Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Automatický obchodní systém

2020

Jan Kvapil

Vedoucí práce: Mgr. Petr Osička, Ph.D.

Studijní obor: Aplikovaná informatika, prezenční forma

Bibliografické údaje

Autor:	Jan Kvapil
Název práce:	Automatický obchodní systém
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2020
Studijní obor:	Aplikovaná informatika, prezenční forma
Vedoucí práce:	Mgr. Petr Osička, Ph.D.
Počet stran:	35
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Jan Kvapil
Title:	Algorithmic trader
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2020
Study field:	Applied Computer Science, full-time form
Supervisor:	Mgr. Petr Osička, Ph.D.
Page count:	35
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Aplikace pro sběr a analýzu obchodních dat, komunikující s platformu MetaTrader. Na základě obchodních dat automaticky vykonává obchodní poknyny. Součástí textu práce je popis základních pojmů, principů obchodování na Forexu a popis platformy MetaTrader z programátorského pohledu.

Synopsis

Application for collecting and analyzing trading data, communicating with the MetaTrader platform. Based on trading data, it automatically performs trading orders. Part of the text is a description of basic concepts, principles of trading on Forex and a description of the MetaTrader platform from a programming perspective.

Klíčová slova: Algoritmické obchodování; Forex; MetaTrader

Keywords: Algorithmic trading; Forex; MetaTrader

Děkuji Mgr. Petru Osičkovi, Ph.D. za ochotný přístup při vedení této práce a Mgr. Janu Třískovi, Ph.D za odborné konzultace k problematice algoritmického obchodování. Dále chci poděkovat své rodině za podporu po celou dobu studia.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Seznam vět

Seznam zdrojových kódů

1 Úvod

Cílem této bakalářské práce je vytvoření programu pro sběr a analýzu obchodních dat, jakožto plugin pro platformu MetaTrader, který na základě sesbíraných dat bude provádět automatické obchody. Součástí textu práce bude popis základních pojmů, principy obchodování na Forexu a popis platformy MetaTrader z programátorského pohledu.

1.1 Burza

Burza je instituce realizující trh s **instrumenty** (akcie, komodity, měnové páry). Burza v současnosti funguje jako datacentrum a zajišťuje celkovou organizaci obchodů s pomocí systému Matching Engine. Součástí systému Matching Engine je evidenční databáze obchodních příkazů (Central Order Book), s jejíž pomocí se automaticky vyhledávají a párují objednávky. Dalšími vlastnostmi burzy jsou anonymizace a eliminace kreditního rizika - garance, že se obchod řádně provede a dojde k požadované finanční transakci. **Forex** je globální decentralizovaný typ burzy, zaměřený na směnu měnových párů [6].

1.2 Broker

Broker je společnost s možností přímého přístupu na burzu. Retail investor (běžný obchodník bez makléřské licence) může obchodovat na burze pouze skrze brokera. Uživatel si založí u brokera účet, na který vloží svůj kapitál, se kterým chce obchodovat. Většina brokerů nabízí možnost i tzv. demo účtů, kde si uživatel zvolí libovolný fiktivní kapitál, se kterým může simulovat obchodování. Broker si (v případě obchodování na reálném účtu) může brát za každý provedený obchodní pokyn určitou provizi v podobě spreadu nebo předem stanoveného poplatku.

1.2.1 Market Maker

Market Maker (neboli tvůrce trhu) vytváří trh pro své klienty, nezávisle na burze. Sám si určuje nákupní a prodejní cenu [2]. Market Maker tedy automaticky tvoří protistranu a je důležitý zejména pro realizaci obchodů v případě instrumentů s nízkou likviditou [6].

1.2.2 STP

STP (Straight Through Processing) brokeři posílají objednávky klientů přímo poskytovatelům likvidity - bankám a jiným finančním institucím. Broker by měl vždy vybírat toho nejlepšího poskytovatele likvidity, který je v daném okamžiku k dispozici, a spárovat přes něj objednávky [2].

1.2.3 ECN

ECN (Electronic Communication Network) brokeři provozují tzv. likvidní koš, ve kterém se účastní obchodování všichni klienti (retailoví i institucionální) spolu se všemi poskytovateli likvidity (jako v případě STP). Pokud se jedna ze stran chystá koupit určité množství daného instrumentu a druhá strana se chystá ve stejnou dobu prodávat, jejich objednávky jsou okamžitě vypořádány uvnitř ECN likvidního koše [2].

