuje riziká spojené s obchodovaním a zvažuje použitie obchodného systému v reálnom prostredí.

Kľúčové slová: Burza, technická analýza obchodného systému, automatický obchodný systém, .NET Core, C #, domain-driven design

Abstract

The main goal of the thesis is the technical analysis and implementation of an automatic trading system.

In the theoretical part, the bachelor thesis explains the basic aspects of the stock exchange, trading on the stock exchange, technical analysis of trading systems, analysed trading strategy and its evaluation. At the end of the theoretical part, the thesis describes design of automatic trading system and the technologies and methods by which such a system is implemented.

In the practical part, the implementation of the proposed system is realized. It discusses in detail data processing, calculation of trading indicators and implementation of trading strategy. It integrates the necessary tools for strategy analysis. Based on the results obtained from the reports generated by the implemented system, the practical part answers the question of the performance of individual approaches. It compares the performance of the strategy using different established basic indicators, considers the most appropriate time to trade the strategy on the market, clarifies the risks associated with trading and considers the use of the system in a real environment.

Keywords: Stock exchange, technical analysis of trading system, automatic trading system, .NET Core, C #, domain-driven design

Obsah

Úvod	12
1. Burza.....	13
1.1 Význam burzy pre firmy.....	13
1.2 Typy derivátov.....	13
2. Úvod do obchodovania	16
2.1 Technická a fundamentálna analýza	16
2.1.1 Cenový graf	16
2.1.2 Trend	17
2.1.3 Patterny.....	18
2.2 Indikátory.....	18
2.2.1 Simple moving average (SMA).....	19
2.2.2 Relative strength index (RSI)	20
3. Analyzovaná stratégia	22
3.1 Prístup Larryho Connorsa založený na RSI indikátore.....	22
3.1.1 Identifikácia poklesu a nákup	22
3.1.2 Len akcie v dlhodobom rastúcom trende.....	22
3.1.3 Výstup z obchodu	23
3.1.4 Kombinácia pravidiel	23
3.1.5 Modifikácie stratégie	23
3.2 Dáta	24
3.3 Vyhodnocovanie	27
4. Automatický obchodný systém.....	30
4.1 Vybrané technológie	30
4.2 Použitý prístup pri návrhu systému	31
4.3 Návrh systému	31

4.3.1 Fyzický pohľad.....	32
4.3.2 Vrstvy	32
4.3.3 Web	33
4.3.4 Infraštruktúra	33
4.3.5 Zdieľané Jadro.....	34
4.3.6 Jadro	34
4.3.7 Dátový sklad.....	36
5. Praktická časť.....	38
5.1 Vývojárske nástroje a knižnice	38
5.2 Spracovanie dát.....	38
5.2.1 Integrácia Entity Frameworku	39
5.2.2 Implementácia dátového skladu	44
5.3 Príprava dát pre Backtest	48
5.3.1 Implementácia indikátorov	48
5.4 Implementácia obchodnej stratégie	51
5.4.1 API rozhranie	51
5.4.2 Hlavná trieda Strategy	53
5.5 Integrácia Excelu a Reporting.....	56
5.6 Integrácia Swagger API rozhrania	57
5.7 Optimalizácia parametrov a experimentovanie	58
5.7.1 Postup testovania.....	58
5.7.2 Test s jednoduchým priemerom a momentom	59
5.7.3 Test s pridaným dlhodobým priemerom	60
5.7.4 Test s optimalizáciou RSI indikátoru	61
5.7.5 Test s dokupovaním do existujúcich pozícii na základe RSI indikátoru.....	61
5.7.6 Test s dokupovaním do existujúcich pozícii pri nižšej cene	62
5.8 Vyhodnotenie prístupov.....	63

5.8.1 Testované koše akcií	64
5.8.2 Výkonné prístupy	64
5.8.3 Možné riziká.....	64
5.8.4 Obchodovanie v reálnom prostredí	65
Záver	67
Conclusion	69
Zoznam tabuľiek	71
Zoznam grafov	72
Zoznam obrázkov	73
Zoznam použitej literatúry	74
Zoznam použitých skratiek.....	76
Zoznam príloh.....	78

Úvod

S digitalizáciou služieb a rozširovaním internetu sa sprístupnila a zjednodušila možnosť obchodovania na burzových trhoch aj pre bežných zákazníkov. Dnes nie je problém otvoriť si účet v brokerskej spoločnosti a začať investovať do najväčších podnikoch sveta. S digitalizáciou sa veľmi rozšírilo aj algoritmické obchodovanie. Myšlienka je veľmi jednoduchá. Pomocou počítača a algoritmizácie som schopný vyhodnocovať obrovské množstvo obchodov v priebehu niekoľkých sekúnd a generovať obchodné príkazy. Či takýto prístup má zmysel a naozaj môže zlepšiť výnosnosť investovaného kapitálu sa budem snažiť zistíť v tejto práci.

V teoretickej časti priblížim obchodovanie na burze, základné deriváty a ukazovatele ktoré súvisia s obchodovaním. Budú vysvetlené algoritmy použité na vyhodnocovanie obchodov a vysvetlím základnú stratégiu a postupy ktoré budem realizovať v praktickej časti.

Cieľom praktickej časti bakalárskej práce je implementovať a zanalyzovať automatický obchodný systém popísaný v knihe Short Term Trading Strategies That Work od Larryho Connorsa. Na základe historických dát akciových titulov, uskutočniť backtest a analýzu výkonnosti a rizikovosti obchodného systému a zhodnotiť zmysel použitia obchodného systému v praxi. Daný systém bude obohatený o obchodovanie nie na jeden akciový titul, ale na vybraný kôš akciových titulov. Takýto prístup zahrnuje simultánne vyhodnotenie stovky akciových titulov, kde sa z vybraného koša vždy vyberie taký titul, ktorý najviac vyhovuje podmienkam daného systému.

Na základe historických výsledkov z obchodného systému, bude realizovaná analýza výkonnosti, v porovnaní s akciovým indexom Standard and Poor's 500 Index (SPY). Rovnako zhodnotím najvhodnejší čas nasadenia systému na akciový trh a budem sa snažiť identifikovať rizika, ku ktorým môže dôjsť obchodovaním pomocou takéhoto systému. Systém bude implementovaný vo Frameworku .NET Core a poskytne Application Programming Interface (API) s výstupnými dátami ktoré budem následne analyzovať v programe MS Excel.

1. Burza

Burzu môžem charakterizovať ako miesto kde sa stretávajú obchodníci ktorí nakupujú a predávajú rôzne typy derivátov za cenu stanovenú trhom. V minulosti takýto obchod vyžadoval agenta, ktorému obchodník zavolal a u ktorého si zistil všetky potrebné informácie. Tento agent následne zadal príkaz na takzvaný PIT čo bolo miesto kde sa realizovali obchody vo forme nákupného certifikátu a týmto bola dokončená obchodná transakcia. S postupom času sa ale tento proces začal digitalizovať. Dnes obchodného agenta vymenil broker, ktorý disponuje obchodnou platformou kde si obchodník môže jednoducho vyhľadať daný derivát a všetko realizovať online.

1.1 Význam burzy pre firmy

Je veľmi dôležité položiť si otázku prečo najväčšie spoločnosti na svete majú v záujme ponúknuť svoj podiel firmy širokej verejnosti. Princíp je jednoduchý. Hlavným cieľom je prínos kapitálu do podniku ktorá ponúka svoj podiel. Každá myšlienka, každý dobrý nápad potrebuje zdroje k tomu aby mohol byť realizovaný. Tieto zdroje v dnešnej dobe znamenajú ľudí, ktorých potrebujem zaplatiť, materiál, marketing, priestory, zásoby a ďalšie. Podnik má dve možnosti, požičať si od banky alebo ísť s podielom na burzu. Banka môže predstavovať problém, lebo podnik nedisponuje dostatočným majetkom, ktorý bude banka od neho vyžadovať ako protihodnotu. Samozrejme si za požičanie priráta aj úrok. Druhá možnosť je ísť s podielom na burzu a ponúknuť časť podniku verejnosti. Takto môže firma získať prístup k potrebnému kapitálu bez toho aby zaň platila úrok alebo dávala niečo ako protihodnotu v prípade že neuspeje.

1.2 Typy derivátov

- **Akcie:** „Nákup akcií je ako nákup podniku“. (1) Akcie sú cenné papiere ktoré predstavujú časť vlastníctva pre toho kto takéto cenné papiere vlastní. Z pohľadu nakupujúceho takto získam podiel vo firme, v ktorej vidím potenciál a nákupom jej časti sa môžem podieľať na jej raste čím bude rást samozrejme aj moja investícia. Z pohľadu spoločnosti si takto navýšim dostupný kapitál, s ktorým môžem ďalej operovať a rozvíjať sa. S obchodovaním akcií je spojené aj vyplácanie dividend. Dividendy sú odmeny pre vlastníkov akcií, ktoré sú nezávislé od ceny akcie a spoločnosť sa ich jednoducho rozhodla vyplácať, pretože dosahuje pozitívne hospodárske výsledky.

- **Opcie:** „Opcia je obchodovateľný cenný papier, ktorý dáva držiteľovi právo, nie však povinnosť, kúpiť alebo predať do určitého dátumu iný cenný papier za konkrétnu cenu“. (2)(str.166) Inými slovami, sú predkupné práva definované na presnú cenu a presne stanovenú časovú dobu. Opcie môžem nakupovať alebo predávať. Nákup opcie dáva kupujúcemu právo nakúpiť v danej časovej dobe derivát, na ktorý bola táto opcia vydaná. Napríklad, kúpim opciu Apple, ktorá vyprší za mesiac za cenu 5 dolárov na hodnotu akcie 130\$. Aktuálne sa cena Apple pohybuje 125\$ za kus. Ak cena v tomto mesiaci narastie napr. na 140\$ tak ja môžem moju opciu uplatniť a inkasovať tak akcie za cenu definovanú opciou, tj. 130\$ a 10 dolárov bude môj zisk. V konečnom dôsledku, môj celkový zisk je 5 dolárov. Samozrejme, cena narásť nemusí, ja sa rozhodnem neuplatniť opciu, tým ale prichádzam o 5 dolárov, ktoré som za danú opciu zaplatil. Jednoducho sú to práva a za tieto práva musím samozrejme zaplatiť. Opcie sa nemusia len nakupovať, ale dajú sa aj predávať, začo ako predávajúci inkasujem zisk. Rozdiel oproti nákupu je v tom, že namiesto práv mám povinnosť. To znamená, že inkasujem zisk za predaj opcie, ale v prípade, že bude druhá strana, ktorá mi za toto zaplatila, chcieť uplatniť tieto opcie, tak moju povinnosťou je tento obchod realizovať.
- **Komodity:** „Prírodne zdroje, suroviny používané vo všetkých druhoch produkcie“. (2)(str.163) Pod komoditami si je možné predstaviť surovinové kontrakty napr. pšeniciu, kukuricu, kávu, ropu alebo zlato a striebro. Vyznačujú sa zvýšenou volatilitou oproti akciám, niektorí obchodníci ale využívajú komodity k obchodovaniu v čase keď sa akciám nedarí, pretože majú zvyčajne protichodný pohyb. Na niektoré komodity je možné dokonca vypozorovať určité sezónne pohyby, napr. na základe počasia alebo dopytu v určitých mesiacoch.
- **Indexy:** Patria medzi najrozšírenejšie obchodovateľné deriváty na svete. Považujú sa aj za veľmi dobré ukazovatele vývoja ekonomiky. Sú to jednoducho „koše“, akcií. Ich zloženie sa môže zameriavať na rôzne oblasti. Medzi najznámejšie patria spomínaný index SPY, ktorý obsahuje 500 najväčších akciových firiem v Spojených Štátach. Nasdaq-100 index (SP100) podobne ako SPY, ale so 100 najväčšími firmami, Nasdaq 100 (QQQ), ktorý obsahuje prevažne technologické firmy, Vanguard Value Index (VTV) hodnotový index, ktorý je široko diverzifikovaný a má zastúpenie vo vyše 400 akciách.
- **Forex:** „Forex označuje globálny elektronický trh pre obchodovanie s medzinárodnými menami a menovými derivátmi“. (3) Patrí medzi oblúbene trhy hlavne pre začiatočníkov, známy aj pod názvom menový trh. Obchoduje sa 24 hodín denne od pondelka do piatku. Za

zmienku stojí, že sa dá obchodovať aj cez centralizovaný trh v podobe komoditných kontraktov. Patrí medzi najväčšie trhy na svete, extrémne reaguje na makroekonomickej a politické udalosti.

2. Úvod do obchodovania

Pri obchodovaní na finančných trhoch sa obchodníci rozdeľujú na dva tábory. Tí, ktorí sa zaoberajú fundamentálou analýzou a tí, ktorí sa skôr prikláňajú k technickej analýze. „Primárny rozdiel medzi týmito dvoma prístupmi spočíva v tom, že fundamentálna analýza sa zaoberá otázkou prečo, zatiaľ čo technická analýza sa zaoberá otázkou kedy“. (4)(str.51)

2.1 Technická a fundamentálna analýza

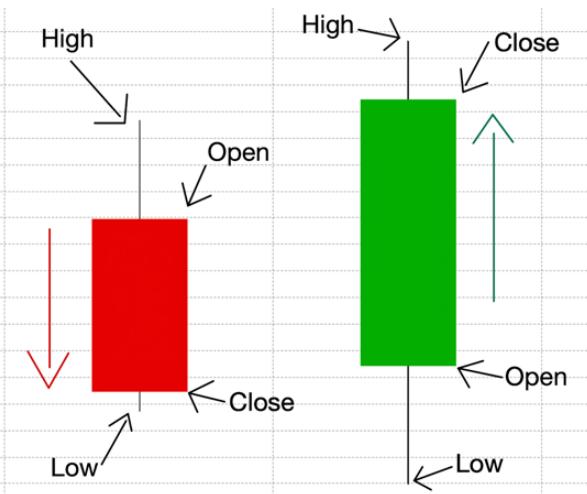
Fundamentálna analýza znamená vyhodnocovanie obchodov pomocou ekonomických ukazovateľov, finančných správ spoločností, aktuálneho vývoja ekonomiky, ale aj ukazovateľov ako sú počasie, politická situácia, import a export.

Technická analýza na druhej strane je analýza tzv. cenovej aktivity. Je to analýza aktuálneho vývoja ceny a snaha o identifikovanie cenových patternov, ktoré sa v minulosti vyskytovali. Vychádza sa z predpokladu, že cena má už v sebe započítané všetky fundamentálne ukazovatele a len odzrkadľuje aktuálnu náladu na trhu, ktorú identifikuje pomocou cenových indikátorov. (4)

2.1.1 Cenový graf

Graf je grafická reprezentácia dát. Cenový graf je pre obchodníka primárny nástrojom pre technickú analýzu. Cenový graf reprezentuje daný derivát a vývoj jeho ceny za obdobie špecifikované tzv. časovým úsekom grafu. To znamená, že existujú grafy, ktoré reprezentujú cenový vývoj za 10 rokov, 5 rokov, 1 rok ale aj 1 hodinu. Na X osy je zobrazené obdobie a na Y osy je zobrazený cenový vývoj daného derivátu. Existuje niekoľko druhov cenových grafov ako napr. barové, sviecové, čiarové grafy a veľa ďalších. Ja sa budem zaoberať hlavne sviecovým grafom, ktorý je najpoužívanejší.

Graf 1: Cenové sviece



Zdroj: Aplikácia Trader Workstation (vlastné spracovanie)

Na obrázku je zobrazený sviecový graf, ktorý je zložený z tzv. sviec, ktoré vyjadrujú štyri základné cenové informácie. Open znamená na akej cene sa začala akcia obchodovať pri otvorení trhov. Close znamená na akej cene sa akcia zastavila na konci obchodovania. High a Low označujú maximálne hodnoty na ktorých sa akcia obchodovala v priebehu dňa. Zelená svieca znamená, že akcia v daný deň skončila so ziskom, červená naopak so stratou.

2.1.2 Trend

Identifikácia začínajúceho trendu býva jedna z najlepších obchodných techník. Obchodníci sa snažia naskočiť na začínajúci trend a ukončiť ho v momente keď sa na grafe začínajú cenové sviece „ohýbať“. Začínajúci rastúci trend môžem v jednoduchosti charakterizovať ako cenu sviece, ktorá uzatvára vyššie ako predchádzajúci deň a najlepšie niekoľko dní za sebou. Naopak môže fungovať aj klesajúci trend kde cena uzatvára nižšie ako predchádzajúce dni. Samozrejme záleží na každom obchodníkovi ako si nastaví pravidlá pre určenie patternov, na základe ktorých bude realizovať obchodné príkazy.

Graf 2: Ročný graf rastu cien Amazonu s trendovou linkou



Zdroj: Aplikácia Trader Workstation (vlastné spracovanie)

Na grafe môžem vidieť identifikáciu rastúceho trendu ktorý začal začiatkom apríla a skončil v septembri. Pozdĺž ceny je znázornená trendová linka. Cena má za dané obdobie samozrejme aj klesajúce dni. Čo je ale pekne na grafe vidieť je, že nikdy za dané obdobie neklesla pod túto trendovú linku.

2.1.3 Patterny

Pri technickej analýze sa budem hlavne zameriavať na Patterny. Patterny sú cenové pohyby, ktoré sa v minulosti vyskytovali a môžem ich identifikovať na základe pevne daných pravidiel. Tieto pravidlá generujú signály na nákupné alebo predajné príkazy.

Je veľmi dôležité si uvedomiť že takýto signál je len pravdepodobnosť, ktorá bola vypočítaná na základe historických cien. To, že sa cena bude pohybovať predpokladaným smerom je len predpoklad.

2.2 Indikátory

Indikátory mi pomôžu identifikovať cenové patterny. K testovaniu obchodnej stratégie potrebujem presný súhrn pravidiel, kedy realizovať daný obchod. Bez týchto pevne daných pravidiel by som neboli schopný otestovať a vyhodnotiť takýto systém.

Indikátor je matematická formula aplikovaná na cenu daného derivátu. Pomáha identifikovať cenovú aktivitu a dokáže generovať signály. Existuje veľké množstvo cenových indikátorov ako napríklad: Simple Moving Averages (SMA), Relative Strength Index (RSI), Moving Average Convergence Divergence (MACD), Rate of Change (ROC) a veľa iných. Ja sa ale budem zaoberať hlavne dvoma: SMA a RSI.

2.2.1 Simple moving average (SMA)

„Klzávے priemery poskytujú veľmi jednoduchý prostriedok na vyhladenie cenových radov a zvýšenie viditeľnosti akýchkoľvek trendov“. (4)(str.51)

Čo chcel autor touto myšlienkovou povedať? Jednoducho mi tento indikátor pomôže spriemerovať ceny čím dostanem priamku na základe ktorej môžem generovať obchodné signály alebo filtrovať budúce obchody. Klzávý priemer počítam ako cenu za posledných n dní, končiacu súčasným dňom. Samozrejme sa cena nemusí počítať len z dní, výpočet priemera ovplyvní časový úsek grafu a typ zvolenej svieci napr. minútová, hodinová, denná atď.