1.3 Základní pojmy

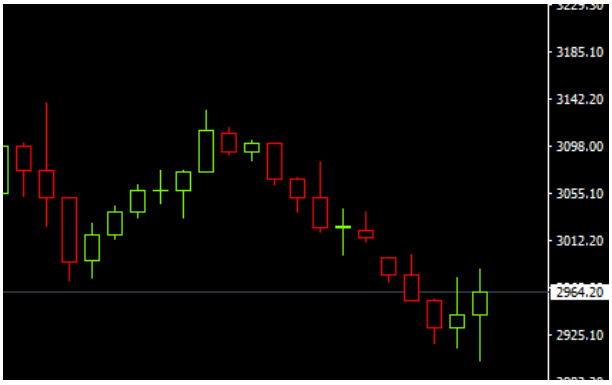
- **Cena** instrumentu v daném čase je dvojice - nabídka (bid) a poptávka (ask). Rozdíl mezi nabídkou a poptávkou se nazývá **spread**.
- **Objem** je velikost obchodované transakce. Základní objemovou jednotkou transakce je 1 Lot, neboli transakce o hodnotě 100 000 jednotek obchodované měny [4]. Někteří brokeři umožňují otevírat pozice o velikosti mikrolotu, neboli 0.01 lotu.
- **Volatilita** - míra kolísání hodnoty aktiva (obchodovaného instrumentu) nebo jeho výnosové míry (obvykle jako směrodatnou odchylku těchto změn během určitého časového úseku). Jedná se o nástroj, pomocí kterého lze předpokládat potenciální nárůst či pokles hodnoty aktiva v budoucnosti na základě změn hodnot tohoto aktiva v minulosti [5].
- **Likvidita** - schopnost rychle přeměnit investiční portfolio na hotovost, aniž by došlo k výrazné ztrátě jeho hodnoty [2]. V případě Forexu lze jednoduše říct, že jde o celkový zobchodovatelný objem daného instrumentu v konkrétním čase.

1.4 Pohyb ceny

Nejmenší cenový pohyb, o který může cena instrumentu vzrůst (klesnout), se označuje jako **pip** (Percentage in Point). V případě některých měnových párů je hodnota pipu určena ještě přesněji - v bodech, kdy deset bodů odpovídá jednomu pipu. U ostatních instrumentů je pip stejný jako bod, tj. minimální hodnota pohybu ceny instrumentu [4].

1.5 Svíčkový graf

Základním znázorněním historie pohybu ceny je svíčkový graf. Jednotlivé svíčky představují **časová období** (M1 - minutová, M5, M15, H1, ...), tělo svíčky pak zobrazuje rozdíl mezi otevírací a uzavírací cenou. Součástí svíčky mohou být také knoty, které zobrazují nejvyšší a nejnižší hodnotu nabídky a poptávky, za kterou v daném časovém rámci byl realizován nějaký obchod.



Obrázek 1: Svíčkový graf

1.6 Typy obchodování

Obchodování na burze se dá obecně rozdělit do těchto dvou kategorií - spekulace a investování. Spekulace se vyznačuje obchodováním v kratších časových rámcích. Investování, nebo též poziční obchodování, je naopak zaměřeno na dlouhodobé držení otevřených pozic. Můžeme se také setkat s pojmy jako scalping, intradenní a nebo swingové obchodování. Jedná se o typy spekulativních obchodních stylů, seřazeny od nejkratších po delší časová období, ve kterých se tyto styly používají.

1.7 Obchodní pokyny

Mezi obchodní pokyny patří nákup a prodej daného instrumentu, které se odesílají brokerovi (případně přímo burze). Pokyn lze realizovat za tržní cenu nebo lze nastavit tzv. čekající příkaz, který se vykoná v případě, že cena dosáhne požadované hodnoty. Mezi další nastavení patří velikost obchodovaného objemu a hodnoty **Stop Loss** (SL) a **Take Profit** (TP), které omezují případný zisk nebo ztrátu. Dále lze u otevřených pozic například upravit tyto hodnoty SL/TP nebo (v případě čekajících příkazů) také úplně zrušit objednávky.

1.8 Pákový efekt

Někteří brokeři umožňují využít také pákového efektu, díky kterému lze manipulovat s několikanásobně větším objemem než je velikost vlastního portfolia. Typicky jde o poměry řádově 1:10, ale lze najít i brokery, kteří umožňují obchodovat s pákou 1:500 a výš. Bereme-li v úvahu, že je v dnešní době možné otevřít si obchodní účet o velikosti několika set dolarů, stává se obchodování na burze dostupné téměř pro každého. Je ale nutné zmínit, že právě tato dostupnost je snadnou cestou k rychlé ztrátě kapitálu pro většinu začínajících obchodníků.

1.9 Marže

Při využití pákového efektu je nutné mít zálohu na otevřenou pozici - marži. Jde o část kapitálu, která je blokována během doby, kdy je pozice otevřena. Výška marže závisí na celkovém objemu otevřených pozic a velikosti páky.

1.10 Technická a fundamentální analýza

Obchodníci mohou zadávat obchodní pokyny dle svých vlastních strategií. Mimo jiné mohou vycházet z technické nebo fundamentální analýzy. Technická analýza je zaměřena na předešlý vývoj ceny daného instrumentu (případně korelace ceny jiných instrumentů), zobchodované objemy a specifické svíčkové formace. Fundamentální analýza naopak vychází z ekonomického a politického dění, které se pak odráží na ceně instrumentů (akcie společnosti stouply po zveřejnění svého produktu).

1.11 Indikátory

Základním nástrojem technické analýzy jsou indikátory. Jde o statistické metody, vycházející z určitého časového rámce a vývoje ceny daného instrumentu. Pomocí indikátorů se pak definují obchodní strategie. Pro zjednodušení můžeme uvažovat cenu pro technické indikátory jako průměr hodnot nabídky a poptávky. Na příkladu si můžeme ukázat výpočet jednoduchého klouzavého průměru, který lze vyjádřit tímto vzorcem (obrázek 2).

$$MA = \frac{P_1 + P_2 + ... + P_n}{n}, \text{kde}$$

P_n je cena v čase n,

n je počet period

Obrázek 2: Indikátor Moving Average

1.12 Algoritmické obchodování

Jedná se o způsob obchodování, řízený automatickým obchodním systémem. Algoritmické obchodování funguje v mnoha úrovních, lišící se zejména ve velikostech obchodovaných objemů nebo v rychlosti vykonávání obchodních příkazů. V případě Market Makerů, kteří se snaží v krátkých časových úsecích reagovat na požadavky trhu, se bavíme o tzv. automatickém high-frequency tradingu [6]. Na druhou stranu je běžné, že i individuální obchodníci realizují své strategie v podobě automatizovaných obchodních robotů.

1.13 Obchodní strategie

Strategie je (z našeho algoritmického pohledu) jednoznačně definovaný předpis, který říká, kdy se mají odeslat obchodní pokyny, s jakými parametry a s jakým rizikem. Základní obchodní strategie většinou vychází z předem definovaných indikátorů (například v rámci platformy MetaTrader). Indikátory lze různě kombinovat, nastavovat jejich parametry, odchylky a vytvářet z nich komplexní obchodní schémata.

2 Architektura aplikace

2.1 Automatický obchodní systém

Jedná se o programový celek, který má provést uživatele od definice strategie až po její spuštění a automatickou správu. V následujících několika bodech si definujeme, co by měla aplikace obsahovat a jakým způsobem by měla fungovat.

- Aplikace je rozdělena do dvou základních částí - klient a server
- Serverová část z velké části vychází z open-source implementace od Darwinex [8] v jazyce MQL [22]. Spouští se jako Expert-Advisor v MetaTraderu [23] (upřesníme v následujících kapitolách) a vykonává obchodní pokyny, zasílané klientem. Komunikace s klientem probíhá pomocí protokolu TCP s využitím knihovny ZeroMQ [10].
- Klientská část obsahuje uživatelské rozhraní pro připojení k severu, komponenty pro definici indikátorů, strategií a dále komponentu pro vizualizaci probíhajícího obchodování (aktuální ceny, stavu strategie a otevřených pozic)
- Uživatel má možnost:
  - zvolit instrument, který chce obchodovat
  - zvolit časové období, ve kterém se bude obchodovat
  - nastavit hodnoty SL/TP
  - definovat vlastní indikátory
  - definovat obchodní strategie pomocí definovaných indikátorů
  - perzistentně uložit strategii a znova ji načíst

2.2 Správa strategií

Při implementaci bylo testováno několik přístupů ke správě obchodujících strategií. Prvním přístupem bylo uzavírání pozic určitou entitou (Strategy Manager),

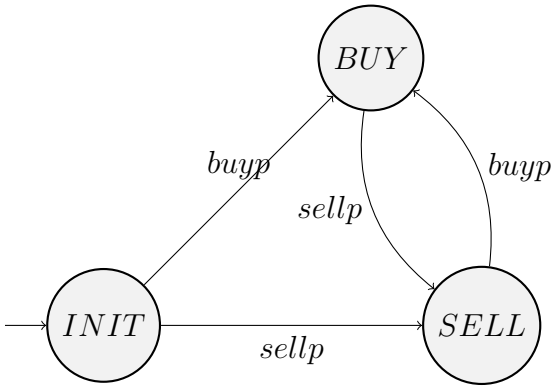
která by také řešila vyhodnocování efektivity strategií a omezování maximálního počtu souběžně otevřených pozic. Nevýhodou v tomto přístupu byla určitá nepředvídatelnost a chybovost v závislosti na čase. Docházelo například k situacím, kdy se pozice kvůli zpoždění uzavřela za jinou cenu nebo se kvůli chybě v komunikaci (výpadek internetového spojení) neuzavřela vůbec.

2.3 Uzavírání pozic

Tento problém byl však jednoduše vyřešen. Uzavírání pozic se vykonává pomocí předem nastavených hodnot SL/TP. Tím zajistíme, že se pozice vždy uzavře za požadovanou částku (samozřejmě tedy pouze pokud je dostupná protistrana, která je ochotna za danou cenu pozici zobchodovat).