Graf 3: Ročný graf rastu cien Amazonu so 100 dňovým cenovým priemerom



Zdroj: Aplikácia Trader Workstation (vlastné spracovanie)

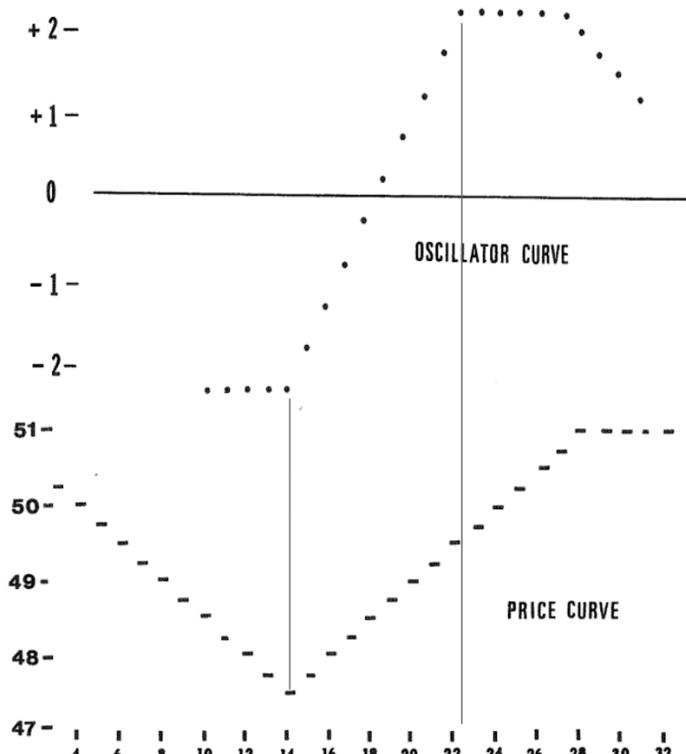
Na grafe je zobrazený klzávý priemer ktorý sa počíta z posledných 100 dní z denných close cien. Vidíme, že by tiež veľmi dobre poslúžil k identifikácii rastúceho trendu. Naroziel od trendovej linky by predajný príkaz vygeneroval až na konci novembra.

2.2.2 Relative strength index (RSI)

RSI sa radí medzi tzv. momentum indikátory. Momentum nám pomáha s určením sily ceny obchodovaného aktíva. Čo to znamená? Čím cena silnejšie stúpa alebo silnejšie klesá, tým bude momentum hodnota vyššia. Úloha takého momentum indikátoru je určiť silu napr. trendu, respektívne identifikovať extrémy v cenových pohybach. SMA reaguje na takéto extrémy oveľa pomalšie pretože sa jeho hodnota priemeruje s predchádzajúcich cien, momentum indikátory sú ale na tieto zmeny oveľa citlivejšie, čo mi pomôže lepšie načasovať a identifikovať nákupné, resp. predajné signály. Výpočet veľmi jednoduchého momentum indikátoru si môžem predstaviť ako rozdiel medzi dnešnou cenou a cenou z posledných n dní. To znamená čím je tento rozdiel väčší, tím je cenové momentum silnejšie a cena sa mení rýchlejšie.

RSI indikátor bol pôvodne vyvinutý J. Welles Wilderom v roku 1978. Dodnes sa však používa jeho pôvodný výpočet a považuje sa medzi kľúčové indikátory k technickej analýze. Jeho základné vlastnosti popisuje autor na nasledujúcom grafe.

Graf 4: Vývoj RSI indikátoru



Zdroj: (5)(upravené)

Graf 4 znázorňuje RSI indikátor v jeho zjednodušenej podobe. Hodnoty sa pohybujú medzi +2.25 až -2.25. Na spodnom grafe mám cenovú krivku. Rozdiel ceny medzi prvým dňom 50.75 a desiatym

dňom 48.50 je -2.25 čo je minimálna hodnota indikátoru. Dôležité si je všimnúť hodnotu indikátoru medzi 10 dňom a 14 dňom, ktorá sa nemení a zostáva na hodnote -2.25 zatiaľ čo cena na spodnom obrázku klesá. Na 15. deň hodnota indikátoru rastie dvakrát rýchlejšie ako hodnota ceny, ktorá mení svoj smer po niekoľkodňovom klesaní. Práve preto, že sa mení smer ceny a indikátor meria výšku zmeny ako som ukázal pri momente, rastie jeho hodnota dvakrát rýchlejšie, čím naznačuje výraznú zmenu. Tento trend pokračuje až do 23. dňa, kde hodnota RSI prestáva rásť a pohybuje sa horizontálne zatiaľ čo cena rastie. Hodnota zmeny RSI spomaľuje, pretože sa cena znova pohybuje len jedným smerom za posledných 10 dní. Tento scenár vysvetľuje základné myšlienky indikátoru.(5)

Graf 5: Relative strength index rovnica

$$RSI = 100 - \left[\frac{100}{1 + RS} \right]$$

$$RS = \frac{\text{Average of 14 day's closes UP}}{\text{Average of 14 day's closes DOWN}}$$

Zdroj: (5)

Na grafe môžem vidieť základnú rovnicu pre výpočet RSI. Hlavná časť je výpočet RS, tzv. Relative strength ceny. K výpočtu potrebujem priemer z cien, ktoré skončili v pluse v predchádzajúcich 14 dní a tento následne vydelím priemerom z cien ktoré skončili v mínuse v predchádzajúcich 14 dní.

3. Analyzovaná stratégia

„Filozoficky čo sa obchodovania týka, žijem vo svete ktorý sa vracia k priemeru“. (6)

Čo tým chcel autor povedať? Akcie majú tendenciu sa po výraznejšom poklese ktrátkodobo vrátiť k priemeru z predchádzajúceho obdobia. To znamená, že po poklese cena zvykne krátkodobo narásť. Autor ďalej popisuje, že je výhodnejšie kupovať poklesy a predávať rast ako kupovať rast a predávať rast. Čo to presne znamená popíšem v nasledujúcich kapitolách. Samozrejme každú z myšlienok otestujem na historických dátach. Na identifikáciu obchodov použijem indikátory ktoré som popisoval v úvode a budem sa snažiť nájsť najvhodnejšie nastavenie jednotlivých parametrov tak, aby som dosahoval čo najlepšie výsledky.

3.1 Prístup Larryho Connorsa založený na RSI indikátore

Existuje viacero druhov stratégií, ktoré tento prístup používajú. Každá sa ale lísi rôznymi nastaveniami a modifikáciami, ktoré môžu znamenať veľké rozdiely vo výkonnosti. Ďalej v práci popíšem základné myšlienky Larryho stratégie. Na týchto myšlienkach budem stavať a analyzovať jednotlivé výsledky výkonnosti. Danú stratégiu aplikujem na kôš akcií napr. z indexu SPY, čo znamená, že budem simultánne vyhodnocovať stovky akciových titulov a vždy vyberiem ten akciový obchod, ktorý najlepšie splňa kritéria stratégie.

3.1.1 Identifikácia poklesu a nákup

Pullback znamená, že akcia určitú dobu rastie a po raste nastane spomalenie alebo pokles. Obchodníci sa zameriavajú a vyhľadávajú tieto pohyby, lebo akcie sa po takomto poklese zvyknú vrátiť aspoň na určitý čas, alebo pokračujú v dlhodobom trende. Tento prístup sa nazýva „návrat k priemeru“. K identifikovaniu tohto pohybu použijem RSI indikátor. Ako som popísal fungovanie momenta vyššie, budem vyhľadávať akcie s najväčším klesajúcim pohybom. Pomocou tohto indikátora budem generovať nákupné príkazy. (7)

3.1.2 Len akcie v dlhodobom rastúcom trende

Pre stratégiu je dôležité vyfiltrovanie obchodov, ktoré nie sú v dlhodobom rastúcom trende. Prečo je toto dôležité? Očakávam, že akcie s dlhodobým rastúcim trendom, po krátkodobom prepade znova začnú rásť. Rast nemusí byť výrazný, stačí keď sa akcia vráti len do polovice z toho čo

stratila. Čo sa ale stane, keď akcia výrazne stráca, RSI indikátor vygeneruje signál k nákupu a ja vstúpim do takéhoto obchodu? Očakávam, že nastane krátkodobý rast, aby som mohol z takéhoto obchodu profitovať. Samozrejme sa môže stať, že akcia bude ďalej pokračovať v klesaní a ja budem strácať. K prevencii tohto scenáru použijem indikátor priemeru s dlhodobým parametrom minimálne 200 dní. To znamená, že budem obchodovať len akcie, ktoré za posledných 200 dní rástli. Týmto si zvyšujem pravdepodobnosť, že ak akcia za posledných 200 dní rástla, môžem očakávať, že aj napriek tomu, že mala krátkodobý pokles sa znova vráti k svojmu dlhodobému trendu a bude pokračovať v raste. Jednoducho budem nakupovať len v smere dlhodobého rastúceho trendu. Mimochodom, obchodníci tento rast odborne nazývajú býčí trend. Pre úplnosť doplním, že dlhodobý klesajúci pohyb nazývajú medvedím trendom. (8)

3.1.3 Výstup z obchodu

Je dôležité si uvedomiť, že výstup z obchodov je rovnako dôležitý ako vstup do obchodu. To, kedy a ako z obchodu vystúpim, určuje to, či obchod skončí so ziskom alebo stratou. Pre výstup použijem krátkodobý indikátor priemeru. Keď cena po poklese začne rásť a uzavrie nad trendovou linkou indikátoru, vystupujem z obchodu. Hodnota parametra indikátoru bude mať vplyv na dĺžku obchodu a aj výšku zisku respektíve straty. Hodnotu tohto indikátoru budem testovať na základe výsledkov z testovania stratégie.

3.1.4 Kombinácia pravidiel

Kombináciou týchto indikátorov dostávam prístup, ktorý bude predmetom tejto analýzy. K výberu akcií použijem indexy SPY, SP100, QQQ, VTV. Pre stanovenie býčieho trendu použijem indikátor priemeru s vysokou hodnotou parametru. Na stanovenie výrazného krátkodobého poklesu použijem RSI indikátor a akciu, ktorá bude mať najmenšiu hodnotu tohto momentum indikátoru, nakúpim. Zostávam v obchode dovtedy, dokým cena akcia neuzavrie nad indikátorom priemeru s krátkodobou hodnotou parametru.

3.1.5 Modifikácie stratégie

Connors RSI stratégia sa obchoduje s množstvom modifikácií. Ide hlavne o líšiace sa parametre pre použité indikátory a mierne modifikácie tykajúce sa jednotlivých pravidiel. Pravidlá môžu vyzerať napríklad takto: Na nákup sa používa 200 dňový dlhodobý priemer kde cena musí byť nad týmto

priemerom a zároveň musí byť cena pod 10 dňovým priemerom a *RSI* indikátor sa nachádza pod hodnotou 10. Predajný príkaz je vygenerovaný keď následne cena stúpne nad hodnotu 5 dňového cenového priemeru. (9)

Ďalší z možných prístupov používa napríklad klasicky 200 dňový priemer, všetky akcie pod týmto priemerom ignoruje. Na nákup musí akcia padať posledné tri dni a zároveň musí byť hodnota *RSI* indikátoru v deň nákupu pod hodnotou 10. Na výstup z obchodu nie je použitý jednoduchý priemer, ale *RSI* indikátor a jeho hodnota musí byť nad 70. (10)

V nasledujúcim prístupe sa vôbec nepracuje s jednoduchými priemermi, ale len s *RSI* indikátorom. Akcia sa nakupuje keď sa cena nachádza pod hodnotou *RSI* indikátoru s hodnotou 35 a predáva, keď je cena nad hodnotou indikátoru 65. Test nieje vyhodnocovaný na jednotlivých akciách ale na indexe. (11)

Prístupov ako modifikovať túto stratégiu je naozaj veľa. Čo je možné z tohoto usúdiť je, že stratégia je robustná a funguje aj keď sa mierne zmenia jej parametre čo je určite pozitívna správa. Ja sa budem držať pôvodného prístupu a stratégiu budem stavať postupne. Zaujímavé je, že sa moc nehovorí o správe kapitálu a s akými veľkými pozíciami vstupovať do obchodov.

3.2 Dáta

Pre vyhodnotenie obchodnej stratégie a spočítanie hodnôt indikátorov potrebujem historické dáta pre dané typy akcií, ktoré budem rozdeľovať do košov podľa jednotlivých akciových indexov. Toto mi umožní otestovať stratégiu na dané akcie bez toho aby som musel vyberať kandidátov a vytvárať koše sám.

Tabuľka 1: Kôš akcií indexu SPY 500

Symbol	Popis
AAPL	APPLE INC
MSFT	MICROSOFT CORP
AMZN	AMAZON.COM INC
FB	FACEBOOK INC-CLASS A
TSLA	TESLA INC
GOOGL	ALPHABET INC-CL A
GOOG	ALPHABET INC-CL C
BRK B	BERKSHIRE HATHAWAY INC-CL B
JNJ	JOHNSON & JOHNSON
JPM	JPMORGAN CHASE & CO
V	VISA INC-CLASS A SHARES
PG	PROCTER & GAMBLE CO/THE
UNH	UNITEDHEALTH GROUP INC
DIS	WALT DISNEY CO/THE
NVDA	NVIDIA CORP
MA	MASTERCARD INC - A
HD	HOME DEPOT INC
PYPL	PAYPAL HOLDINGS INC
VZ	VERIZON COMMUNICATIONS INC
ADBE	ADOBE INC
CMCSA	COMCAST CORP-CLASS A
NFLX	NETFLIX INC
BAC	BANK OF AMERICA CORP
...
...

Zdroj: Vlastné spracovanie

Na tejto tabuľke je možné vidieť príklad takého koša, konkrétnie ide o index SPY. Z tohto koša sa budú vyberať jednotlivé akcie podľa toho, ktorá akcia na základe indikátorov najlepšie splňa podmienky obchodnej stratégie.

K historickému testovaniu budem používať takzvané upravené dátá. Upravené dátá sú kompenzované o udalosti, ktoré vyhlásila daná akciová spoločnosť. Tieto udalosti môžu byť napríklad výplata dividend alebo rozdelenie akcií. Výplatu dividend môže vyhlásiť spoločnosť v prípade, keď sa jej nadpriemerne darí a chce napríklad dodatočne odmeniť investorov, ktorí vlastnia určitý podiel v akciách. Pre investorov to je samozrejme pozitívna správa, pre samotnú spoločnosť to ale znamená, že sa hodnota akcií zníži o hodnotu vyplácaných dividend. Ďalšia z udalostí je rozdelenie akcií. Napríklad keď daná akcia sa stane veľmi drahou, spoločnosť sa môže rozhodnúť rozdeliť akcie a tak ju sprístupniť širšiemu publiku, čím si priláka väčší počet potenciálnych investorov. Zvýši počet cenných podielov čím si zníži svoju cenu. Povedzme že 10 podielov akcie má hodnotu 500 dolárov. Daná akcia zvýši svoje podiely na 20 čím zníži hodnotu týchto podielov o polovicu na 250 dolárov. V prepočte je celková hodnota danej akcie stále rovnaká,

čo sa ale stalo je, že umelo znížila svoju cenu zvýšením podielov a učinila ju tak dostupnejšiu širšiemu publiku. (12)

Efekt rozdelenia a prepočítania akcií realizovala dňa 31. 8. 2020 aj firma Apple. Všetko je možné vidieť v nasledujúcej tabuľke.

Tabuľka 2: Prepočítanie podielov akcií spoločnosti Apple

Name	Label	Close	Open	UOpen	UClose	Change	ChangePercent
Apple Inc.	Sep 1, 2020	134,18	132,76	132,76	134,18	5,14	0,0398
Apple Inc.	Aug 31, 2020	129,04	127,58	127,58	129,04	4,2325	0,0339
Apple Inc.	Aug 28, 2020	124,8075	126,0125	504,05	499,23	-0,2025	-0,0016
Apple Inc.	Aug 27, 2020	125,01	127,1425	508,57	500,04	-1,5125	-0,012
Apple Inc.	Aug 26, 2020	126,5225	126,1791	504,7165	506,09	1,6975	0,0136
Apple Inc.	Aug 25, 2020	124,825	124,6975	498,79	499,3	-1,0325	-0,0082
Apple Inc.	Aug 24, 2020	125,8575	128,6975	514,79	503,43	1,4875	0,012
Apple Inc.	Aug 21, 2020	124,37	119,2625	477,05	497,48	6,095	0,0515
Apple Inc.	Aug 20, 2020	118,275	115,75	463	473,1	2,5675	0,0222

Zdroj: Vlastné spracovanie

28. 8. 2020 cena akcií uzavrela v stĺpci *UClose* na hodnote 499,23 dolárov za jednotku akcie. V nasledujúci obchodný deň 31. 8. 2020 po otvorení trhov sa cena nachádzala na hodnote 127,58 za jednotku akcie. V prípade neuvedomenia si efektu prepočítania podielov akcií, by som si mohol mylne mysliť že nastal obrovský prepad v ich hodnote a ja by som tak stratil za jediný deň vyše približne 75 percent kapitálu, v prípade, že by som akcie vlastnil. Akcie však nasledujúci deň mierne zvýšili svoju hodnotu ako je vidieť v stĺpcach *Change* a *ChangePercent*. Ďalší, veľmi dôležitý fakt prečo používať upravené dátá pri testovaní obchodných systémov je, že v prípade ich nepoužitia, by v deň keď k prepočítaniu podielov došlo, generovali obchodné indikátory extrémne hodnoty, čo ale neodpovedá skutočnosti hodnoty ceny akcií. Takto by dochádzalo k chybným signálom a moje testovanie by neodpovedalo realite. V Stĺpcoch *Open* a *Close* sú upravené ceny akcie o tieto udalosti. Na obrázku je znázornené, že za dané obdobie firma nezaznamenala výraznejšie výkyvy. Pracovať s takto upravenými dátami je určite príjemnejšie. V systéme nemusím brat' do úvahy tieto udalosti a starat' sa kedy firma plánuje takéto prepočítanie alebo výplatu dividend. Toto by znamenalo niekoľko násobne zvýšenie komplexity spracovania dát.

Ďalšia, veľmi dôležitá skutočnosť, čo sa týka dát, je takzvaný *survivorship bias*. Dáta ktoré sú dnes dostupné obsahujú akcie, ktoré boli dostupné aj v minulosti. Čo ale nemusí úplne platiť keď sa pozeraeme z pohľadu súčasnosti. Ak zoberiem do úvahy akcie a získam dátu od roku 2007 tak mi musí byť jasné, že tieto akcie boli v danom roku obchodované a sú obchodované až k dnešnému dňu. Čo ale nemusí úplne platiť keby som chcel akcie ktoré sa v roku 2007 obchodovali a tým tak simulovať realitu z pohľadu minulosti. Niektoré akcie už mohli zbäkrotovať alebo sa rozdelili

alebo zlúčili a preto napríklad dnes daný kôš už tieto akcie nemusí obsahovať. (13) Toto môže mať negatívny vplyv na testovanie stratégií. Minimálne, je dobré si byť vedomý tohto *survivorship bias* efektu. Tým že používam koše akcií, ktoré patria medzi najvýznamnejšie firmy na svete znižujem tento dopad, každopádne môj test nebude úplne vyjadrovať realitu z pohľadu daného roka v minulosti.

3.3 Vyhodnocovanie

Pre vyhodnocovanie stratégie potrebujem základné ukazovatele na základe ktorých sa budem rozhodovať o úspešnosti jednotlivých prístupov. Je kritické vedieť rýchlo vyfiltrovať prístupy, ktoré jednoducho nefungujú. Pre takéto vyhodnocovanie je potrebných niekoľko základných ukazovateľov. (14)

- **Dĺžka testu a počet vygenerovaných obchodov**
- **Celkový čistý zisk**
- **Profit faktor**
- **Percentuálna ziskovosť**
- **Maximálny prepad**

Dĺžka testu a počet vygenerovaných obchodov hovorí o relevantnosti testovanej stratégie. Čo to znamená ? Čím dlhšiu históriu dát mám k dispozícii, tým viacej rôznych tržných situácií bude moja testovaná stratégia čeliť. Moja história dát siaha až do roku 2006. Čo je pre môj test veľmi dobré, keďže koncom roku 2007 nastala globálna finančná kríza a trhy čelili jedným z najviac turbulentných období v histórií. Bude zaujímavé sledovať čo sa so stratégiou dialo v danom období. Následne zažil trh niekoľko turbulentných období ako napríklad, Grécka dlhová kríza v roku 2012 alebo kríza korona vírusu v súčasnosti. Všetky tieto udalosti budú mať samozrejme veľký vplyv na testovanú stratégiu. Čím dlhší test stratégie, tým viacej scenárov na základe ktorých je možné vyhodnotiť úspešnosť. Počet vygenerovaných obchodov je samozrejme priamo úmerný s touto historiou dát.