2.4 Strategie jako stavový automat

Cílem je formalizovat pojem strategie jako určitý teoretický model [1]. K tomu může posloužit stavový automat. V našem případě lze stavy považovat za fáze strategie. Ta má také nějaký počáteční stav, kdy ještě neproběhl žádný obchod a dále fáze, kdy se buď nakupuje, a nebo prodává.



Obrázek 3: Strategie

Máme k dispozici dva základní modely automatických obchodních systémů, které se liší počtem stavů a zejména svým chováním při obchodování. V prvním modelu se obchodní pokyn odesílá pouze při přechodu z jednoho stavu do druhého (tento model je implementován). V druhém modelu se obchoduje v předem definovaných intervalech, kdy stav určuje pouze typ obchodních pokynů, které se odesílají (buy/sell). V druhém případě je tedy možné otevřít více pozic stejného typu po sobě. Dále potřebujeme množinu přechodových funkcí (sellp, buyp), které nám říkají, za jakých podmínek má dojít ke změně stavu. K tomu můžeme použít například následující ukázkový kód č. 1 (vycházející z finální implementace přechodových funkcí).

```
1  // SELL predicate
2   price < indicators.get("ma100")
3  // BUY predicate
4   price >= indicators.get("ma100")
```

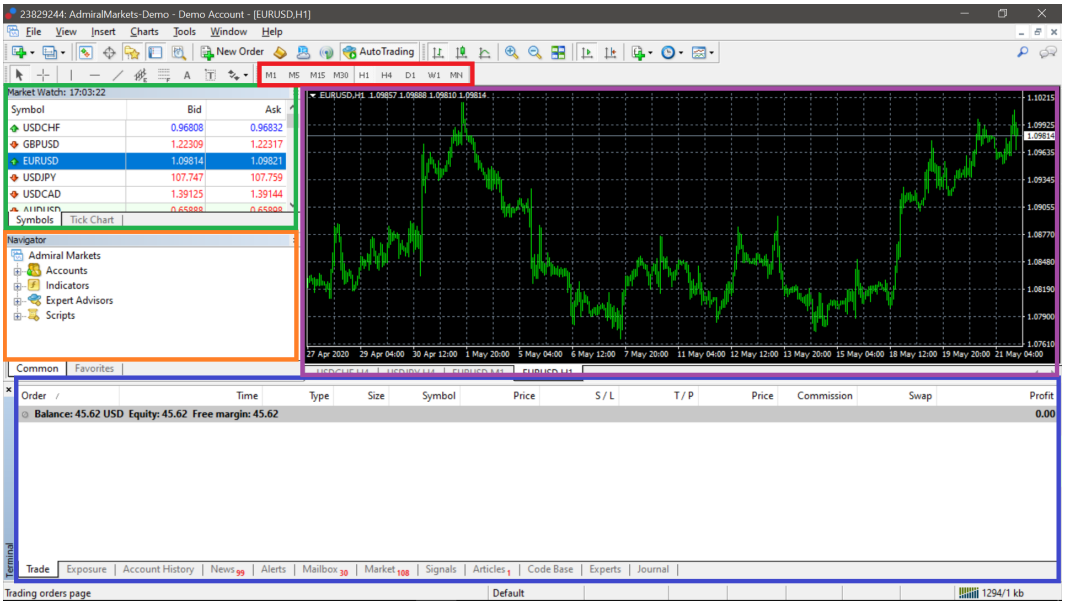
Zdrojový kód 1: Definice přechodových funkcí

3 Platforma MetaTrader

Platforma MetaTrader je jedna z nejdostupnějších obchodních platforem pro obchodování na burze. Je vyvíjena společností MetaQuotes a není spojena s žádným konkrétním brokerem (někteří brokeři používají vlastní obchodní platformy jako např. TastyWorks). Skrze MetaTrader lze zadávat obchodní pokyny. Stačí pouze založit účet u brokera, který tuto platformu podporuje. Je třeba poznamenat, že platforma nabízí pouze rozhraní pro obchodování. Obchodovatelné instrumenty, poplatky a výši spreadu si vždy stanovuje už konkrétní broker sám.

3.1 Popis platformy

Zde jsou zobrazeny nejdůležitější části obchodní platformy MetaTrader 4, rozděleny do 5 rámečků. V následujících několika odstavcích si popíšeme jejich význam.