Celkový čistý zisk reprezentuje základnú informáciu pre testovanú stratégiu a jej ziskovosť. Veľa obchodníkov ho používa ako hlavnú informáciu k vyhodnoteniu. Je potrebné si ale uvedomiť že tento ukazovateľ sám o sebe skoro nič nevyjadruje. Je potrebné ho zasadíť do kontextu ostatných ukazovateľov. Napríklad ako je veľmi jednoducho možné skresliť výsledok obchodnej stratégie. Použijem dátu pre test po globálne finančnej kríze v roku 2007-2008 a tento test spustím presne

potom ako sa trh prepadol. Samozrejme môže stratégia generovať veľa úspešných obchodov za toto obdobie a celkový čistý zisk môže byť veľmi vysoký vzhľadom na zainvestovaných kapitál. Kritické ale je čo by sa so ziskom dialo pred týmto obdobím a z dlhšieho časového horizontu. Preto je veľmi dôležité sledovať všetky ukazovatele a nezameriavať sa len na jeden. Výpočet je veľmi jednoduchý, od celkového zisku sa odpočíta celková strata, čím dostanem čistý zisk.

Profit faktor je definovaný ako celkový zisk vydelený celkovou stratou. Čo znamená že ak tento podiel bude väčší ako jedna tak pôjde o ziskový systém a ak tento podiel bude nižší ako jedna pôjde o stratový systém. Jednoduchý príklad kde mám štyri ziskové obchody (300, 250, 400, 600) a dva stratové (420, 380). Keď spočítame ziskové 1500 a stratové 800 vydelíme tento súčet čím sa dostaneme k číslu 1,88, čo znamená že moje ziskové obchody sú 1,88 krát väčšie ako moje stratové obchody a pre každý 1 investovaný dolár získam naspäť 1,88 dolára. Opačná situácia je keď tento podiel bude menší ako 1 napríklad 0,65. Kde sa každý investovaný dolár budem strácať skoro polovicu a dostanem naspäť len 65 centov.

Percentuálna ziskovosť je počet ziskových obchodov delený k počtu stratových obchodov. Pri výbere obchodnej stratégie sa odporúča ziskovosť väčšia ako 60 percent. Tiež je tento ukazovateľ chápaný ako percentuálna pravdepodobnosť, že nasledujúci obchod skončí so ziskom. Samozrejme chceme aby toto číslo bolo čo najväčšie, všetko závisí aj od nastaveného *money managementu* a rizikovosti daného systému. Napríklad systém ktorý bude mať nastavený *stop loss* bude mať samozrejme nižšiu percentuálnu ziskovosť ako systém bez *stop lossu*. Napríklad, keď vstúpim do obchodu a tento obchod sa bude následne pohybovať v smere na ktorom budem strácať systém so *stop lossom* napríklad 500 dolárov uzavrie obchod, keď sa táto hodnota prekročí. Systém bez *stop lossu* tento obchod neuzavrie a strata môže narásť niekoľko násobne vyššie. Ďalšou možnosťou je, že systém prekročí túto hodnotu ale následne sa vráti a dostane sa do plusových čísel a tým sa zvyšuje percentuálna ziskovosť obchodnej stratégie. Cena za toto zvýšenie je prijatie rizika potenciálne niekoľko násobnej vyššie straty.

Maximálny prepad znamená najhoršie obdobie pre ktoré bola obchodná stratégia v mínusových číslach. Je dobré sa využívať stratégiami, ktoré majú tento ukazovateľ vysoký. Čím nižšie toto číslo je, tým konzervatívnejšia je stratégia. Pri vyhodnocovaní si je dobré uvedomiť, aký vzťah má obchodník k rizikovosti. Napríklad, keď systém vykazoval za testované obdobie maximálny prepad 45 percent tak to pre mňa znamená že pri zainvestovaní 100 000 dolárov je možné že nastane obdobie kde budem strácať 45 percent čo je pokles na účte o celých 45 000, zostatok je tak 55 000 dolárov. K tomu aby som sa dostal na moju pôvodnú sumu 100 000 dolárov, ktoré som pôvodne zainvestoval potrebujem zhodnotiť zostatok 55 000 o celých 85 percent čo je

skoro o polovicu viac ako môj maximálny prepad. Je dobré si byť vedomý faktu, že v reálnom prostredí by mohol byť maximálny prepad ešte väčší ako vychádza z testu. Základné príslovie znie: „Minulé výsledky s určitosťou nehovoria, že aj budúce výsledky budú také isté.“. (15)

4. Automatický obchodný systém

K tomu aby bolo možné otestovať prístupy ktoré som popísal v predchádzajúcich kapitolách je nutné mať automatizované riešenie, ktoré dokáže simulovať obchodný prístup a aplikovať ho na historické dátu. Systém musí implementovať indikátory, ktoré obchodník používa k identifikovaniu cenových patternov a rovnako implementáciu pravidiel samotnej obchodnej stratégie. Zvláda jednoduchý reporting, na základe ktorého vie obchodník usúdiť kvalitu obchodnej stratégie. V nasledujúcich kapitolách sa pokúsim popísať a implementovať takýto systém.

4.1 Vybrané technológie

Pre vývoj systému som si zvolil platformu Microsoft a Framework .NET Core s programovacím jazykom C#. C# je jazyk ktorý sa dá rýchlo naučiť a je jednoduchý na čítanie. Je to silno typovaný jazyk, ktorý mi pomôže vyvarovať sa typových chýb už pri komplikovaní zdrojového kódu. Ponúka nástroje ako automatickú správu pamäte, jazykové integrované dotazy LINQ, generické typy a množstvo knižníc. C# spolu s frameworkom .Net Core, ktorý je open-source, rýchlo reaguje na požiadavky od komunity a beží na platformách Windows, Linux a Mac OS.

.NET Core je postavený na sade NuGet balíčkov, ktoré je možné podľa potreby pridávať a odoberať. Toto robí z tohto Frameworku vysoko modulárny systém a prispieva to aj k celkovej výkonnosti, keďže je možné obmedziť balíčky len na tie, ktoré skutočne potrebujem. Celková architektúra .NET Core aplikácií spolu so C# vysoko využíva princípy Dependency Injection. Tento Framework využíva vo svoj prospech tento prístup a rovnako ho budem využívať aj ja pri implementovaní mojej aplikácie. (16)

Pre uloženie dát som použil Microsoft SQL databázu spolu s Entity Framework Core. Entity Framework Core je nadstavba nad SQL databázou. Je to Object-relational mapping (ORM) Framework čo znamená že pomáha abstrahovať nad SQL databázou a pracovať s ňou ako s klasickými objektami v C#. Spolu s LINQ môžem písat klasické metódy, ktoré framework preloží do SQL dotazov a postará sa o nízko úrovňovú komunikáciu s Microsoft SQL databázou. Tento prístup mi pomôže urýchliť vývoj a zvýšiť produktivitu. (17)

Ako obsluhu aplikácie som zvolil technológiu Swagger ktorá ponúkne API, cez ktoré bude možné spustiť jednotlivé kroky potrebné k vyhodnoteniu stratégie. Je to veľmi užitočný nástroj, ktorý ponúka aj User Interface rozhranie, ktoré je automaticky generované na základe dátového modelu a API rozhrania aplikácie.

Ako reporting pre jednotlivé druhy implementácií obchodnej stratégie som zvažoval UI rozhranie, ktoré by využívalo niektorú z populárnych javascriptových knižníc. Nakoniec som sa rozhodol pre Excel spolu s knižnicou EPPlus. Excel je nástroj, ktorý je zaužívaným lídrom na trhu napríklad pre analýzu dát, vytvárania grafov, prezentovanie výsledkov. Rovnako dátu budú vyexportované v tomto Excel súbore, takže nebudem obmedzený len nato čo implementujem v aplikácii, ale viem veľmi rýchlo analyzovať dané dátu pomocou Excelu a jeho nástrojov. EPPlus patrí celkovo medzi najpopulárnejšie knižnice pre prácu s Excelom a podporuje .NET Core.

4.2 Použitý prístup pri návrhu systému

Pri návrhu systému som použil prístup založený na doménach, takzvaný Domain-Driven Design (DDD). Je to prístup k architektúre systému založeného na business doméne ktorú práve modeluje. Pomáha tak rozložiť komplexitu systému do jednotlivých častí, ktoré odzrkadlujú svoju implementáciu a návrhom aktuálne riešený problém a stanovujú hranice medzi jednotlivými časťami, čo pomáha pri spravovaní a kontrole zložitosti celého riešenia. (18)

Daný prístup má presné stanovené pravidlá, ako v takomto návrhu postupovať. Poskytuje patterny a princípy pri riešení náročných problémov. Pri implementácii dodržujem pravidlá takéhoto prístupu, čím dosahujem kvalitnejší dizajn, rozširiteľný systém a prístup pre riešenie komplexných problémov.

V tomto prístupe je uprednostňovaný iteratívny proces, čo znamená, že sa nesnaží dopredu zachytiť všetky architektonické aspekty, ktorých by som sa mal držať v celom procese vývoja. Skôr stavia na aktuálnych vedomostiah business domény a odporúča upravovať systém postupne, s tým ako sa tieto vedomosti prehľbujú a vychádzajú na povrch nové poznatky a skutočnosti. Nedáva do popredia použité technológie a technické riešenia, ale model domény a jej schopnosť riešiť základný problém, pre ktorý sa aplikácia vyvíja. (18)

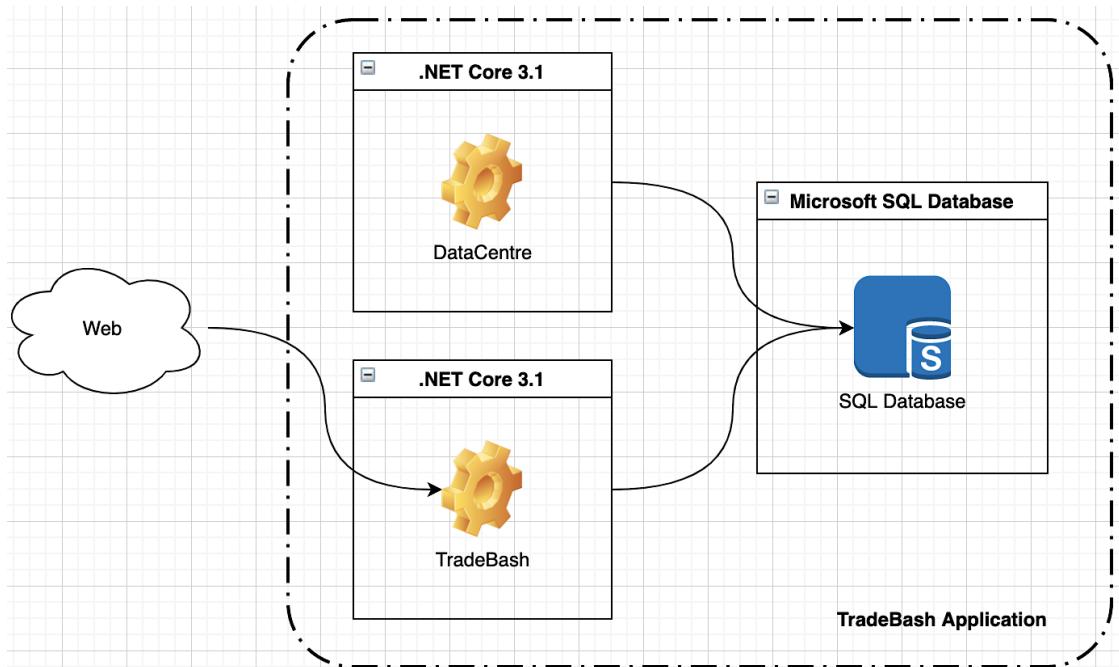
4.3 Návrh systému

Pri vytváraní projektu aplikácie som použil vzor, ktorý ponúka priamo Microsoft po nainštalovaní Nuget balíčku Clean Architecture. (19) Vybraný vzor obsahuje základnú štruktúru projektu so základmi pre vývoj DDD aplikácie. Od týchto základov som sa odrazil a ďalej rozširoval aplikáciu podľa potrieb môjho automatického obchodného systému. Jednotlivé časti systému popíšem v nasledujúcich kapitolách.

4.3.1 Fyzický pohľad

Architektúra fyzických vrstiev je veľmi jednoduchá. V tejto práci som nasadenie projektu na živé prostredie neriešil, ale projekt ako taký ja nato určite pripravený.

Obrázok 1: Fyzický pohľad na architektúru aplikácie



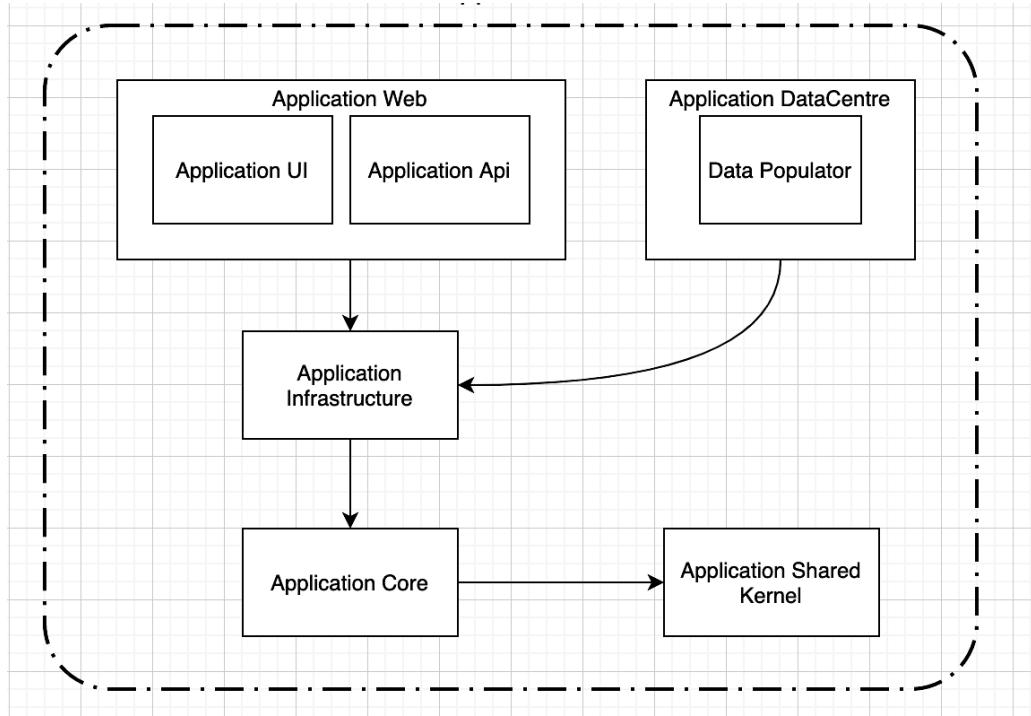
Zdroj: Vlastné spracovanie

Ako je možné vidieť na obrázku, hlavnými časťami aplikácie je TradeBash čo je samotná aplikácia a DataCentre ktoré sa stará o automatické získavanie a spracovanie dát. Tieto dve časti by boli v prípade nasadenia oddelené a používali by iné výpočtové zdroje. SQL databáza je relačná databáza kde budem ukladať dátá potrebné pre analýzu, výpočty, reporty a generované obchodné príkazy.

4.3.2 Vrstvy

Systém je rozdelený do niekoľkých vrstiev, ktoré majú presne definované hranice a každá ma pre systém iný význam. Základne rozdelenie znázorňujem na obrázku 2. Vrstvy si je možné predstaviť ako artefakty, ktoré logicky zodpovedajú za určitú časť systému. Toto mi pomôže rozložiť komplexitu a lepšie manažovať kód. Samotnému popisu jednotlivých vrstiev sa budem venovať v nasledujúcich kapitolách.

Obrázok 2: Vrstvy aplikácie



Zdroj: Vlastné spracovanie

4.3.3 Web

User interface rozhranie. Táto časť vrstvy slúži na zobrazovanie a možnosť interakcie so systémom. Je to tenká vrstva, čo znamená, že neobsahuje žiadnu business logiku. V user interface rozhraní je umiestnené takzvané potrubie, ktoré smeruje požiadavky do infraštruktúry. Následne, infraštruktúra priamo pracuje s objektami jadra, ktoré sa starajú o hlavnú business logiku. Možnosť zobrazenia Webového API pomocou Swaggru je takisto jednou z častí danej vrstvy. Webové API poskytuje výsledky testu obchodnej stratégie vo formáte JSON, ktoré je možné vyexportovať a ďalej s nimi pracovať. Rovnako sa tu nachádza API rozhranie na testovanie, využitie pre generovanie reportov pre obchodnú stratégiu.

4.3.4 Infraštruktúra

Infraštruktúra je zodpovedná za uloženie dát do databázy. Pracuje s objektami jadra, ktoré sa pri inicializácii nachádzajú v pamäti, a následne tieto objekty a ich reprezentáciu ukladá do samotnej databázy. V tejto vrstve neskôr implementujem Repository dizajn pattern, ktorý slúži na získanie, uloženie a aktualizáciu dát v aplikácii. Dôležité je, že vrstva infraštruktúry nijako nezasahuje do

vrstvy jadra, ale priamo závisí od vrstvy jadra, pretože ukladá objekty vznikajúce v jadre. Väčšina externých knižníc je integrovaná práve v tejto vrstve.

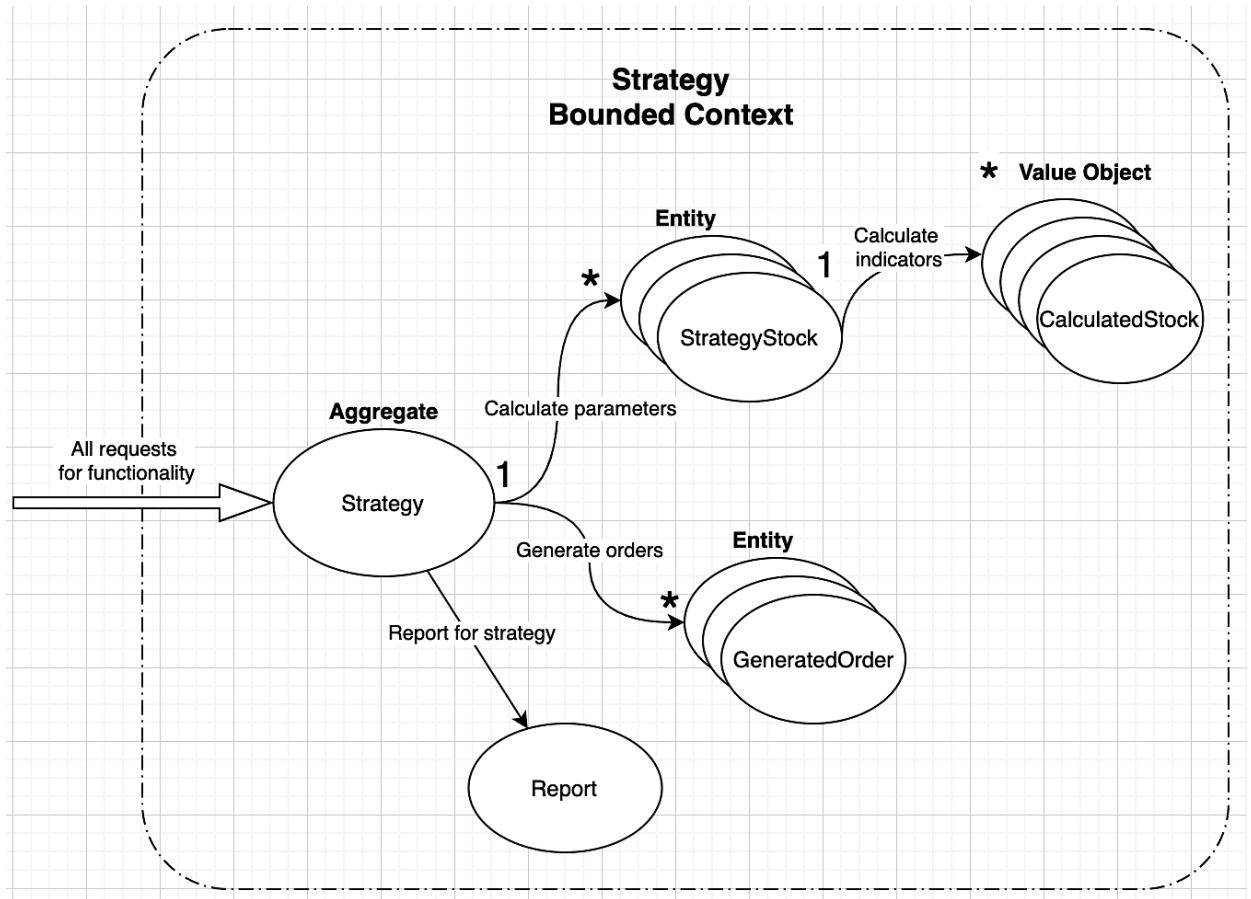
4.3.5 Zdieľané Jadro

Názov tejto vrstvy v podstate vysvetľuje jej význam. Obsahuje zdieľané základné triedy a rozhrania, ktoré používajú entity v jadre, ale aj vo vrstve infraštruktúry. Toto umožní redukovať množstvo kódu v systéme a tým zvýši je udržateľnosť a správu. Pod vrstvou si je jednoducho možné predstaviť znova použiteľný kód, ktorý sa využíva vo viacerých častiach aplikácie.

4.3.6 Jadro

Najdôležitejšia časť systému. V jadre je implementovaná hlavná business logika. Všetky pravidlá stratégie, algoritmizácia indikátorov, vyhodnocovanie výkonnosti budú implementované práve v tejto vrstve. Triedy v jadre sú takzvané Plain Old Class Objekty (POCO) čo v jednoduchosti znamená, že vrstva nie je závislá na žiadnom Frameworku ani externých knihovniach. Väčšina dotazov v aplikácií a samotné závislosti smerujú na túto vrstvu. Tým, že táto vrstva nemá žiadne závislosti a externé knižnice, je odolnejšia voči zmenám a samozrejme stabilnejšia. To ako ju používajú ostatné vrstvy ju nezaujíma, hlavné je, že vykonáva business logiku správne. Do budúcnosti je možné zmeniť celú infraštruktúru projektu alebo samotné web rozhranie bez toho aby sa musela vykonať zmena v jadre. Toto je obrovská výhoda čo sa týka samotného návrhu jadra. Pre pripomienku, DDD zdôrazňuje práve tento prístup a jeho základy sú postavené práve na takomto návrhu.

Obrázok 3: Návrh jadra



Zdroj: Vlastné spracovanie

Hlavný objekt je *Strategy* ktorý je na obrázku vyznačený ako Aggregate. Aggregate znamená, že je kontajnerom pre ostatné objekty a zaistuje integritu dát celej business domény. (20) Akékoľvek smerovanie požiadavky na entitu *GeneratedOrder* alebo *StrategyStock* je možné len cez hlavný *Strategy* objekt. Takýto prístup zaistuje správnosť, konzistenciu a celistvosť každej transakcie. Pomáha udržovať komplexitu tým, že definuje jedinú vstupnú bránu pre danú *Strategy* business doménu. Veľa systémov trpí problémom, že je možné sa dotazovať na objekty z každej strany, čo vytvára akúsi pomyselnú sieť závislosti a vzťahov, ktorá pri postupnom raste systému prináša obrovskú komplexitu a chaos.

Táto doména obsahuje dve entity *StrategyStock* a *GeneratedOrder*. *StrategyStock* slúži tejto doméne na uloženie hodnôt parametrov indikátorov a obsahuje implementáciu samotných indikátorov ako bude možné vidieť v praktickej časti. *StrategyStock* ďalej obsahuje kolekciu *CalculatedStock*, ktorá je na obrázku vyznačená ako ValueObject.

ValueObject je ďalší z princípov DDD návrhu. Vyjadruje informáciu ako celok a je nemenný, čo znamená, keď chceme aktualizovať daný ValueObject musíme vytvoriť novú inštanciu z

pôvodného objektu alebo zmazať pôvodný a vytvoriť nový na základe vstupných parametrov. *ValueObject* nemá životnosť a je len daných informácií v ňom uložených daných informácií. (21) V mojom prípade, sú tieto informácie napočítané hodnoty indikátorov z entity *StrategyStock* a základné informácie k hodnotám akcie ako otváracia cena, dátum, symbol atď.

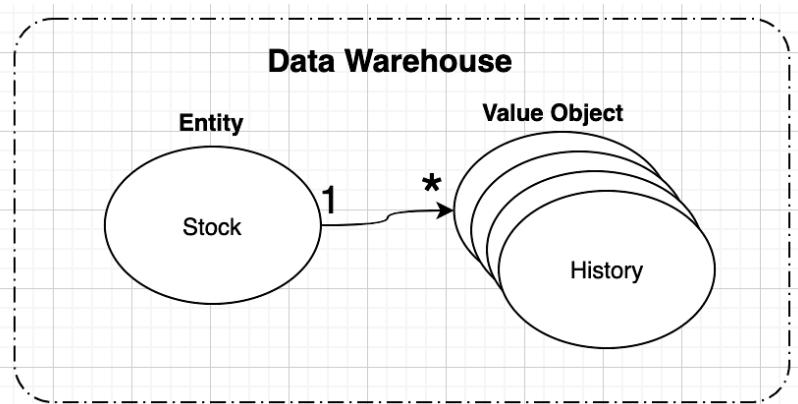
Ďalšou z entít je *GeneratedOrder*. Vyjadruje vygenerovaný príkaz, ktorý by sa poslal na burzu v prípade splnenia podmienok obchodnej stratégie. Tieto podmienky obchodnej stratégie definuje agregát *Strategy*, ktorý obsahuje kolekciu týchto *GeneratedOrder* objektov. Objekt sa vytvorí pri vygenerovaní príkazu a následne sa aktualizuje v prípade, že stratégia vygeneruje zatvárací príkaz.

Ked' sa pozriem na návrh domény ako celku, tak dáta z vrstvy infraštruktúry smerujú do objektu *Strategy*. Tento objekt zaistuje hlavne integritu celého obchodného systému. Implementácia a aplikácia indikátorov na dáta sa nachádza v ďalších objektoch, ktoré *Strategy* objekt spravuje. Po simulácii obchodnej stratégie je možné vygenerovať obchodné príkazy a report z týchto obchodných príkazov. Na jeho základe, budem vyhodnocovať jednotlivé prístupy.

4.3.7 Dátový sklad

Samostatná časť aplikácie je dátový sklad. Prvotný návrh aplikácie túto časť neobsahoval, narazil som ale na niekoľko problémov, ktoré ma privideli k implementácii automatického mechanizmu na spracovanie dát, slúžiacich na realizáciu testov a vypočítavanie hodnôt pre stratégiu a indikátory. Problémy, na ktoré som pri vývoji narazil následne popíšem v praktickej časti.

Obrázok 4: Návrh dátového skladu



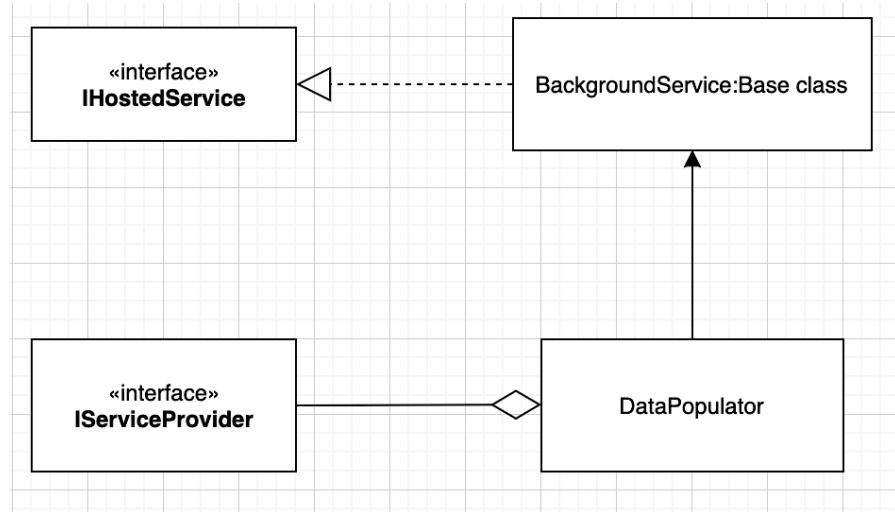
Zdroj: Vlastné spracovanie

Ako znázorňuje obrázok, návrh je veľmi jednoduchý ide prakticky o jednu entitu *Stock* a ValueObject *History*. Derivát akcie reprezentuje objekt *Stock* a pre každý takýto objekt, ukladám jeho historiu do kolekcie *History*. V tomto objekte sa nachádzajú detailné informácie o danej akcie

ako otváracia cena, zatváracia cena, dátumy, maximálne dosiahnutá cena alebo minimálne a ďalšie.

Samotný automatický mechanizmus využíva implementáciu rozhrania *IHostedService* a triedy *BackgroundService*, ktorú poskytuje Microsoft spolu s Frameworkom .NET Core. Implementácia sa automaticky v definovaných intervaloch na pozadí spúšťa a vykonáva určitú činnosť.

Obrázok 5: Návrh automatického mechanizmu dátového skladu



Zdroj: Vlastné spracovanie

Mnou definovaná trieda *DataPopulator* obsahuje základnú logiku spracovania dát. Táto trieda dedí zo spomenutej *BackgroundService* triedy čiže je určená pre tento automatický samospúšťiaci mechanizmus. Ďalej využíva triedu *IServiceProvider*, pomocou ktorej získava implementácie funkčností potrebnú k tomu aby mohla spracovať dátá. Detaily popíšem v praktickej časti pri samotnej implementácii.

5. Praktická časť

V praktickej časti implementujem popísaný systém a popíšem jednotlivé problémy, s ktorými som sa pri vývoji stretol. K otestovaniu systému potrebujem dátá, na ktoré budem aplikovať obchodné myšlienky. Najskôr potrebujem zaobstarať implementáciu spracovania dát. Popíšem implementáciu indikátorov a otestujem obchodnú stratégiu a porovnám jej výkonnosť pri rôznych nastaveniach vstupných parametroch.

5.1 Vývojárske nástroje a knižnice

Ešte predtým ako sa pustím do samotného vývoja aplikácie, predstavím technológie, s ktorými som aplikáciu vyvíjal. V prípade samotného vývoja .NET Core aplikácie som použil vývojové IDE od spoločnosti Jetbrains Rider. Rider funguje na operačných systémoch Linux, Windows aj Mac OS a má skvelú podporu .NET aplikácií s bohatou ponukou funkčností. Tým, že som časť aplikácie vyvíjal aj na operačnom systéme Mac OS bol Rider ideálna voľba. V samotnej aplikácii som použil niekoľko NuGet balíčkov, ktorých prehľad a stručný popis sa nachádza v nasledujúcej tabuľke.

Tabuľka 3: Použité NuGet balíčky

Názov NuGet balíčku	Verzia	Popis
EPPlus	5.6.3	Práca s Excel súbormi
EntityFrameworkCore	3.1.3	ORM nadstavba nad SQL databázou
Newtonsoft.Json	12.0.3	Práca s dátami vo formáte Json
Swashbuckle.AspNetCore	5.3.3	Dokumentácia API

Zdroj: Vlastné spracovanie

5.2 Spracovanie dát

Spracovanie dát bude prebiehať v niekoľkých krokoch. Najskôr potrebujem zaobstarať historické dátá minimálne o dĺžke 10 rokov. Čím dlhšia história tým relevantnejšie budú výsledky z testovania obchodnej stratégie. Ako zdroj dát som použil cloud službu <https://iexcloud.io/>. Narazil som ale na problém, že od určitého počtu objemu získaných dát je táto služba spoplatnená. Ja ale nepotrebujem všetky zdroje dát naraz. Databázu môžem plniť postupne aby som neprekročil povolený limit objemu dát. Zároveň, chcem aby sa mi dátá každým dňom aktualizovali a mohol by som tak sledovať vývoj stratégií vždy k aktuálnemu dátumu. Na takéto postupné plnenie potrebujem samozrejme mechanizmus, ktorý moje požiadavky zabezpečí. Zároveň, sa daný mechanizmus

postará o celé spracovanie tak, aby som mohol s dátami akokoľvek pracovať, bez závislosti na tejto cloudovej službe. To znamená, že dáta z tejto služby si uložím do vlastnej databázy a pri následnom testovaní obchodných stratégii nebudem žiadať historické dáta z externej služby, ale z mojej naplnenej databázy. Takýmto postupom zabezpečím splnenie všetkých vyššie uvedených požiadaviek.

Je potrebné si uvedomiť, že v mojom prípade pôjde skutočne o veľké množstvo potrebných dát. Pre lepšiu predstavu, história dát na ktorých budem realizovať testy je 1 903 200 dní. Ked' ďalej zoberiem do úvahy, že nad týmto objemom dát je potrebné počítať indikátory a generovať obchodné príkazy, a to pre viacero prístupov s odlišnými parametrami, náročnosť a vyťaženie dát je obrovské.

Pre vyššie uvedené dôvody som sa rozhodol implementovať mechanizmus, ktorý sa sám dokáže v pravidelných intervaloch spustiť a aktualizovať vlastnú históriu dát. Ďalej vie identifikovať pre aké akcie a k akému dátumu obsahuje tieto historické dáta a dokáže sám určiť akú dlhú históriu potrebuje k dnešnému dátumu. Na základe dĺžky potrebnej histórie zostaví API požiadavok pre danú akciu, pošle ju do vyššie uvedenej cloudovej služby <https://iexcloud.io/> a výsledkom si aktualizuje vlastnú históriu dát. Všetko prebieha plne automaticky.

5.2.1 Integrácia Entity Frameworku

Microsoft SQL databázu a Entity framework som zvolil ako sklad pre moje dáta. Pre prácu s týmto Frameworkom je ako prvé potrebné definovať *DbContext*. Definícia *DbContextu* sa nachádza v projekte *TradeBash.Infrastructure* ktorý je zodpovedný za celú infraštruktúru aplikácie. *DbContext* je kombinácia *UnitOfWork* a *Repository* patternov. *DbContext* je v podstate inštancia ktorá reprezentuje takzvanú session s databázou, ktorú použijem na dotazovanie a ukladanie mojich entít do databázy pomocou LINQ dotazov.

Obrázok 6: Implementácia DbContextu pre prácu s Entity Frameworkom

```
public class AppDbContext : DbContext
{
    public AppDbContext(DbContextOptions<AppDbContext> options)
        : base(options)
    {

    }

    public DbSet<Strategy> Strategies { get; set; }

    public DbSet<StrategyStock> StrategyStocks { get; set; }

    public DbSet<Stock> Stocks { get; set; }

    public DbSet<StockHistory> StockHistories { get; set; }

    public DbSet<CalculatedStock> CalculatedStocksHistory { get; set; }

    public DbSet<GeneratedOrder> GeneratedOrders { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder){...}

    public override async Task<int> SaveChangesAsync(
        CancellationToken cancellationToken = new CancellationToken()){...}

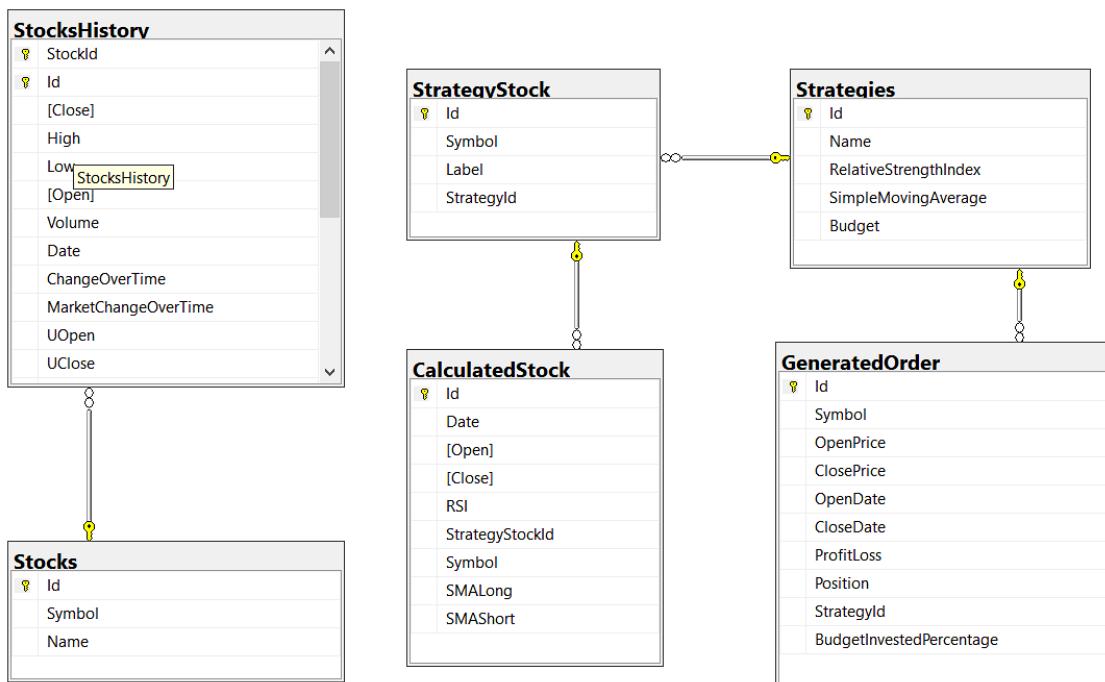
    public override int SaveChanges(){...}
}
```

Zdroj: Aplikácia TradeBash (vlastné spracovanie)

Na obrázku je znázornená implementácia, kde dedím z *DbContextu* a definujem moje entity ako vlastnosti danej triedy. Každá entita reprezentuje tabuľku v Microsoft SQL databáze. Implementácia triedy obsahuje niekoľko metód. *OnModelCreating* je preťažená metóda z *DbContextu*, kde je možné konfigurovať rôzne vlastnosti modelu. V mojom prípade ide o konfiguráciu migrácií, kde sa skenuje aktuálna zostava, ktorá musí obsahovať zložku migrácií aplikujúcich sa pri vytvorení alebo aktualizácii databázy. Ďalšie metóda je *SaveChangesAsync*, ktorá po zavolaní, ako názov napovedá, uloží aktuálny objekt v pamäti do databázy.

Na obrázku 7 je znázornených niekoľko základných entít. *Stocks* a *StocksHistory* entity reprezentujú implementáciu dátového skladu. Pre obchodné stratégie je hlavnou entitou *Strategy*. Táto entita obsahuje *StrategyStock*, čo je kontajner pre napočítane hodnoty indikátorov pre jednotlivé typy akcií, na obrázku entita *CalculatedStock*. A každá Stratégia obsahuje *GeneratedOrder* vygenerované príkazy, ktoré realizovala stratégia potom, ako sa spustil backtest.

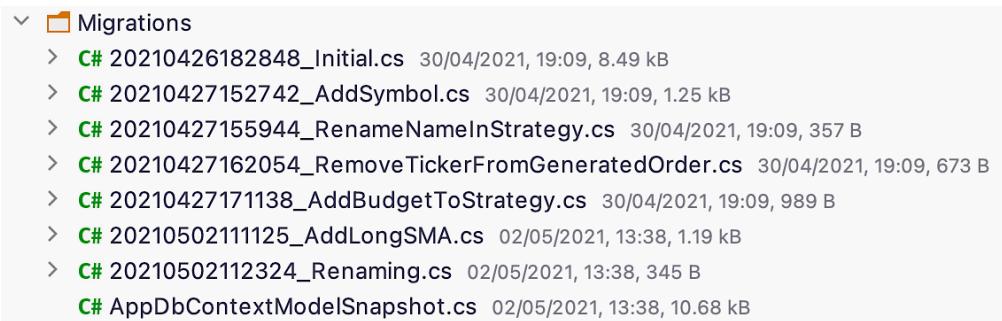
Obrázok 7: Databázový diagram



Zdroj: Aplikácia Microsoft SQL Server Management Studio (vlastné spracovanie)

Zmeny v dátovom modeli sa generujú do databázy pomocou spomínaných migrácií. Migrácie mi pomôžu inkrementálne aktualizovať stav mojej databázy, bez toho aby som stratil existujúce dátá. O všetko sa stará Entity Framework, ktorý tento mechanizmus ponúka ako súčasť balíčku. Ja len generujem migrácie po úprave entity reprezentujúcej model, pomocou terminálu a definovaných príkazov. Zoznam vygenerovaných migrácií v priebehu vývoja aplikácie je možné vidieť na obrázku 8. *Initial* reprezentuje začiatočný stav databázy. Postupne, ako som model upravoval a doplňoval funkcionality, bolo potrebné vygenerovať nové migrácie. Funkcionalita dobré slúži aj ako dokumentácia úpravy databázy. Čo je ešte dôležité je fakt, že je možné sa v prípade chybnej migrácie vrátiť a aplikovať stav databázy pred chybnou migráciou.