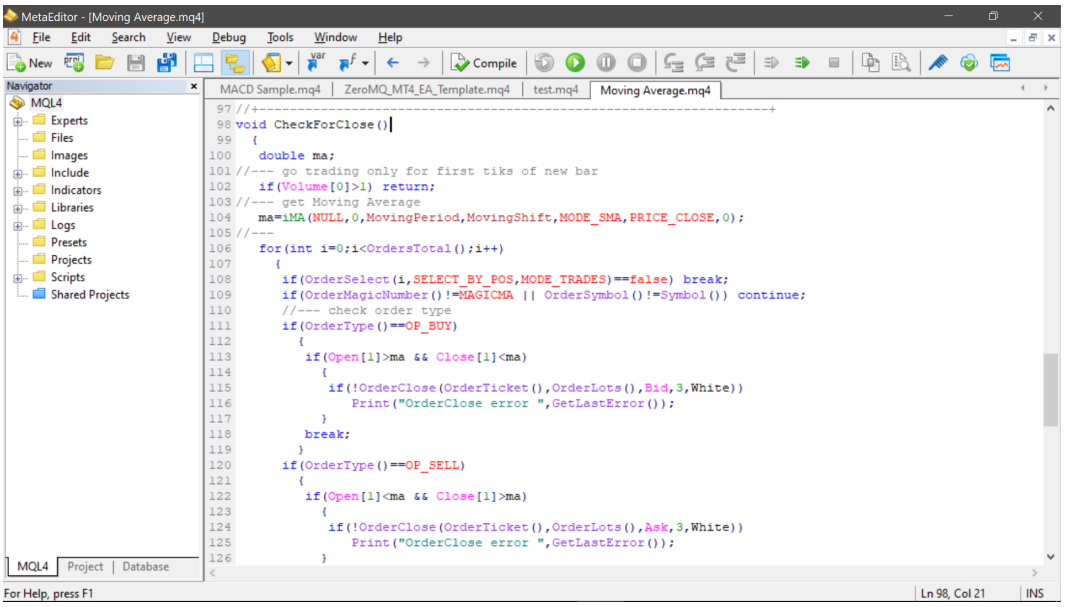
Obrázek 4: Popis platformy

- Nejvýše umístěným zvýrazněním je přepínač časových období grafu (červený rámeček).
- Ve fialovém rámečku je graf zvoleného instrumentu.

- V levém horním rohu (zelený rámeček) je seznam obchodovatelných instrumentů, rozdělených podle kategorií (CFD, Forex).
- Pod seznamem obchodovatelných instrumentů (oranžový rámeček) máme 4 složky - Accounts, Indicators, Expert Advisors a Scripts. Nás bude zajímat zejména složka Expert Advisors, ve které jsou umístěny automatické obchodní systémy a strategie. Obchodní strategie je vždy svázána s konkrétním instrumentem (grafem) a spouští se přetažením do grafu (případně kliknutím pravým tlačítkem myši a zvolením "Add to chart"). Aby mohla strategie obchodovat, je nutné ještě povolit automatické vykonávání obchodních příkazů.
- Poslední zvýrazněnou částí platformy je terminál (modrý rámeček). V první záložce (Trade) je seznam otevřených pozic a stav účtu. Další důležitou záložkou je Account History, kde najdeme seznam všech uzavřených pozic.

3.2 Programovací jazyk MQL4

Výhodou platformy MetaTrader je programovatelnost vlastních indikátorů a strategií. Ne každá obchodní platforma tuto možnost poskytuje. MetaTrader používá programovací jazyk MQL. Jedná se o nízkoúrovňový, staticky typový jazyk podobný například jazyku C++. Přestože MQL nabízí i vyšší úroveň abstrakce díky objektové orientaci, stále se jedná o jazyk, ve kterém je vývoj složitějších programů poměrně obtížný a zdlouhavý. Proto je jednou z hlavních motivací této bakalářské práce usnadnit vývoj strategií a používání automatických obchodních systémů díky možnosti implementace systému v rámci jiné technologie.