Obrázok 8: Vygenerované migrácie na úpravu databázového modelu



Zdroj: Aplikácia TradeBash (vlastné spracovanie)

Ako som hovoril jednotlivé entity reprezentujú dané tabuľky v databáze. O prácu s entitami v aplikácií sa stará implementácia Repository vzoru. V jednoduchosti ide o abstrakciu nad entity frameworkom ktorá implementuje rozhrania, ktoré definujú akým spôsobom môžem pracovať s danou databázou a danými entitami.

Obrázok 9 zobrazuje vytvorené generické rozhranie na základnú prácu s entitami. Každá entita musí dediť z *BaseEntity* v prípade že chce použiť túto implementáciu. *BaseEntity* je použitá ako generický parameter pre toto rozhranie.

Obrázok 9: Repository rozhranie pre základnú prácu s entitami v databáze

```
public interface IRepository<T>
{
    Task<T> GetByIdAsync<T>(int id) where T : BaseEntity;
    Task<List<T>> ListAsync<T>() where T : BaseEntity;
    Task<T> AddAsync<T>(T entity) where T : BaseEntity;
    Task UpdateAsync<T>(T entity) where T : BaseEntity;
    Task DeleteAsync<T>(T entity) where T : BaseEntity;
}
```

Zdroj: Aplikácia TradeBash (vlastné spracovanie)

EfRepository trieda implementuje toto rozhranie. Na obrázku sú znázornené jednotlivé metódy a ich implementácia. Takto jednoducho je možné použiť túto implementáciu pre akúkoľvek entitu definovanú v *DbContext* a zdediť jej funkčnosť.

Obrázok 10: Implementácia repository rozhrania

```
public class EfRepository : IRepository
{
    protected readonly AppDbContext _dbContext;

    public EfRepository(AppDbContext dbContext){...}

    public T GetById<T>(int id) where T : BaseEntity{...}

    public Task<T> GetByIdAsync<T>(int id) where T : BaseEntity{...}

    public Task<List<T>> ListAsync<T>() where T : BaseEntity{...}

    public async Task<T> AddAsync<T>(T entity) where T : BaseEntity
    {
        await _dbContext.Set<T>().AddAsync(entity);
        await _dbContext.SaveChangesAsync();
        return entity;
    }
}
```

Zdroj: Aplikácia TradeBash (vlastné spracovanie)

V aplikácii som implementoval aj ďalšie typy repozitárov ako *StockRepository* a *StrategyRepository*. Hlavný rozdiel je v tom, že napríklad pre *Strategy* entitu potrebujem získať aj všetky objekty definované v tejto triede. Generický repozitár neumožňuje získanie týchto závislostí. Kľúčovým slovom, ktoré umožňuje získať tieto závislosti medzi entitami je takzvaný *Include*, ktorého použitie je znázornené nasledujúcim obrázkom.

Obrázok 11: Metoda na získanie všetkých stratégií aj s ich závislosťami

```
public Task<List<Strategy>> GetAllAsync()
{
    return _dbContext.Set<Strategy>()
        .Include(x => x.StocksHistory)
        .ThenInclude(xx => xx.OrderedStocksHistory)
        .Include(x => x.GeneratedOrders)
        .ToListAsync();
}
```

Zdroj: Aplikácia TradeBash (vlastné spracovanie)

Metóda implementovaná v *Strategy* repozitári na získanie všetkých existujúcich stratégií a aj všetkých závislostí ktoré sú naviazané na túto entitu. Len doplním že tento repozitár dedí zo základného *IRepository* rozhrania a ďalej implementuje *IStrategy* rozhranie, ktoré definuje predpis pre metódy, ku ktorým potrebujem získať závislosti danej entity.

5.2.2 Implementácia dátového skladu

Pre automaticky mechanizmus, ktorý v pravidelných intervaloch spustí implementáciu dátového skladu a aktualizuje potrebné dáta som použil Microsoft *IHostedService* a implementoval triedu *DataPopulator* ako som popísal v teoretickej časti návrhu architektúry aplikácie. V nasledujúcich častiach popíšem jednotlivé triedy, ktoré sú zodpovedné za danú funkčnosť.

IHostedService hlavnou myšlienkou tohto rozhrania je, že zaregistrujem jeho implementáciu pomocou Dependency Injection kontajneru, ktorý potom spustí alebo zastaví spolu s aplikáciou kde je rozhranie definované, beh mojej implementácie dátového skladu reprezentovaného triedou *DataPopulator* znázornenej na obrázku 12.

Obrázok 12: Registrácia DataPopulator triedy do Dependency Injection kontajneru

```
public static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
        .ConfigureWebHostDefaults(webBuilder =>
    {
        webBuilder.UseStartup<Startup>();
    })
    .ConfigureServices(services =>
    {
        services.AddHostedService<DataPopulator>();
    });
}
```

Zdroj: Aplikácia TradeBash (vlastné spracovanie)

BackgroundService implementuje dve klíčové metódy *ExecuteAsync* a *StopAsync*. Implementácia *ExecuteAsync* musí vrátiť *Task*, ktorý reprezentuje životnosť opakujúceho procesu vykonávanom v definovanom intervale. Na obrázku 13 je znázornená implementácia, ktorá pravidelne spúšťa *DataPopulatorEngine* v intervale 1 deň, až pokiaľ sa nenastaví zatvárací token.

Obrázok 13: Implementácia metódy ExecuteAsync

```
protected override async Task ExecuteAsync(CancellationToken stoppingToken)
{
    _logger.LogInformation("Hosted Service is working");

    while (!stoppingToken.IsCancellationRequested)
    {
        await DataPopulatorEngine();

        await Task.Delay(TimeSpan.FromDays(1), stoppingToken);
    }
}
```

Zdroj: Aplikácia TradeBash (vlastné spracovanie)

StopAsync sa spustí, keď zastavím aplikáciu, čo znamená, že trieda *WebHost* iniciuje zastavenie aplikácie čo vydá signál pre spustenie tejto metódy. To znamená že sa nastaví zatvárací token čím sa preruší opakujúci proces implementácie mojej *DataPopulator* triedy.

Obrázok 14: Implementácia metódy StopAsync

```
public override async Task StopAsync(CancellationToken stoppingToken)
{
    _logger.LogInformation("Hosted Service is stopping");

    await base.StopAsync(stoppingToken);
}
```

Zdroj: Aplikácia TradeBash (vlastné spracovanie)

IServiceProvider poskytuje implementácie servis z Dependency Injection kontajneru, ktoré potrebujem pre získanie dát, serializáciu a následné uloženie serializovaných dát do SQL databázy.

Obrázok 15: Poskytnutie servis z Dependency Injection kontajneru triedou ServiceProvider

```
_stocksCsvReader = scope.ServiceProvider.GetRequiredService<IStocksCsvReader>();
_stockRepository = scope.ServiceProvider.GetRequiredService<IStockRepository>();
_dataProvider = scope.ServiceProvider.GetRequiredService<IDataProvider>();
```

Zdroj: Aplikácia TradeBash (vlastné spracovanie)

DataPopulator v triede je implementovaná hlavná logika dátového skladu. Implementáciu prejdem len z pohľadu verejného rozhrania, ktoré jednotlivé triedy poskytujú ako servisu k tomu, aby som mohol vykonať všetky potrebné kroky. Implementácia začína zavolaním *CreateScope* metódy, ktorá mi poskytne implementácie všetkých definovaných rozhraní a ich funkčností potrebných na prácu v metóde. Najskôr potrebujem načítať zoznam akcií pre ktoré budem získavať dátu

definované v .csv súboroch. Pre každý index mám definovaný jeden .csv súbor. Následne použijem metódu *LoadFile*, kde ako parameter predám názov daného indexu. Takto načítam všetky akcie pre ktoré chcem aktualizovať históriu.

Obrázok 16: Implementácia metódy DataPopulator

```
private async Task DataPopulatorEngine()
{
    using (var scope = _services.CreateScope())
    {
        _stocksCsvReader = scope.ServiceProvider.GetRequiredService<IStocksCsvReader>();
        _stockRepository = scope.ServiceProvider.GetRequiredService<IStockRepository>();
        _dataProvider = scope.ServiceProvider.GetRequiredService<IDataProvider>();

        _stocksCsvReader.LoadFile(IndexVersion.Spy100);
        _stocksCsvReader.LoadFile(IndexVersion.Spy500);
        _stocksCsvReader.LoadFile(IndexVersion.QQQ);
        _stocksCsvReader.LoadFile(IndexVersion.VTV);

        var stocksToUpdate = _stocksCsvReader.GetAllToUpdate();
        foreach (var (symbol, name) in stocksToUpdate)
        {
            _logger.LogInformation($"Populating data for {symbol}");

            var existingStock = await _stockRepository.GetBySymbolAsync(symbol);
            if (existingStock == null)
            {
                var stocks = await _dataProvider
                    .GetSerializedStocksFromDataProviderAsync(symbol, HistoryRange.Max);
                await AddHistoryToDb(symbol, name, stocks);
            }
            else
            {
                var lastDateDifference = GetDateDifferenceFromStockLastDate(existingStock);
                var historyRange = _dataProvider.GetRangeForHistoricalData(lastDateDifference);
                var stocks = await _dataProvider
                    .GetSerializedStocksFromDataProviderAsync(symbol, historyRange);
                RemoveExistingHistory(existingStock, stocks);
                await AddHistoryToDb(existingStock, stocks);
            }

            _logger.LogInformation($"Populating succesfull for {symbol}");
        }
    }
}
```

Zdroj: Aplikácia TradeBash (vlastné spracovanie)

Týmto zoznamom akcií budem prechádzať. Pre každú potrebujem poslať dotaz na databázu a zistiť, či už v mojom dátovom sklage existuje. Ak sa v sklage ešte nenachádza, znamená to, že chcem od IEX providera maximálnu dostupnú históriu dát. Do parametru metódy *GetSerializedStocksFromDataProviderAsync* poskytujem symbol, ktorý reprezentuje danú akciu a históriu, ktorá je v tomto prípade nastavená na maximálnu hodnotu. Potom ako z IEX providera dostanem odpoveď v podobe JSON, potrebujem realizovať serializáciu aby danému formátu rozumela moja SQL databáza. Na obrázku je možné vidieť dátu vo formáte JSON pre spoločnosť Apple od IEX providera.

Obrázok 17: Neserializované dátá spoločnosti Apple od providera IEXCloud

```
"close": 134.72,  
"high": 135.06,  
"low": 133.56,  
"open": 134.83,  
"symbol": "AAPL",  
"volume": 66905069,  
"id": "HISTORICAL_PRICES",  
"key": "AAPL",  
"subkey": "",  
"date": "2021-04-26",  
"updated": 1619485201000,  
"changeOverTime": 0,  
"marketChangeOverTime": 0,  
"uOpen": 134.83,  
"uClose": 134.72,  
"uHigh": 135.06,  
"uLow": 133.56,  
"uVolume": 66905069,  
"fOpen": 134.83,  
"fClose": 134.72,  
"fHigh": 135.06,  
"fLow": 133.56,  
"fVolume": 66905069,  
"label": "Apr 26, 21",  
"change": 0,  
"changePercent": 0
```

Zdroj: IEXCloud (vlastné spracovanie)

Pri serializácií vynechávam len *subkey*, ktorý je pre moje stratégie nepodstatný. Po následnom spracovaní a uložení dostávam dátu v relačnej podobe, znázornené obrázkom 18.

Obrázok 18: Serializované dátá spoločnosti Apple od IEX providera

	StockId	Name	Symbol	Close	High	Low	Open	ChangeOverTime	MarketChangeOverTime	Label	Change	ChangePercent
4	1	Apple Inc.	AAPL	134.84	135.47	133.34	133.51	0.112449467865688	0.112449467865688	Apr 19, 21	0.680000000000007	0.0051
5	1	Apple Inc.	AAPL	134.16	134.67	133.28	134.3	0.10683936968897	0.10683936968897	Apr 16, 21	-0.340000000000003	-0.0025
6	1	Apple Inc.	AAPL	134.5	135	133.64	133.82	0.109644418777329	0.109644418777329	Apr 15, 21	2.47	0.0187
7	1	Apple Inc.	AAPL	132.03	135	131.655	134.94	0.089266562164838	0.089266562164838	Apr 14, 21	-2.400000000000001	-0.0179
8	1	Apple Inc.	AAPL	134.43	134.66	131.93	132.44	0.109066908670902	0.109066908670902	Apr 13, 21	3.19	0.0243
9	1	Apple Inc.	AAPL	131.24	132.85	130.63	132.52	0.082748948106592	0.082748948106592	Apr 12, 21	-1.755	-0.0132
10	1	Apple Inc.	AAPL	132.995	133.04	129.47	129.8	0.0972279514891512	0.0972279514891512	Apr 9, 21	2.634999999999999	0.0202
11	1	Apple Inc.	AAPL	130.36	130.39	128.52	128.95	0.0754888210543686	0.0754888210543686	Apr 8, 21	2.460000000000001	0.0192

Zdroj: Aplikácia Microsoft SQL Management Studio (vlastné spracovanie)

Druhá časť bloku implementácie triedy *DataPopulator* počíta stým, že daná akcia v databáze existuje. Čo potrebuje vedieť je, akú dlhú história akcia obsahuje, a v prípade, že história nie je aktualizovaná, sa spočíta časový rozdiel k dnešnému dňu. Na základe tohto rozdielu sa zloží API príkaz pre IEX s daným časovým rozsahom, ktorý reprezentuje obdobie potrebné k aktualizácii. Následne prebehne vyššie popísaný postup a dátá sa na základe dátumu od poslednej uloženej hodnoty aktualizujú k dnešnému dňu.

Nad takto pripravenými dátami môžem začať počítať hodnoty indikátorov a testovať obchodné stratégie bez toho aby som musel zakaždým použiť externého dátového poskytovateľa, ktorý by si od určitého objemu dát začal účtovať poplatky. Implementácia skladu beží nepretržite. V časových intervaloch sa zostavujú API požiadavky pre IEX providera, spočíta sa časové obdobie potrebné k aktualizácii danej akcie a následne sa na základe tohto časového obdobia aktualizujú dátá v dátovom sklade. Moja obsluha tejto funkčnosti nie je potrebná, všetko prebieha automaticky. Každý deň sa mi takto aktualizujú dátá pre viac ako 600 typov rôznych akcií.

5.3 Príprava dát pre Backtest

Jedným z prvých krov po implementácií dátového skladu je príprava dát pre backtest, ktorý bude generovať obchodné signály. Hlavná časť tejto prípravy dát je implementácia indikátorov, ktoré sú hlavnými časťami na generovanie obchodných signálov. Moja testovaná stratégia využíva dva hlavné indikátory ako som popisoval v teoretickej časti. Jedným z nich je SMA a druhý RSI, tak ako ho pôvodne používal Wilder. Hodnoty indikátorov potrebujem spočítať pre každú akciu zvlášť, čo znamená že potrebujem prejsť celú história a pre každý deň realizovať tieto výpočty.

5.3.1 Implementácia indikátorov

Počítanie hodnôt indikátorov spadá pod danú stratégiu. Stratégia môže mať rôzne parametre pre indikátory. Na základe týchto parametrov sa budú výsledky jednotlivých stratégií lísiť. Hlavná entita

v systéme, cez ktorú prechádzajú všetky informácie a cez ktorú systém ovláda celú funkciu je spomínaná entita *Strategy*. Táto entita je brána ktorá ma celú manipuláciu pod kontrolou a nič sa v systéme nerealizuje bez toho, aby to táto entita nemala pod kontrolou. Výnimkou nie je ani počítanie hodnôt indikátorov. Metoda *RunCalculationForStock*, ktorá ako parameter príma dátu akcie a volá sa v entite *Strategy*. Táto následne prechádza dostupnú históriu a vola metódu *CalculateForStock*, zobrazenu nasledujúcim obrázkom.

Obrázok 19: Implementácia metódy pre počítanie hodnôt indikátorov

```
public void CalculateForStock(
    string symbol,
    DateTime date,
    double open,
    double close,
    double low)
{
    var stock = CalculatedStock.From(symbol, date, open, close, low);
    CalculatedStocksHistory.Add(stock);

    var smaShort = CalculateSMA(_smaShortParameter);
    var smaLong = CalculateSMA(_smaLongParameter);
    var rsi = CalculateRelativeStrengthIndex();

    stock.SetIndicators(smaShort, smaLong, rsi);
}
```

Zdroj: Aplikácia TradeBash (vlastné spracovanie)

V tele *CalculateForStock* sa volajú *CalculateSMA* metódy, ktoré počítajú hodnoty SMA indikátorov a *CalculateRelativeStrengthIndex* metóda, ktorá počíta hodnoty RSI indikátoru. Výsledné hodnoty z týchto metód sa následne nastavia pre danú akciu metódou *SetIndicators*. Takto spočítané hodnoty sa uložia do databázy, aby som s nimi mohol neskôr pracovať pri testovaní pravidiel a vyhodnocovaní výkonnosti obchodnej stratégie.

Implementácia SMA indikátoru je veľmi jednoduchá. Jedná sa o klasický priemer vypočítaný za posledných n dní. Na obrázku 20 je vidieť že na základe parametru z kolekcie vyberie potrebný počet dní, dátu sa materializujú *Select* príkazom ktorý vyberie zatváraciu cenu a zavolá sa metóda *Average* ktorá spočíta priemer. Dôležité je ak kolekcia neobsahuje potrebnú dĺžku histórie, indikátor sa nepočíta a do databázy sa uloží nulová hodnota.

Obrázok 20: Implementácia SMA indikátoru

```
private double? CalculateSMA(int? smaParameter)
{
    if (!smaParameter.HasValue) return null;

    if (CalculatedOrderedStocksHistory.Count < smaParameter) return null;

    var average = CalculatedOrderedStocksHistory
        .TakeLast(smaParameter.Value)
        .Select(x => x.Close)
        .Average();

    return average;
}
```

Zdroj: Aplikácia TradeBash (vlastné spracovanie)

Pri načítavaní dát z databázy má entita v kolekcií *IReadOnlyCollection<CalculatedStock>* uložené dátá, ktoré ale nie sú zoradené podľa dátumu. Preto, po získaní dát z databázy pri materializovaní tejto entity do pamäte mám pomocnú premennú *CalculatedOrderedStocksHistory*, ktorá vracia zoradené dátá podľa dňa v histórii. Takto si môžem byť istý, že pri nasledujúcej práci s danou kolekciou vždy dostanem zoradené dátá.

Výpočet RSI indikátoru som vysvetlil v teoretickej časti. Najskôr prebieha materializácia zatváracích cien do dátovej štruktúry pole, príkazom *_stocksHistory.Select(x => x.Close).ToArray()*. Pole je v tomto prípade pre mňa vyhovujúce, pretože môžem pristupovať k prvkom cez ich index, čo je pri výpočtoch tohto indikátoru klúčové. Následne, na základe parametru, ktorý predávam pri vytváraní stratégie spočítajú rozdiely posledných cien a uložia sa do premenných. Premena môže byť rastová alebo klesajúca, na základe toho, či je rozdiel väčší ako nula. Časť výpočtu je možné vidieť na nasledujúcom obrázku.

Obrázok 21: Časť výpočtu cenového rozdielu RSI indikátoru

```
var diff = price[i] - price[i - 1];
if (diff >= 0)
{
    gain += diff;
}
else
{
    loss -= diff;
}
```

Zdroj: Aplikácia TradeBash (vlastné spracovanie)

Následne sa dostávam k hlavnej časti výpočtu, takzvanej relatívnej sile ceny. K výpočtu priemerujem ceny s pozitívnym rozdielom, ktoré vydelím cenami, ktoré skončili s negatívnym rozdielom. Všetko na základe parametru indikátoru.