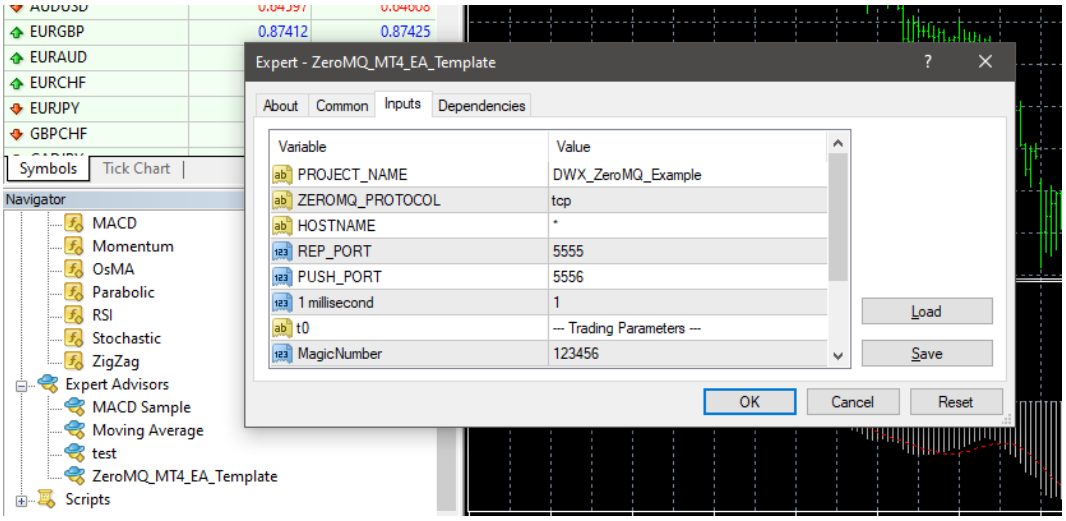
Obrázek 5: MetaEditor

3.3 MetaEditor

Jedná se o vývojové prostředí platformy MetaTrader. Obsahuje kompilátor .mq4 zdrojových souborů do spustitelných programů .ex4 (v MT4). V rámci platformy MetaTrader máme předdefinované ukázkové strategie - například implementaci strategie Moving Average, dostupnou ve složce Experts/Moving Average.mq4. Z tohoto příkladu je zřejmé, že pro realizaci i takto jednoduché strategie je nutné napsat poměrně velké množství kódu.

4 Popis serverové části

V této kapitole si stručně popíšeme Darwinex (dále DWX) implmenentaci serverové části, způsob programování automatických obchodních systémů a zároveň se zaměříme na některé vlastnosti jazyka MQL.



Obrázek 6: Spuštění EA Severu

```
1   extern int REP_PORT = 5555;
```

Zdrojový kód 2: Externí proměnná

Externí proměnné slouží k nastavování parametrů strategií nebo indikátorů při jejich inicializaci.

4.1 Základy MQL

Jazyk MQL poskytuje sadu základních funkcí, pomocí kterých reaguje na vnější události (změna ceny, tik timeru).

- **OnInit** - inicializace serverové části, nastavení timeru a TCP portů, přes které se bude komunikovat
- **OnTimer** - programová smyčka, ve které server například kontroluje, zda mu nepřišla nějaká zpráva
- **OnTick** - handler události, která se vyvolá při každém pohybu ceny (vhodné použít v případě, že chceme například streamovat cenové pohyby přes ZMQ publish socket)

4.2 DWX implementace

Jak bylo již zmíněno, serverová část vychází z open-source implementace od DWX a funguje tak, že na základě přijatých požadavků od klienta otevírá, modifikuje a uzavírá obchodní pozice. Zpátky posílá odpovědi ve formátu JSON (lehká změna oproti původní implementaci).

```
1   "TRADE|ACTION|TYPE|SYMBOL|PRICE|SL|TP|COMMENT|TICKET"
```

Zdrojový kód 3: Formát zprávy pro otevření obchodní pozice

4.2.1 Zpracování zpráv

- **MessageHandler** - funkce zpracuje příchozí zprávy od klienta, rozparsuje do jednotlivých řetězců a vloží do pole, se kterým se bude dále pracovat.
- **InterpretZmqMessage** - funkce nastaví požadovanou akci podle typu požadavku a následně zavolá příslušnou obsluhu dané akce (např. otevřít pozici, uzavřít všechny pozice atd.).

4.2.2 Obslužné funkce

V rámci obslužných funkcí se provádějí požadované obchodní operace. Zde si pro ukázku popíšeme jednu z nich - GetOpenOrders. Jedná se o funkci, která má vrátit seznam otevřených pozic. K tomu využívá zabudovaných MQL funkcí jako OrdersTotal, která vrací počet aktuálně otevřených pozic nebo OrderSelect, která vybere podle indexu (nebo tiketu) aktuální objednávku pro budoucí použití (implicitním uložením do nějaké globální proměnné).

4.2.3 Spuštění serveru

Pro komunikaci přes knihovnu ZeroMQ je potřeba použít několik externích .dll knihoven (celá instalace je popsána v [8]). Příloha k bakalářské práci také obsahuje zdrojový kód severové části (src/server/server.mq4). Tento soubor je nutné vložit do složky Experts v adresáři MQL4 a zkompilovat v MetaEditoru. Poté

se zkompilovaný soubor objeví v MetaTraderu, ve složce Expert Advisors. Přidáním tohoto souboru do libovolného grafu spustíme server a nastavíme porty, na kterých chceme komunikovat.

5 Použité technologie

Součástí této práce bylo mimo jiné projít si všemi fázemi vývoje aplikace - od abstraktního návrhu, vhodného výběru technologií, zprovoznění základní funkcionality až po dokončení reálně použitelného funkčního celku, včetně přívětivého uživatelského rozhraní. Pro účely této aplikace jsem se rozhodl použít javascriptové open-source technologie. Veškerý kód, psaný v rámci klientské části bakalářské práce, odpovídá specifikaci jazyka ECMAScript 6 [21].

5.1 Electron

Jedná se o populární framework pro tvorbu multiplatformních desktopových aplikací pomocí webových technologií [9]. Vývoj probíhá velmi podobným způsobem jako u webových aplikací. Proto je velice oblíbený a rozšířený nejen u startupů a malých projektů. Jsou na něm postaveny známé aplikace jako například Visual Studio Code nebo Slack. Na rozdíl od běžné webové stránky, načtené v prohlížeči, umožňuje Electron (díky NodeJS [11]) používat aplikační rozhraní operačního systému (přístup k souborovému systému atd.) - jako u serverových aplikací. Nevýhodou je však nutnost pro každou aplikaci distribuovat současně i "celé jádro prohlížeče". Nicméně v době psaní této bakalářské práce je to i přesto, že nejmenší aplikace může mít kolem 200MB, zanedbatelný fakt. Výhodou je pak jednoduchý přechod z vývoje webových aplikací k tvorbě pro desktop a snadná přenositelnost mezi platformami. V první verzi aplikace [19] byl použit Electron v "čisté" formě, bez jakýchkoliv dalších frontendových frameworků.

```
1  const Button = (props) => {
2    return (
3      <button onClick={props.onClick}>
4        { props.value }
5      </button>
6    )
7  }
```

Zdrojový kód 4: React komponenta

5.2 React

Javascriptová knihovna pro tvorbu uživatelského rozhraní [12], vyvíjena společností Facebook. Díky své jednoduchosti a funkcionálnímu přístupu je velice silným nástrojem pro tvorbu frontendu, který se skládá z nezávislých komponent.

React je také velmi přívětivým nástrojem i z vývojářského hlediska, díky sdruženým technologiím jako Hot-Reload [14] (změna UI bez nutnosti restartu aplikace při zachování aplikačního stavu) nebo Redux [13] (možnost logování všech změn aplikačního stavu od spuštění aplikace). React natolik ovlivnil svým přístupem vývoj aplikací, že začaly vznikat i další implementace jako React-Native (pro vývoj mobilních aplikací).

5.3 Proton-native

Za zmínku určitě stojí i tento open-source projekt [17], vytvořen uživatelem kusti8, který od verze 2.0 poskytuje rozhraní k "nativním" komponentám skrze framework Qt5. Jedná se o zajímavou multiplatformní implementaci React-Native pro desktop. Nicméně je bohužel implementovaných jen několik základních komponent a lze jej tedy použít pouze pro opravdu jednoduché aplikace. Tato technologie byla použita v druhé verzi bakalářské práce [20], ale právě kvůli nejistému a pomalému vývoji jsem byl nucen přejít na robustnější řešení s celkově lepší podporou.

5.4 Next-js

V poslední verzi aplikace jsem zvolil právě technologii Next-js [15] v kombinaci s Electronem. Next-js vychází z Reactu, ale navíc umožňuje server-side rendering a routing. Je velmi inspirován jazykem PHP, ale zachovává veškeré principy Reactu. Pro tvorbu desktopových aplikací je už předpřipravený boilerplate s názvem Nextron [16]. Ten obsahuje i veškeré nastavení a nástroje pro generování optimalizovaného produkčního buildu.

```
1  const [timeframe, setTimeframe] = useState(10)
2
3  const handleTimeframeChange = (e) => {
4    const value = parseInt(e.target.value)
5
6    if (isNaN(value) || value < 1) {
7      setTimeframe(1)
8    } else setTimeframe(value)
9   }
10
11   ...
12
13  <input
14    value={timeframe}
15    onChange={handleTimeframeChange}
16  />
```

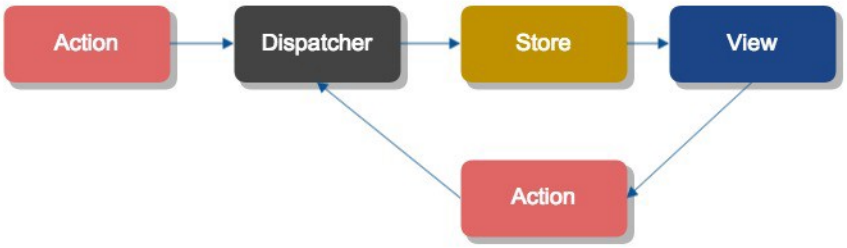
Zdrojový kód 5: Stav komponenty

5.5 Aplikační stav

Funkcionální komponenty si udržují vlastní stav pomocí tzv. hooků. Při každé změně stavu dojde k automatickému překreslení komponenty. Jak lze vyčíst z ukázek zdrojového kódu č.4 a č.5, je-li třeba předat komponentě nějaká data (případně funkce), využije se vlastností komponenty (props).

5.5.1 Flux

Jak bylo již dříve zmíněno, v Reactu se typicky řeší práce s aplikačním stavem použitím knihovny Redux. Jedná se o implementaci architektury Flux [7], která odděluje stav komponent do jednoho místa v aplikaci (store). Stav se poté mění voláním akcí. To řeší problém s komplexitou případů, kdy se stav mění z více míst aplikace, a nebo se změna projevuje ve více pohledech (na rozdíl od MVC přístupu).