Obrázok 22: Časť výpočtu relatívnej sily RSI indikátoru

```
var diff = price[i] - price[i - 1];

if (diff >= 0)
{
    avrg = ((avrg * (period - 1)) + diff) / period;
    avrl = (avrl * (period - 1)) / period;
}
else
{
    avrl = ((avrl * (period - 1)) - diff) / period;
    avrg = (avrg * (period - 1)) / period;
}

rs = avrg / avrl;

rsi[i] = 100 - (100 / (1 + rs));
```

Zdroj: Aplikácia TradeBash (vlastné spracovanie)

Takto sa počíta hodnota relatívnej sily ceny, tento výpočet sa musí realizovať pre každú akciu a každý deň. Na základe tejto hodnoty realizuje obchodná stratégia nákupné príkazy.

5.4 Implementácia obchodnej stratégie

Po implementácii dátového skladu a príprave dát pre backtest, kde som napočítal hodnoty indikátorov pre celú história dát, ktorú mám k dispozícii, môžem prejsť k implementácii obchodnej stratégie. Začнем s najjednoduchším prístupom a na základe výsledkov sa budem snažiť pravidlá optimalizovať tak aby stratégia generovala čo najlepšie výsledky. Tento postup následne popíšem v sekcií Optimalizácia parametrov a experimentovanie.

5.4.1 API rozhranie

Ovládanie vytvárania stratégie je implementované cez API rozhranie v projekte *TradeBash.Web*. Toto API rozhranie implementuje niekoľko kontrolérov, čo sú metódy pomocou ktorých je možné ovládať aplikáciu. Samotnej integrácií API rozhrania a možnosti ako posielat' príkazy sa budem venovať v samostatnej kapitole.

Obrázok 23: API kontrolér vytvorenia stratégie

```
[HttpGet("calculateStrategy/{budget}/{smaShort}/{smaLong}/{rsi}")]
public async Task<IActionResult> CalculateStrategy(int budget, int smaShort, int smaLong, int rsi)
{
    var strategyName = $"Budget-{budget}-SMA-{smaShort}-SMA-{smaLong}-RSI-{rsi}";
    var strategy = Strategy.From(strategyName, budget, smaShort, smaLong, rsi);

    try
    {
        await CalculateAsync(strategy);
        return Ok();
    }
    catch (Exception e)
    {
        return UnprocessableEntity(e);
    }
}
```

Zdroj: Aplikácia TradeBash (vlastné spracovanie)

Obrázok 23 znázorňuje parametre API rozhrania, ako jednotlivé hodnoty indikátorov a investovaný kapitál. Na základe týchto hodnôt sa zostaví názov stratégie a vytvorí sa jej inštancia, cez ktorú sa následne spracovávajú všetky potrebné úkony. Tento kontrolér ma niekoľko preťažení. Postupne ako optimalizujem parametre vytváram ďalšie API rozhrania, pomocou ktorých je možné otestovať nový prístup obchodovania. V tejto metóde sa volá *CalculateAsync*, v ktorej sa z repozitáru získajú dátá, ktoré boli spracované pomocou implementácie dátového skladu. Ako je vidieť na obrázku 24, najskôr sa uloží inicializovaná stratégia, následne sa prechádzajú jednotlivé akcie z dátového skladu a pre každú akciu sa inštancia triedy *Strategy* postará o potrebné výpočty, ktoré potrebujem pre generovanie obchodných príkazov.

Obrázok 24: Získanie dát z repozitáru

```
private async Task CalculateAsync(Strategy strategy)
{
    var stocks = await _repository.ListAsync<Stock>();

    await _repository.AddAsync(strategy);
    foreach (var stock in stocks)
    {
        _logger.LogInformation($"Start indicators calculation for stock {stock.Name}");

        strategy.RunCalculationFor(stock);

        _logger.LogInformation($"Saving indicator calculations to database");

        await _repository.UpdateAsync(strategy);
    }

    _logger.LogInformation("Saving Finished successfully");
}
```

Zdroj: Aplikácia TradeBash (vlastné spracovanie)

5.4.2 Hlavná trieda Strategy

Na základe popisu architektúry systému v teoretickej časti je srdcom celej aplikácie. Práve v tejto entite budem implementovať jednotlivé prístupy obchodovania. Trieda dedí z rozhrania *IAggregateRoot* čo ju robí zodpovednou za inicializáciu ostatných entít, je zodpovedná za integritu dát celého systému a všetky požiadavky súvisiace s hlavnou logikou celého systému prúdia cez túto triedu.

Obrázok 25: Inicializácia triedy Strategy

```
public class Strategy : BaseEntity, IAggregateRoot
{
    public string Name { get; private set; }

    public double Budget { get; private set; }

    private int? _smaShortParameter;
    private int? _smaLongParameter;
    private int? _rsiParameter;

    public Drawdown _drawdown;
    private double _cumulatedBudgetClosePrice;

    public ICollection<StrategyStock> StrategyStocksHistory { get; set; }
    public ICollection<GeneratedOrder> GeneratedOrders { get; set; }
    public IReadOnlyCollection<GeneratedOrder> OrderedGeneratedOrdersHistory =>
        GeneratedOrders.OrderBy(x => x.CloseDate).ToList();

    private Strategy() {...}
}
```

Zdroj: Aplikácia TradeBash (vlastné spracovanie)

Obrázok 25 znázorňuje inicializáciu základných premenných: názov stratégie, kapitál a hodnoty parametrov. Dôležité sú kolekcie, história akcií *StrategyStocksHistory* a vygenerované príkazy *GeneratedOrders*.

História akcií je kolekcia, ktorá je naplnená dátami z predchádzajúceho kroku napočítania hodnôt indikátorov. Takto mám v tejto triede prístup k týmto dátam, na základe ktorých budem modelovať pravidlá obchodného prístupu. Druhá kolekcia generované príkazy sa bude postupne plniť potom ako sa budú prechádzať jednotlivé dni histórie a ako bude implementovaná stratégia generovať obchodné príkazy. S touto kolekciami bude v ďalších krococh pracovať reporting systém *Excelu*, ktorý bude potrebovať tieto vygenerované príkazy k počítaniu základných vyhodnocovacích ukazovateľov.

Samotná implementácia obchodných pravidiel bude závislá na tom, ako budem postupne testovať viaceré prístupov. Tento proces bude iteratívny a budúca implementácia bude vždy vychádzať z minulej. Napríklad môže ísť o zakomponovanie správy kapitálu, postupné navyšovanie pozície a podobne.

Obrázok 26: Implementácia pravidiel stratégie

```
foreach (var strategyStock in StrategyStocksHistory)
{
    if (StrategyGuard.IndexOutOfRange(index, strategyStock)) continue;

    var currentStock = strategyStock.CalculatedOrderedStocksHistory.ToList()[index];

    if (StrategyGuard.RsiNotCalculated(currentStock)) continue;
    if (StrategyGuard.NotSameDate(currentStock, inDate)) continue;

    if (GenerateBuySignalForRsiIfCurrentStockLower(generatedSignal, currentStock))
    {
        generatedSignal = currentStock;
    }

    var openPosition = GetCurrentNotClosedPositionFor(currentStock);
    if (openPosition == null) continue;

    if (currentStock.SMAShort < currentStock.Close)
    {
        openPosition.ClosePosition(currentStock.Close, currentStock.Date);
        openPosition.CalculateProfitLoss();
    }
}
```

Zdroj: Aplikácia TradeBash (vlastné spracovanie)

Predchádzajúci obrázok znázorňuje jednu z viacerých verzíí implementácií obchodného prístupu. Ide o prechádzanie jednotlivých dní histórie a vyhodnocovanie obchodov na základe definovaných pravidiel. K implementácií využívam pomocné metódy a triedy. Výhoda je v tom že napríklad tieto pomocné metódy môžem využiť pri nasledujúcich implementáciách obchodných prístupov. Medzi tieto metódy patria napríklad *GenerateBuySignalForRsiIfCurrentStockLower*. Táto metóda vygeneruje obchodných príkaz keď sa cena akcie nachádza pod definovanou hodnotou indikátoru relatívnej sily. Ďalšou je *GetCurrentNotClosedPositionFor*, ktorá vráti neuzatvorenú pozíciu nakúpenú v predchádzajúcich dňoch a mnoho ďalších.

Rovnako simulácia otvárania a zatvárania obchodných príkazov sa nachádza v tejto triede. K otvoreniu dôjde samozrejme na základe definovaných pravidiel v implementovanej obchodnej stratégií. Stratégia môže pracovať s fixnou správou kapitálu, čo znamená, že ak už existuje otvorená pozícia pre danú akciu tak sa znova nenakúpi. Alebo je možné pracovať s riedením existujúcich pozícii - ak je aktuálne otvorená pozícia a správa kapitálu dovoľuje nakúpiť do existujúcej pozície, cena sa spriemeruje a dokúpia sa ďalšie pozície pre danú akciu. Všetko závisí od toho, aké nastavenie práve budem testovať.

Obrázok 27: Otváranie nových pozícií v triede Strategy

```
private void OpenPositionAndGenerateOrder(CalculatedStock generatedSignal, int budgetPercentage)
{
    var currentOpenPosition = GetCurrentNotClosedPositionsFor(generatedSignal.Symbol);

    if (currentOpenPosition != null)
    {
        currentOpenPosition.UpdateOpenPrice(currentOpenPosition, generatedSignal);
        currentOpenPosition.UpdateCurrentPositions(Budget, budgetPercentage);
    }
    else
    {
        var generatedOrder = GeneratedOrder.OpenPosition(
            generatedSignal.Symbol,
            generatedSignal.Open,
            generatedSignal.Date,
            generatedSignal.GetIndicatorValues());
        generatedOrder.UpdateCurrentPositions(Budget, budgetPercentage);
        GeneratedOrders.Add(generatedOrder);
    }
}
```

Zdroj: Aplikácia TradeBash (vlastné spracovanie)

Na obrázku je vidieť, že pomocou metody *GetCurrentNotClosedPositionFor* sa vráti neuzatvorená pozícia, ak takáto existuje. Ak existuje, tak sa na základe aktuálnej ceny spriemeruje otváracia cena metódou *UpdateOpenPrice* a aktualizujú sa otvorené pozície metódou *UpdateCurrentPositions*. Druhý blok metody sa realizuje v prípade, že pre danú akciu neexistuje aktuálne otvorená pozícia. Jednoducho sa na triede *GeneratedOrder* zavolá metoda na otvorenie pozície so základnými údajmi, spočítajú sa aktuálne pozície metodou *UpdateCurrentPositions* a vygenerovaný príkaz sa vloží do kolekcie *GeneratedOrders*.

5.5 Integrácia Excelu a Reporting

Pre vyhodnocovanie jednotlivých prístupov obchodnej stratégie budem využívať Excel s knižnicou *EPPlus*. Potom, ako mi systém vygeneruje obchodné príkazy je potrebné ich adekvátne vyhodnotiť a reagovať na dané výsledky. K tomu mi bude slúžiť práve tento *Excel*, kde uložím realizované obchody a základne vyhodnocovacie ukazovatele. Následne zhotovím graf, na základe ktorého bude na prvý pohľad možné rozpoznať úspešnosť implementovanej stratégie. Takto budem generovať reporty a vyhodnocovať výkonnosť jednotlivých prístupov k danej obchodnej stratégií.

Integrácia samotnej knižnice sa nachádza v projekte *Tradebash.Infrastructure*, ktorý je zodpovedný za prácu týkajúcej sa samotných dát. Pre prácu s Excelom som vytvoril jednoduchý interface *IExcelReporting* ktorý implementuje metódu *GenerateAsync*. Táto metóda je zodpovedná za samotné generovanie Excelu, ktorý obsahuje základné štatistiky analyzovanej stratégie a vygenerované obchody.

Predtým, ako je možné používať toto rozhranie je potrebné ho zaregistrovať do Dependency Injection kontajneru. Toto rozhranie je zaregistrované pomocou takzvanej *Scoped* metódy, ktorá znamená, že vždy keď pošlem z klienta požiadavku, je vytvorená nová inštancia tohto rozhrania.

Po registrácii je potrebná samotná implementácia rozhrania *IExcelReporting*. V triede, ktorá implementuje toto rozhranie dochádza k vytvoreniu Excel súborov s názvami definovaných parametrov pre testovanú stratégiu. Ďalej dochádza ku kontrole či existujú dané Excel súbory. Ak áno, pôvodné sa zmažú a vytvoria sa nové. Následne pracujem s knižnicou *EPPlus*, pomocou ktorej definujem pozície v tabuľke, vytváram samotný report a exportujem dátu vygenerovaných obchodov. Dôležité sú aj agregátne funkcie, v ktorých počítam základne ukazovatele dôležité pre vyhodnocovanie reportu. Tieto agregátne funkcie obaľuje statická trieda *StrategyStats*, kde implementujem statické metódy slúžiace pre výpočet čistého zisku, maximálneho prepadu, percentuálnej úspešnosti, profit faktoru a ďalších ukazovateľov popísaných v teoretickej časti vyhodnocovanie. Všetky vyhodnocovacie metódy sú zobrazené na nasledujúcim obrázku.

Obrázok 28: Statická trieda na výpočet základných ukazovateľov

```
var nettProfit = StrategyStats.GetNettProfit(strategy);
var profitFactor = StrategyStats.GetProfitFactor(strategy);
var endingCapital = StrategyStats.GetEndingCapital(strategy);
var number0fTrades = StrategyStats.GetNumber0fTrades(strategy);
var testedHistory = StrategyStats.GetTestedHistory(strategy);
var winnersPercentage = StrategyStats.GetPercentageWinners(strategy);
```

Zdroj: Aplikácia TradeBash (vlastné spracovanie)

Tieto základne ukazovatele sú následne pomocou knižnice *EPPlus* spracované a vyexportované do Excelu.

5.6 Integrácia Swagger API rozhrania

S aplikáciou je možné komunikovať pomocou Swagger API rozhrania. Integrácia tejto technológie v prostredí .Net Core je veľmi jednoduchá. Potom ako nainštalujeme daný NuGet balíček je potrebné zaregistrovať servisu v triede *Startup* a metóde *ConfigureServices*.

Obrázok 29 znázorňuje základné použitie rozširujúcej metódy *AddSwaggerGen*. Pomocou tejto metódy definujem základné informácie, ako verzia API rozhrania a titul, ktorý slúži na identifikáciu.

Obrázok 29: Registrácia Swagger knižnice

```
services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo
    {
        Title = "TradeBash API",
        Version = "v1"
    });
    c.EnableAnnotations();
});
```

Zdroj: Aplikácia TradeBash (vlastné spracovanie)

Na záver potrebujem nakonfigurovať použitie automaticky generovaného užívateľského rozhrania, pomocou ktorého komunikujem s aplikáciou. Stačí v metóde *Configure* zavolať rozširujúce metódy *UseSwagger* a *UseSwaggerUI*. Takto mám pripravenú knižnicu k používaniu.

5.7 Optimalizácia parametrov a experimentovanie

Všetko mám pripravené k tomu aby som mohol simulovať obchodovanie, tak ako ho pôvodne predstavil Larry Connors a snažiť sa prístup vylepšiť na základe reportov z implementovaného systému. Tento systém má množstvo variácií, ktoré môžu rozhodnúť o tom, či ide o úspešný prístup alebo obchodník príde o celý investovaný kapitál. Ani sám Larry neponúkol finálnu verziu tohto systému a je na každom obchodníkovi ako s informáciami naloží a overí si funkčnosť prístupov na vlastných testoch a prispôsobí prístup tak, aby vyhovoval jeho kritériám.

Začнем s jednoduchým postupom, kde sa na základe výsledkov budem pokúšať optimalizovať stratégiu tak, aby generovala čo najviac úspešných obchodov s čo najmenším rizikom. Samozrejme, ku každému testu pridám report so základnými štatistikami úspešnosti testovanej stratégie a parametrami, s ktorými som docielil dané výsledky.

5.7.1 Postup testovania

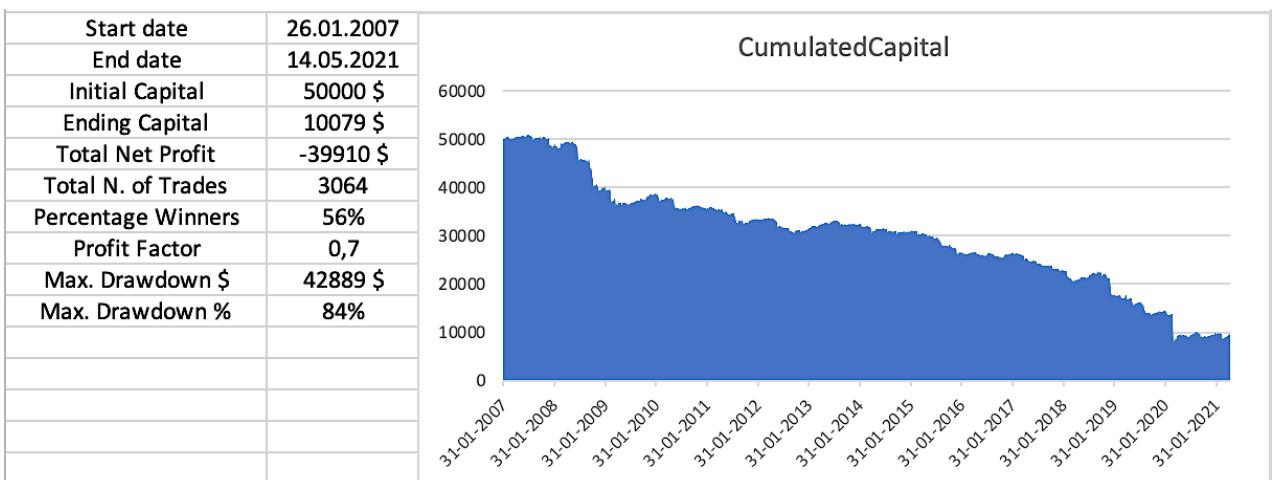
V teoretickej časti som popísal testovanú stratégiu. V krátkosti chcem ale rekapitulovať spomínané pravidlá. Na určenie nákupných signálov používam *RSI* momentum indikátor s parametrom 2. Tento prístup identifikuje akcie, ktoré momentálne klesajú, čím je tento pokles väčší tým bude

výsledná hodnota indikátoru nižšia. Akcia s najnižšou hodnotou sa tak v daný deň nakúpi. Na predajné signály používam jednoduchý priemer s nižšou hodnotou parametru. Hodnotu parametru budem ďalej testovať na základe toho, aké výsledky budú generovať reporty. Na vyfiltrovanie akcií, ktoré sú dlhodobo v poklese použijem spomínaný jednoduchý priemer s parametrom 200. Ked' sa cena akcie nachádza pod hodnotou dlhodobého jednoduchého priemeru bude tak obchodným systémom ignorovaná. Pri testovaní začнем s relatívne jednoduchým prístupom a postupne budem do systému zahrňovať ďalšie pravidlá, tak aby bolo zrejmé aký vplyv majú jednotlivé parametre na celkovú výkonnosť. Systém bude testovaný na dátach akciových indexov SPY, SP100, QQQ, VTV. Z dôvodu dĺžky práce vyberiem len tie reporty, ktoré pre daný index budú generovať najlepšie výsledky.

5.7.2 Test s jednoduchým priemerom a momentom

Prvý test s jednoduchým priemerom a RSI indikátorom. Pravidlá sú jednoduché, nakupujem vždy ked' hodnota *RSI* indikátoru s parametrom 2 je nižšia ako 10. Každý deň sa môže nakúpiť maximálne jedná akcia z vybraného koša. Investuje sa presne 5 percent z celkového účtu na jednu akciu. Maximálny počet pozícii je obmedzený na 20. Teda, ked' sa investuje 5 percent z celkového účtu pri súčasne otvorených 20 pozícii mám zainvestovaný celý kapitál. Výstup z obchodu nastáva v prípade ked' sa cena nachádza nad 15 dňovým jednoduchým priemerom. Testoval som výstup pre jednoduchý priemer od hodnoty 5 a postupne som pridával 5 až do hodnoty 40. Najlepšie pre výstup vychádzala práve hodnota 15.

Tabuľka 4: Test s jednoduchým priemerom a RSI indikátorom



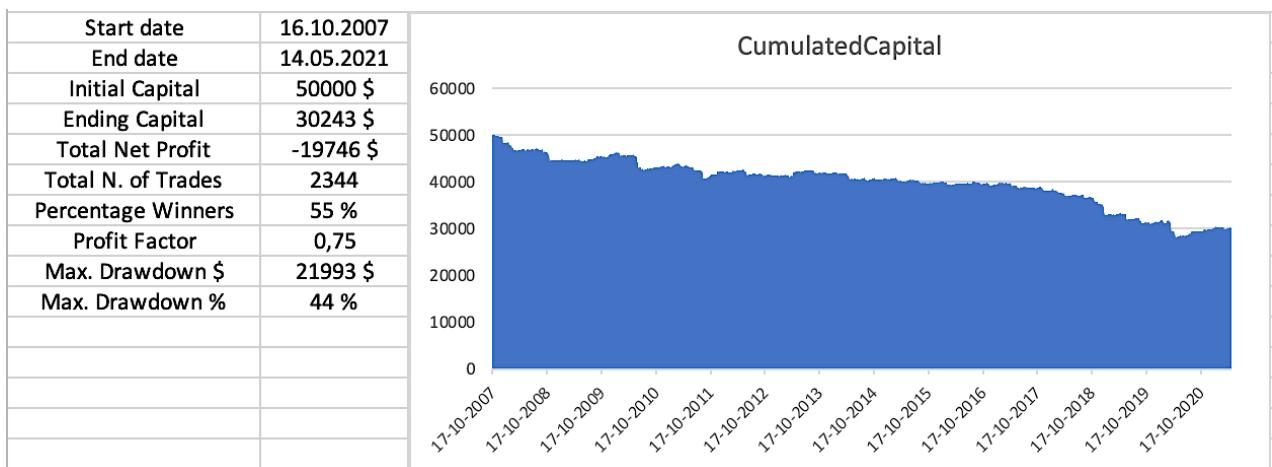
Zdroj: Aplikácia TradeBash (vlastné spracovanie)

Ako je vidieť z reportu, obchodný prístup prakticky od začiatku obchodovania ihneď prechádza do straty, z ktorej sa v celej histórii obchodovania nedostal. Percentuálna úspešnosť obchodov je 56 percent čo je mierne nad polovicou a nestačí to pre pozitívne výsledky. Stratové obchody generujú v priemere väčšie straty ako tie ziskové. Celkovo systém vygeneroval 3064 obchodných príkazov. Kombinácia SMA indikátor s parametrom 15 na signál pre výstup a RSI indikátor ako signál pre vstup vôbec nefungujú.

5.7.3 Test s pridaným dlhodobým priemerom

Bude zaujímavé sledovať ako výsledkom pomôže dlhodobý priemer s hodnotou 200. Tento indikátor by mal vyfiltrovať akcie, ktoré sa nenachádzajú v býcom trende. Toto by určite malo znížiť počet vygenerovaných obchodov.

Tabuľka 5: Test s pridaným dlhodobým priemerom



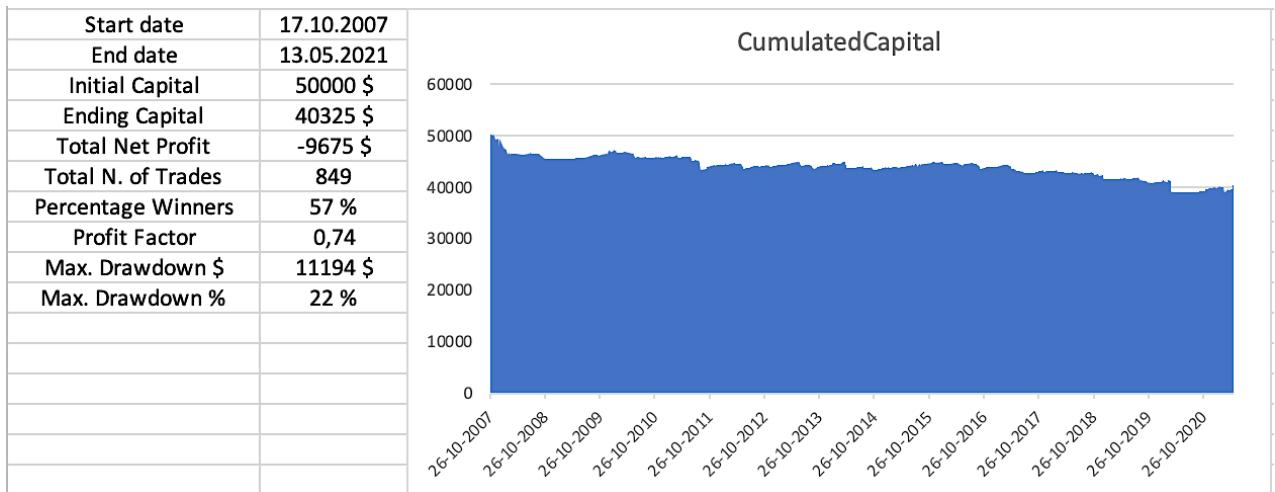
Zdroj: Aplikácia TradeBash (vlastné spracovanie)

Nastavenie testu SMA indikátor s parametrom 20 pre výstup, pridaný dlhodobý priemer na vyfiltrovanie obchodov s parametrom 200 a RSI indikátor. Toto nastavenie vygenerovalo najlepšie výsledky. Znížil sa počet obchodov cca o 600, čo je určite dobré, pretože budem mať menšie náklady na komisie. Znížil sa počet obchodov, ale neubralo sa na výkonnosti, práve naopak, celkový prepad nie je taký vysoký ako pri teste bez dlhodobého priemeru a znížil sa o celú polovicu z pôvodných 84 percent na 44. Aj keď percentuálna výkonnosť klesla o 1 percento, profit faktor sa zvýšil z hodnoty 0,7 na 0,75. Konečný zostatok účtu tak stúpol z 10 079 dolárov na 30 243. Stále však nejde o ziskový systém.

5.7.4 Test s optimalizáciou RSI indikátoru

Z predchádzajúcich testoch mi najlepšie pre výstup s jednoduchým priemerom vychádzala hodnota 15. Pri nasledujúcom testovaní som sa zameral hlavne na hodnotu RSI indikátoru.

Tabuľka 6: Test s optimalizáciou RSI indikátoru



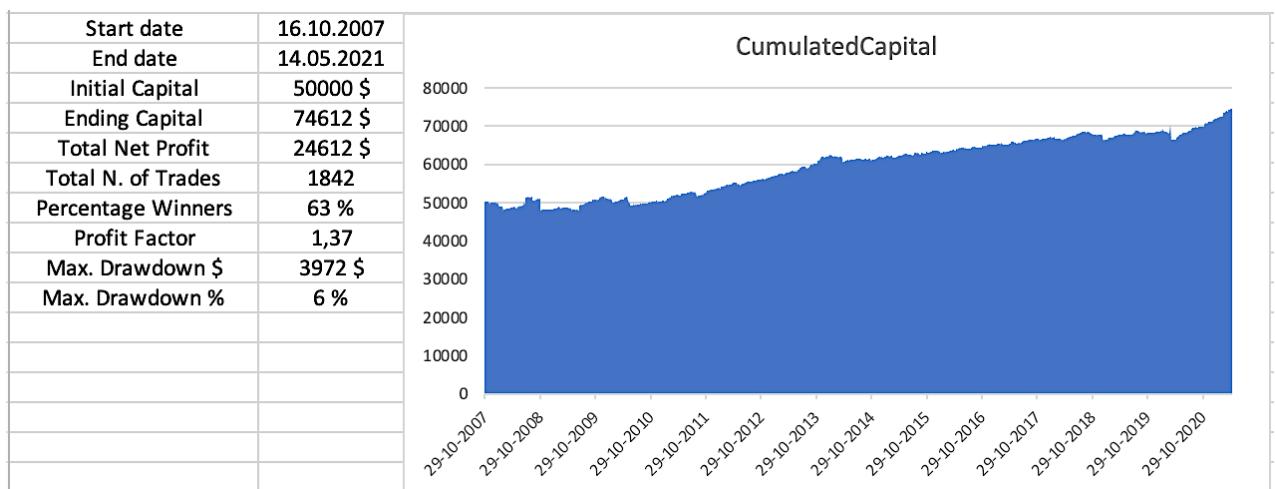
Zdroj: Aplikácia TradeBash (vlastné spracovanie)

Nastavenie testu jednoduchý priemer s parametrom 15, dlhodobý priemer s parametrom 200 a RSI indikátor parameter 2 pre ktorý bola nastavená hodnota 3. Celková výnosnosť z posledného testu stúpla z hodnoty 30 243 dolárov na 40 325. Počet obchodov je trikrát menší. Maximálny prepad sa znížil o polovicu. Ako som postupne znižoval hodnotu RSI indikátoru, tak klesal počet obchodov a stúpala celková výkonnosť. Dôvod je ale jednoduchý, tým že znižujem hodnotu indikátoru, tým sú pravidlá pre vstup prísnejsie a redukuje sa celkový počet. Stále sa však pohybujem v mínusových číslach to znamená keby mi stratégia nevygenerovala žiadny obchod tak som na 0. Čo by bol z doposiaľ vygenerovaných testoch najlepší výsledok.

5.7.5 Test s dokupovaním do existujúcich pozícií na základe RSI indikátoru

Ďalšia z možných úprav je dokupovanie do už existujúcich pozícií. Dopolňoval som testoval prístup, kde môže byť maximálne jedná pozícia pre akciu, čo znamená, že ak akcia znova vygeneruje signál nedokúpi sa. Maximálny počet slotov zostáva 20, ale je možné dokupovať už do existujúcich pozícií. Záleží či vygenerovaný signál bude mať najnižšiu hodnotu RSI indikátoru spomedzi všetkých akcií za daný deň.

Tabuľka 7: Test s dokupovaním do existujúcich pozícii na základe RSI indikátoru



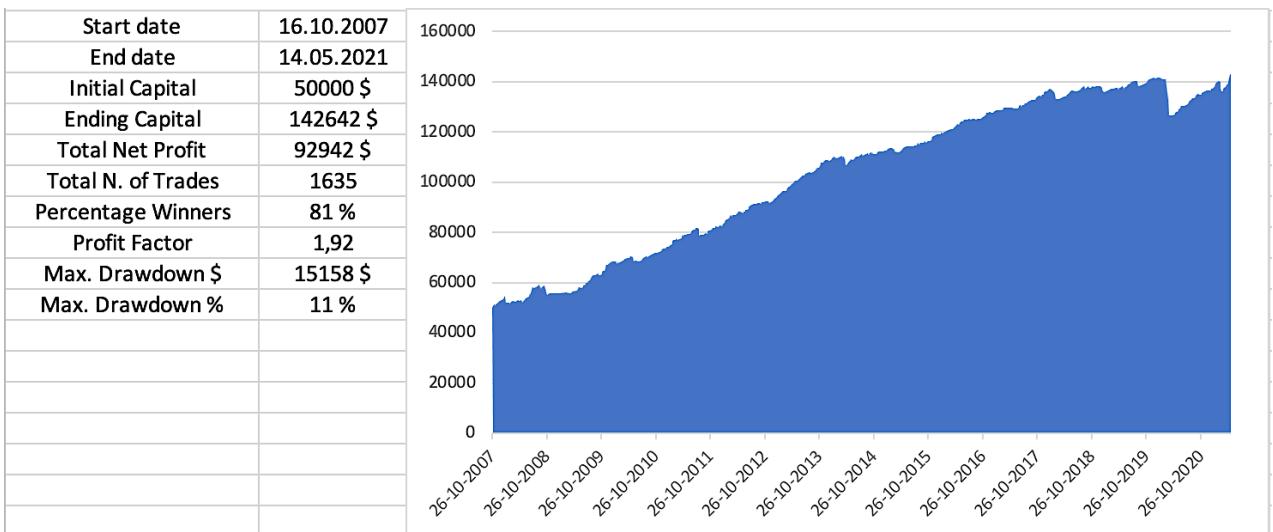
Zdroj: Aplikácia TradeBash (vlastné spracovanie)

Konečne sa dostávam do plusových čísel. Dokupovanie do existujúcich pozícii funguje. Polepšil som si v každom smere, celková výkonnosť sa zvýšila, zvýšila sa úspešnosť obchodov na 63 percent rovnako profit faktor kde za 1 dolár dostanem naspäť 1,37 doláru a aj maximálny prepad sa znížil o viac ako polovicu na 6 percent. Dopolňovala táto zmena najväčší vplyv na výkonnosť stratégie, čo je zaujímavé, lebo v dostupných zdrojoch som sa dočítal hlavne o rôznych nastaveniach indikátorov. Pritom indikátory nemali taký vplyv na výkonnosť ako správa kapitálu a dokupovanie do existujúcich pozícii.

5.7.6 Test s dokupovaním do existujúcich pozícii pri nižšej cene

Nasledujúci test obsahuje jednu základnú zmenu. Pri predchádzajúcim teste som dokupoval do existujúcich pozícii, ale len v prípade, že hodnota RSI indikátoru bola najnižšia z celého koša. V tomto teste je možné dokupovať do existujúcich pozícii v prípade, že cena je nižšia ako cena posledného nákupu. To znamená, keď mám otvorené napríklad 3 pozície a cena každej bude nasledujúci deň nižšia, pozície sa dokúpia do všetkých troch v prípade, že to kapitál dovolí. Je možných maximálne 20 nákupov ako v predchádzajúcich testoch a do jednej pozícii sa môže dokúpiť maximálne 5 krát. Je to z dôvodu, že nechcem aby všetkých 20 dokupov obsahovala jedna akcia, čo sa teoreticky môže stať.

Tabuľka 8: Test s dokupovaním do existujúcich pozícii pri nižšej cene



Zdroj: Aplikácia TradeBash (vlastné spracovanie)

Test s dokupovaním do existujúcich pozícii v prípade, keď je cena v nasledujúcich dňoch nižšia vychádza takmer vo všetkých smeroch lepšie ako všetky doteraz testované prístupy. Celkový kapitál sa ku dňu 14. 05. 2021 takmer strojnásobil. Počet obchodov sa oproti predchádzajúcemu testu mierne znížil z 1842 na 1635. Percentuálna výkonnosť je mierne nad 80 percent, z čoho je možné jednoznačne usúdiť, že systém s týmito pravidlami funguje na historických dátach a pri vstupoch do obchodov dokáže skutočne ľažiť z nastavenia indikátorov a dokupovania do existujúcich pozícii. Profit Faktor sa blíži k hodnote 2, čo hovorí, že celkový súčet ziskových obchodov je skoro dvakrát väčší ako celkový súčet stratových obchodov a za jeden investovaný dolár dostanem skoro 100 percentnú návratnosť. Maximálny prepad sa oproti predchádzajúcemu testu mierne zvýšil na hodnotu 11 percent. Zaujímavé je, že tento najväčší prepad spôsobila korona kríza, ktorá začala v marci 2020 a mala samozrejme veľký dopad na akciové trhy.

5.8 Vyhodnotenie prístupov

V predchádzajúcej sekcií je vidieť aký vplyv na výkonnosť majú jednotlivé prístupy. Stratégiu som implementoval postupne a reporty generoval priebežne, s tým ako som testoval rôzne nastavenia parametrov a pravidiel. Takto som získal lepší obraz toho, ako môže celková výkonnosť reagovať na rôzne nastavenia stratégie.

5.8.1 Testované koše akcií

Testy boli realizované na akciových indexoch SP100, SP500, QQQ, VTV. Prvá zaujímavá skutočnosť, ktorú som odpozoroval je, že z neznámeho dôvodu vychádzajú najlepšie výsledky na indexe SP100. Testované indexy SP500, QQQ, VTV vykazovali horšie výsledky. Domnieval som sa že to bude množstvom zastúpených akcií v indexe, keďže SP500 obsahuje 500 akcií a VTV 337. Čakal som od QQQ ktorý obsahuje 104 akcií podobné výsledky ako index SP100, keďže množstvom sú podobné. Ale ani index QQQ nedosahuje v testoch na výkonnosť SP100. Táto skutočnosť sa prejavovala na všetkých testovaných prístupoch. Možno ide len o náhodu a jednoducho, čo sa týka história a testovanej stratégii, najlepšie sedeli testy indexu SP100.

5.8.2 Výkonnosti prístupov

Čo sa týka samotnej výkonnosti, tak pomocou cenových indikátorov jednoduchého priemeru a momentum indikátoru relatívnej sily sa mi nepodarilo dostať do zisku. Zo začiatku to vyzeralo, že stratégia na mnou testovaných dátach vôbec nefunguje. Najväčšia zmena nastala keď som do obchodovania pridal nakupovanie do existujúcich pozícií a jednoduchú správu kapitálu. Celkové výsledky stratégie s rovnakými indikátorami a parametrami sa niekoľko násobne zlepšili a systém začal generovať zaujímavé výnosnosti.

Pri porovnaní s indexom SP500, ktorý pri investícii 50 000 dolárov od 1. 10. 2007 zhodnotil kapitál na približne 130 000 dolárov, si posledné verzie stratégie viedli dobre. Zaujímavý je maximálny prepad, ktorý pri SP500 v roku 2008 bol až skoro 45 percent. Naproti tomu mala posledná verzia stratégie prepad len 11 percent. Výnosnosť stratégie činila 142 642 dolárov, čím prekonala index SPY.

Ked'že takýto prístup k investovaniu vyžaduje určité znalosti, obchodný systém a aktívny prístup, nebudem porovnávať výkonnosť s voľne dostupnými nástrojmi finančných bank. Investične sa jedná o dva úplne rozdielne svety. Investovanie pomocou automatického systému môžem prirovnáta k podnikateľskej činnosti, ktorá je samozrejme spojená s množstvami rizík. U nástrojov finančných bank sa jedná o pasívne investovanie.

5.8.3 Možné riziká

Pri vyhodnotení rizikovosti je hlavným ukazovateľom maximálny prepad. Je ale potrebné zasadíť tento ukazovateľ do celkového kontextu. Niektoré reporty generovali lepší celkový zisk, ale s

väčším celkovým prepadom. Šlo hlavne o to, aké agresívne bolo dokupovanie do existujúcich pozícií. Záleží od každého obchodníka, aké veľké percento účtu je ochotný riskovať. Nieko preferuje menšie riziko, zato samozrejme inkasuje menší zisk.

Kedy je najvhodnejšie nasadiť stratégiu na trh? Ked' zhodnotím vývoj akumulovanej kriky zisku na posledných grafoch tak je na nej možné vidieť len mierne prepady. Čo hovorí o tom, že aj ked' trh zaznamenával väčšie výkyvy na stratégiu to malo mierne dopady. Samozrejme, je potrebné počítať stým, že ked' sa na trhu vyskytne kríza a celkovo bude všetko klesať, stratégia bude generovať straty. Ak táto kríza bude trvať dlho a akcie sa budú nachádzať pod dlhodobým priemerom, nebude stratégia generovať obchodné príkazy a tak sa bude chrániť pred týmito prepadmi. Celkovo ale môžem povedať, že sa stratégii bude dať, ked' celkový trh bude rásť.

Ďalším dôležitým faktom je príliš veľká alokácia nákupov do jednej akcie. V poslednom nastavení testovanej stratégie som nastavil pravidlo na maximálne 5 nákupov. To znamená, že pre jednu akciu môže byť zainvestovaných maximálne 25 percent kapitálu ak investujem 5 percent účtu. Z môjho pohľadu sa jedná o dosť agresívny nákup. Je dôležité si položiť otázku, čo sa stane ked' sa akcia v ktorej som alokovaný dostane do problémov. Teoreticky riskujem až 25 percent z celkového účtu. Na mnou testovaných dátach sa daná situácia nevyskytla, to ale neznamená, že sa niečo podobné nemôže stať v budúcnosti.

Riziká sa netýkajú len samotného obchodovania, ale aj problémov s realizáciou a implementáciou systému, ktorý bude takúto obchodnú stratégii automaticky obchodovať. Systém musí spracovávať obrovské množstvo dát. Dôležité je implementovať mechanizmy, ktoré zaručia že sú tieto dátá správne a aktuálne. Nad týmito dátami systém realizuje počítanie indikátorov a vyhodnocuje pravidlá obchodnej stratégie. Nemôžem sa spoľahnúť nato, že som všetko správne implementoval, je potrebné mať hlavné časti pokryté testami, aby som si bol istý správnosťou vyhodnocovania pravidiel. Takýto automatický systém by mal vedieť posielat' notifikácie o hlavných udalostiach: napríklad nákupoch, predajoch alebo výpadkoch systému.

5.8.4 Obchodovanie v reálnom prostredí

V prípade rozhodnutia sa obchodovania popisovanej stratégie pomocou implementovaného obchodného systému v reálnom prostredí je potrebné realizovať niekoľko krokov. Implementovaný systém v práci neobsahuje exekučný modul, ktorý by bol schopný posielat' nákupné a predajné príkazy na burzu. Čo znamená implementáciu takéhoto exekučného modulu. Celý systém by bolo potrebné nasadiť, napríklad do cloudu, kde by mohol bežať 24 hodín denne. Ďalší z nutných krokov

je otvorenie účtu v spoločnosti, ktorá poskytuje obchodovanie na burze. Táto spoločnosť musí poskytovať API, pomocou ktorého by môj systém komunikoval s externým systémom spoločnosti, pre posielanie obchodných príkazov. Najdôležitejšie je zvážiť všetky možné riziká popísane v predchádzajúcej sekcií, predtým ako s reálnym obchodovaním začínať.

Záver

Ako hlavný cieľ práce bolo implementovať a analyzovať obchodný systém založený na myšlienkach od *Larryho Connorsa*, ktoré popisuje v knihe *Short Term Trading Strategies That Work*. Tento systém aplikovať na koše akcií, kde sa vyberá zo stovky kandidátov. Hlavné myšlienky a vyhodnocovanie systému som popísal v teoretickej časti. Stručne som vysvetlil základné princípy burzy, technickú analýzu a indikátory, ktoré sú pre tento systém použité. Informácie sú doplnené o grafy na ktorých prezentujem myšlienky a vysvetľujem jednotlivé prístupy. Druhým krokom bolo tieto myšlienky implementovať a analyzovať na reálnych dátach. Navrhol som implementáciu aplikácie, ktorá takýto systém dokáže otestovať. Pri návrhu som vychádzal z princípov *Domain-Driven Designu*, kde najhlavnejšie časti som popísal a doplnil o grafickú reprezentáciu.

V praktickej časti som realizoval myšlienky popísané v teoretickej časti a implementoval automatický systém, ktorý dokáže overiť popísané postupy na reálnych dátach, pomocou Frameworku .NET Core a programovacieho jazyka C#. V rámci optimalizácie nákladov na dátu od externého providera som implementoval dátový sklad. Tento sklad slúži ako zdroj dát pre následné vyhodnocovanie. Takže pri následnom testovaní stratégii som čerpal dátá z dátového skladu, externého providera som použil len na uloženie dát do tohto skladu. Implementoval som indikátory použité pre definovanie pravidiel obchodnej stratégie, integroval API rozhranie a EPPlus knižnicu pre generovanie Excel reportov. Na základe výsledkov z reportov som sa rozhodoval pre vlastné úpravy stratégie, ktoré som následne integroval a znova pomocou reportov vyhodnocoval. V závere praktickej časti práce som na základe výsledkov z implementovaného obchodného systému popísal riziká, ktoré sú s takýmto obchodovaním spojené. Rovnako som načrtol, načo všetko je potrebné myslieť v prípade realizácie systému v živom prostredí.

Čo sa týka implementácie a znovupoužiteľnosti systému. Spracovanie dát je nezávislé na stratégii, to znamená že môže byť použité pre akýkoľvek typ obchodného prístupu. Rovnako integrovaný Excel reporting a vyhodnocovanie je pripravené pre iné prístupy obchodovania. Počítanie hodnôt indikátorov je možné opäťovne použiť pre iné typy stratégii. Systém ako taký nebude mať problém s technickou analýzou iných druhov obchodných prístupov.

Výkonnosť stratégie bez toho aby som nepoužil základnú správu kapitálu nefungovala. Takto sa mi na mnou testovacích dátach nepotvrdili myšlienky z úvodu práce. Indikátory s rôznymi nastaveniami nefungovali a stratégia vždy skončila v záporných číslach, kde by som v niektorých prípadoch prišiel o celý investovaný kapitál. Zlom nastal až v momente, keď som do stratégie zakomponoval dokupovanie do existujúcich pozícií. Z mojich výsledkov mala práve táto zmena

najväčší účinok na výkonnosť. Posledná verzia stratégie prekonala výkonnosť indexu SPY.

Pri hodnotení výsledkov je dôležité si byť vedomý *survivorship bias* efektu, ktorý som rovnako popísal v teoretickej časti. Z dôvodu komplexity systému a náročnosti získať dátu, ktoré týmto problémom netrpia som takéto riešenie v práci nerealizoval. Preto je realita výsledkov daných testov mierne skreslená, táto odchýlka od skutočnosti by ale nemala byť veľká, keďže som stratégii testoval na veľkom množstve akcií.

Potenciál na rozšírenie práce je v niekoľkých smeroch. Najdôležitejší bod považujem zistenie skutočnosti, aký veľký vplyv mala správa kapitálu a dokupovanie do pozícií na celkové výsledky stratégie. Určite by bolo v nasledujúcom rozširovaní vhodné, venovať čas tejto problematike. Som si istý, že výkonnosť systému má práve v tomto smere ešte priestor.

Ďalším bodom na vylepšenie by mohla byť automatická optimalizácia parametrov na základe špecifikovaných vyhodnocovaných ukazovateľov. Jednoducho, v súčasnom systéme som generoval reporty postupne s rôznymi nastaveniami indikátorov. Čo by mi mohlo ušetriť čas je, že systém sám dokáže generovať rôzne parametre a sám vyhodnocovať reporty, v závere by vybral napríklad len 5 najúspešnejších, ktoré by následne vygeneroval.

V prípade rozhodnutia sa obchodovania stratégie v reálnom burzovom prostredí, by bolo nutné implementovať do systému automatický exekučný systém, ktorý dokáže posielat nákupné a predajné príkazy na burzu. Čo je možné chápať ako ďalšie rozšírenie systému.

Conclusion

The main goal of the work was to implement and analyse a trading system based on ideas from Larry Connors, which he describes in the book Short Term Trading Strategies That Work. To apply this system to baskets of shares, where it selects from hundreds of candidates. I described the main ideas and evaluation of the system in the theoretical part. I briefly explained the basic principles of the stock exchange, technical analysis and indicators that are used for this system. The information is supplemented by graphs on which I present ideas and explain individual approaches. The second step was to implement and analyse these ideas on real data. I proposed the implementation of an application that can test such a system. The design was based on the principles of Domain-Driven Design. I described the main parts of the design and added a graphic representation.

In the practical part, I implemented the ideas described in the theoretical part. I implemented an automatic system that can verify the described procedures on real data, using .NET Core Framework and the C# programming language. As part of cost optimization, I implemented a data warehouse for data from an external provider. This warehouse serves as a source of data for subsequent evaluation. I used an external provider only to store data in this warehouse and during the subsequent testing of strategies, I drew data from the data warehouse. I implemented the indicators used to define the rules of the trading strategy, integrated the API interface and the EPPlus library for generating Excel reports. Based on the results of the reports, I decided to do my own adjustments to the strategy. I subsequently integrated them and re-evaluated results using reports. At the end of the practical part of the thesis, I described the risks associated with trading based on the results of the implemented trading system. I also outlined what is to consider when using the system in a live environment.

Regarding the implementation and reusability of the system. Data processing is strategy independent, meaning that it can be used for any type of trading strategy. Also integrated Excel reporting and evaluation is ready for other trading approaches. The calculation of indicator values can be reused for other types of strategies. As such, the system will not have a problem with the technical analysis of other types of approaches.

The performance of the strategy did not work without using basic capital management. Thus, the ideas from the beginning of the work were not confirmed on the test data. Indicators with different settings did not work and the strategy always ended in negative numbers, where in some cases I would lose all the invested capital. The turning point came only when I incorporated the repurchase into existing positions in the strategy. From my results, it was this change that had the

biggest effect on performance. The latest version of the strategy outperformed the SPY index.

When evaluating the results, it is important to be aware of the survivorship bias effect, which I also described in the theoretical part. Due to the complexity of the system and the complexity of obtaining data that do not suffer from this problem, I did not implement such a solution as part of the thesis. Therefore, the reality of the test results is slightly distorted, but this deviation from reality should not be large, as I tested the strategy on a large number of stocks.

The potential for expanding is in several directions. I consider the most important point to be to find out how big is the impact of capital management and repurchase on the overall results of the strategy. It would certainly be beneficiary to devote time to this issue in the next enlargement. I am sure that the performance of the system still has room in this direction.

Another point for improvement could be the automatic optimization of parameters based on the specified evaluated indicators. Simply put, in the current system, I generated reports sequentially with different indicator settings. What could save me time is, that the system itself can generate various parameters and evaluate reports itself. In the end it would select, for example, only the 5 most successful, which it would subsequently generate.

In the case of a decision to trade strategy in a real stock exchange environment, it would be necessary to implement an automatic execution system that can send buy and sell orders to the stock exchange. This can be considered as another extension of the system.

Zoznam tabuliek

Tabuľka 1: Kôš akcií indexu SPY 500.....	25
Tabuľka 2: Prepočítanie podielov akcií spoločnosti Apple	26
Tabuľka 3: Použité NuGet balíčky.....	38
Tabuľka 4: Test s jednoduchým priemerom a RSI indikátorom.....	59
Tabuľka 5: Test s pridaným dlhodobým priemerom.....	60
Tabuľka 6: Test s optimalizáciou RSI indikátoru	61
Tabuľka 7: Test s dokupovaním do existujúcich pozícií na základe RSI indikátoru.....	62
Tabuľka 8: Test s dokupovaním do existujúcich pozícií pri nižšej cene	63

Zoznam grafov

Graf 1: Cenové sviece	17
Graf 2: Ročný graf rastu cien Amazonu s trendovou linkou	18
Graf 3: Ročný graf rastu cien Amazonu so 100 dňovým cenovým priemerom.....	19
Graf 4: Vývoj RSI indikátoru.....	20
Graf 5: Relative strength index rovnica	21

Zoznam obrázkov

Obrázok 1: Fyzický pohľad na architektúru aplikácie	32
Obrázok 2: Vrstvy aplikácie.....	33
Obrázok 3: Návrh jadra	35
Obrázok 4: Návrh dátového skladu.....	36
Obrázok 5: Návrh automatického mechanizmu dátového skladu	37
Obrázok 6: Implementácia DbContextu pre prácu s Entity Framework.....	40
Obrázok 7: Databázový diagram.....	41
Obrázok 8: Vygenerované migrácie na úpravu databázového modelu.....	42
Obrázok 9: Repository rozhranie pre základnú prácu s entitami v databáze	42
Obrázok 10: Implementácia repository rozhrania	43
Obrázok 11: Metoda na získanie všetkých stratégí aj s ich závislosťami	43
Obrázok 12: Registrácia DataPopulator triedy do Dependency Injection kontajneru	44
Obrázok 13: Implementácia metódy ExecuteAsync	45
Obrázok 14: Implementácia metódy StopAsync.....	45
Obrázok 15: Poskytnutie servis z Dependency Injection kontajneru triedou ServiceProvider.....	45
Obrázok 16: Implementácia metódy DataPopulator	46
Obrázok 17: Neserializované dátá spoločnosti Apple od providera IEXCloud	47
Obrázok 18: Serializované dátá spoločnosti Apple od IEX providera.....	48
Obrázok 19: Implementácia metódy pre počítanie hodnôt indikátorov	49
Obrázok 20: Implementácia SMA indikátoru	50
Obrázok 21: Časť výpočtu cenového rozdielu RSI indikátoru	50
Obrázok 22: Časť výpočtu relatívnej sily RSI indikátoru	51
Obrázok 23: API kontrolér vytvorenia stratégie	52
Obrázok 24: Získanie dát z repozitáru	53
Obrázok 25: Inicializácia triedy Strategy	54
Obrázok 26: Implementácia pravidiel stratégie	55
Obrázok 27: Otváranie nových pozícií v triede Strategy	56
Obrázok 28: Statická trieda na výpočet základných ukazovateľov	57
Obrázok 29: Registrácia Swagger knižnice	58

Zoznam použitej literatúry

1. **Cunningham, Lawrence.** *The Essays of Warren Buffett: Lessons for Corporate America, Fifth Edition.* North Carolina : Carolina Academic Press, 2019. s. 84, ISBN: 1531017509.
2. **Cagan, Michele.** *Investing 101: From Stocks and Bonds to ETFs and IPOs, an Essential Primer on Building a Profitable Portfolio.* United State of America : Adams Media, 2016. ISBN: 1-4405-9513-5.
3. **Mitchell, Cory.** Forex. *Investopedia.* [Online] 5.1.2021. [cit. 2021-07-11]. Dostupné z: <https://www.investopedia.com/terms/f/forex.asp>.
4. **Jack, Schwager D.** *Getting started in technical analysis.* United States of America : John Wiley & Sons, 1999. ISBN: 0471295426.
5. **Wilder, J. Welles.** *New concepts in technical trading systems.* North Carolina : Trend Research, 1978. s.63-70, ISBN: 0894590278.
6. **Larry Connors, Cezar Alvarez.** *Short Term Trading Strategies That Work.* United States of America : TradingMarkets, 2009. s.11, ISBN: 0981923909.
7. **Laurence Connors, Cesar Alvarez, Matt Radtke.** An Introduction to ConnorsRSI. [Online] 2012, [cit. 2021-05-15]. Dostupné z: <https://alvarezquanttrading.com/wp-content/uploads/2016/05/ConnorsRSIGuidebook.pdf>.
8. **Teo, Rayner.** The Moving Average Trading Strategy Guide. *Trading with Rayner.* [Online] 16.9.2020, [cit. 2020-8-16]. Dostupné z: <https://www.tradingwithrayner.com/moving-average-indicator-trading-strategy/>.
9. **Moody, Chris.** Larry Connors RSI-2 Trading System. *Tradingview.* [Online] 10.6.2014. [cit. 2021-05-15]. Dostupné z: <https://www.tradingview.com/chart/CELG/EVXQPaR9-Larry-Connors-RSI-2-Trading-System-Surprising-Win-Rate/>.
10. **Groette, Oddmund.** Larry Connors' R3 Strategy (It Still Works) . *Quantified Strategies.* [Online] 12.5.2021. [cit. 2021-05-15]. Dostupné z: <https://www.quantifiedstrategies.com/larry-connors-r3-strategy/>.
11. **Marwood, Joe.** Testing The RSI 2 Trading Strategy On US Stocks. *Decoding Markets.* [Online] 20.1.2016. [cit. 2021-05-15]. Dostupné z: <https://decodingmarkets.com/rsi-2-trading-strategy/>.
12. **Bischoff, Bea.** What is the Adjusted Closing Price. *Finance Zacks.* [Online] 31.3.2019. cit. 2021-3-31. Dostupné z: <https://finance.zacks.com/adjusted-closing-price-vs-closing-price-9991.html>.

13. **Stanley, Brian.** A Primer on Survivorship Bias. *Quantrocket*. [Online] 19.11.2020. [cit. 2021-5-6]. Dostupné z: <https://www.quantrocket.com/blog/survivorship-bias/>.
14. **Riabitskyi, Daniil.** How to evaluate a trading strategy in the right way. *Collective 2 - Automate your trading*. [Online] 9.9.2019. [cit. 2021-05-8]. Dostupné z: <https://www.linkedin.com/pulse/how-evaluate-trading-strategy-right-way-lot-one-topic-riabitskyi/>.
15. **Brown, John.** Past Performance Is Not Indicative Of Future Results. *Forbes*. [Online] 29.9.2016 [cit. 2021-05-08]. Dostupné z: <https://www.forbes.com/sites/johnbrown/2016/09/29/past-performance-is-not-indicative-of-future-results/?sh=db4d1373bf5b>.
16. **Lock, Andrew.** *ASP.NET Core in Action*. New York : Manning Publications Co, 2018. ISBN: 9781617294617.
17. **Brind, Mike.** Learn Entity Framework Core. *learnenityframeworkcore*. [Online] 2021 [cit. 19.5.2021]. Dostupné z: <https://www.learnenityframeworkcore.com>.
18. **Evans, Eric.** *Domain-Driven Design, Tackling Complexity in the Heart of Software*. Boston : Addison-Wesley Professional, 2003. ISBN: 0-321-12521-5.
19. **Martin, Robert C.** The Clean Architecture. *Clean Coder*. [Online] 13.8.2012. [cit. 2021-05-22]. Dostupné z: <https://blog.cleancoder.com/>.
20. **Cesar de la Torre, Bill Wagner, Mike Rousos.** Architecture for Containerized .NET Applications 3.1 Edition. [Online] Microsoft, 2020. cit. [2021-5-11]. Dostupné z: <https://dotnet.microsoft.com/download/e-book/microservices-architecture/pdf>.
21. **Khorikov, Vladimir.** Value Objects explained. *Enterprise Craftsmanship*. [Online] 3.1.2015 [cit. 2021-5-30]. Dostupné z: <https://enterprisecraftsmanship.com/posts/value-objects-explained/>.

Zoznam použitých skratiek

Názov	Význam	Popis
C#	C sharp	Objektovo orientovaný programovací jazyk od spoločnosti Microsoft
.NET Core	Framework	Framework od spoločnosti Microsoft, ktorý sa používa na vývoj webových a desktopových aplikácií fungujúci na platformách Mac OS, Linux, Windows.
SQL	Structured Query Language	Jazyk používaný v relačných databázach na manipuláciu s dátami.
ORM	Object-relational mapping	Objektový prístup pre prácu s dátami. Stará sa o preklad objektového prístupu do jazyka databázy.
LINQ	Language Integrated Query	Dotazovací jazyk v .NET na získanie dát z rôznych zdrojov a formátov.
SMA	Simple Moving Average	Indikátor používaný na technickú analýzu. Poukazuje na skutočnosť čo sa na trhu stalo za posledných n dní.
MACD	Moving Average Convergence Divergence	Indikátor používaný na technickú analýzu. Vyjadruje vzťah medzi dvoma kĺzavými priemermi.
ROC	Rate of Change	Momentum Indikátor používaný na technickú analýzu. Meria rozdiel sily pohybu ceny za definované obdobie
RSI	Relative Strength Index	Momentum Indikátor používaný na technickú analýzu. Meria rozdiel sily pohybu ceny za definované obdobie
API	Application Programming Interface	Rozhranie ktoré sa používa pre komunikáciu s inými systémami.
SPY	Standard & Poor's 500	Najčastejšie sledovaný akciový index na svete. Obsahuje 500 najväčších amerických akciových firiem.
SP100	Standard & Poor's 100	Obsahuje 100 najväčších a najuznávanejších spoločností vybraných z indexu SPY.
QQQ	Nasdaq-100 Index	Akciový index v ktorom prevládajú najväčšie technologické firmy.

VTV	Vanguard Value Index	Široko diverzifikovaný akciový Index zameraný na hodnotové akcie.
DDD	Domain-Driven Design	Prístup k architektúre systému ktorý by mal svojím dizajnom, vyjadrovať navrhovanú doménu.
JSON	JavaScript Object Notation	Jazyk určený na prenos dát, nezávislý na platforme.
POCO	Plain Old Class Object	Jednoduchý objekt vytvorený v .NET. Tieto objekty neobsahujú žiadnu závislosť na externé knižnice alebo balíčky.

Zoznam príloh

Príloha A – USB

USB je priložené do vytlačenej verzie bakalárskej práce.

Obsahom USB je:

- TradeBash - adresár s aplikáciou TradeBash
 - časťou adresára je readme súbor popisujúci spustenie aplikácie
- Reports - adresár s vygenerovanými reportmi
- Data - adresár s indexami, použité pre výber akcií