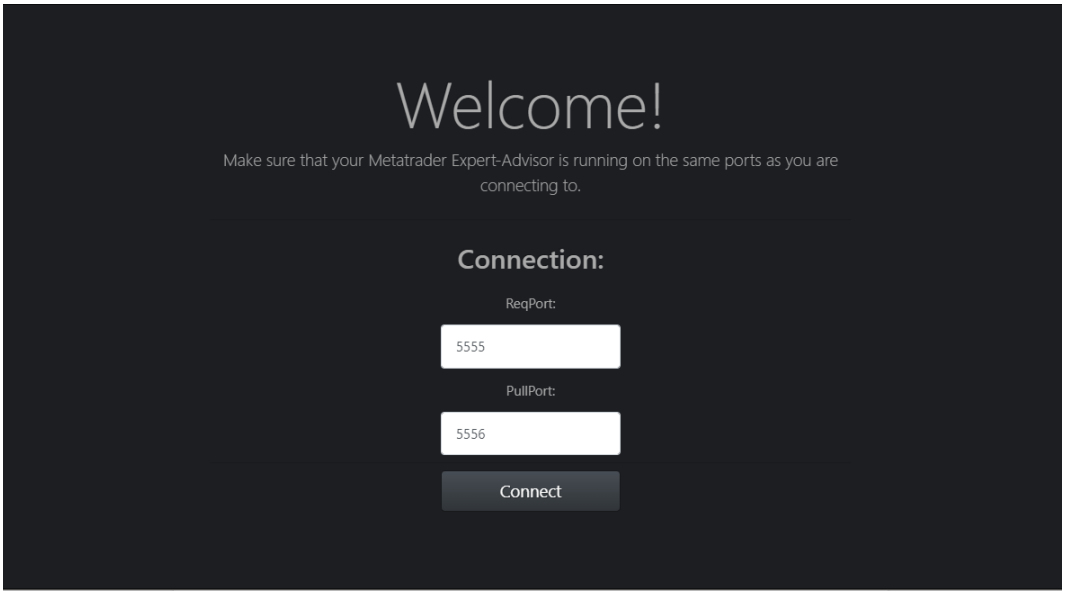
Obrázek 7: Flux: Data flow diagram [7]

5.5.2 Use Global Hook

V jednoduchých aplikacích, kde není třeba využívat všech vlastností Reduxu, je však jeho použití poněkud zbytečně komplikované. Proto je vhodnější použít nějaký jednodušší přístup pro správu aplikačního stavu, jako v případě knihovny use-global-hook [18]. Stále platí, že se veškeré změny aplikačního stavu provádějí na jediném místě aplikace.

5.6 ZeroMQ

Komunikace mezi serverovou a klientskou částí probíhá zasíláním zpráv pomocí knihovny ZeroMQ [10]. Jedná se o univerzální open-source knihovnu pro asynchronní komunikaci s využitím libovolných programovacích jazyků i komunikačních protokolů. V této aplikaci budeme používat komunikační schémata push/-pull (neblokující) a request/reply (blokující).



Obrázek 8: Spuštění aplikace

6 Popis klientské části

6.1 Připojení k serveru

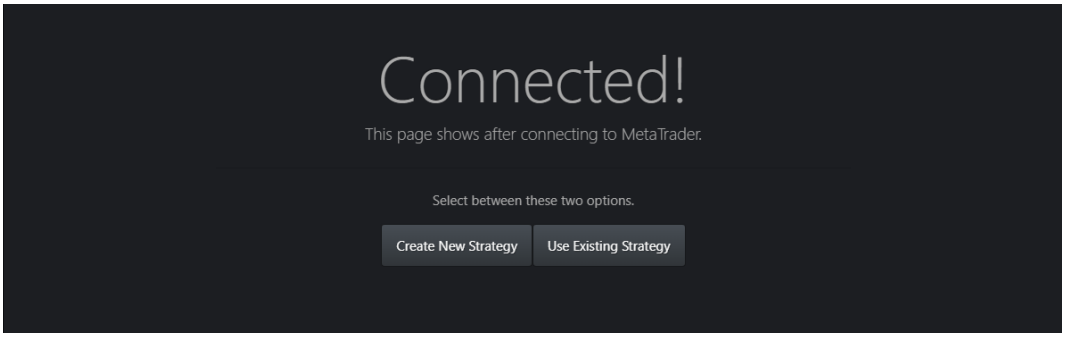
Prvním krokem ke zprovoznění aplikace je spuštění server v MetaTraderu (jak bylo popsáno v kapitole 4). Nyní máme spuštěný server na portech 5555 a 5556, který bude vykonávat příkazy od klienta. Po spuštění aplikace uvidíme následující okno (obrázek 8), kde nastavíme příslušné porty a připojíme se. Dále máme možnost zvolit si, zda chceme vytvořit novou strategii, nebo použít existující.

6.1.1 ConnectionForm

Tato komponenta poskytuje uživatelské rozhraní pro připojení k serverové části. Všechny komponenty mají defaultně nastavený svůj počáteční stav, mají definované handlery pro změnu stavu a další funkce pro ošetření uživatelských vstupů.

6.1.2 Třída Client

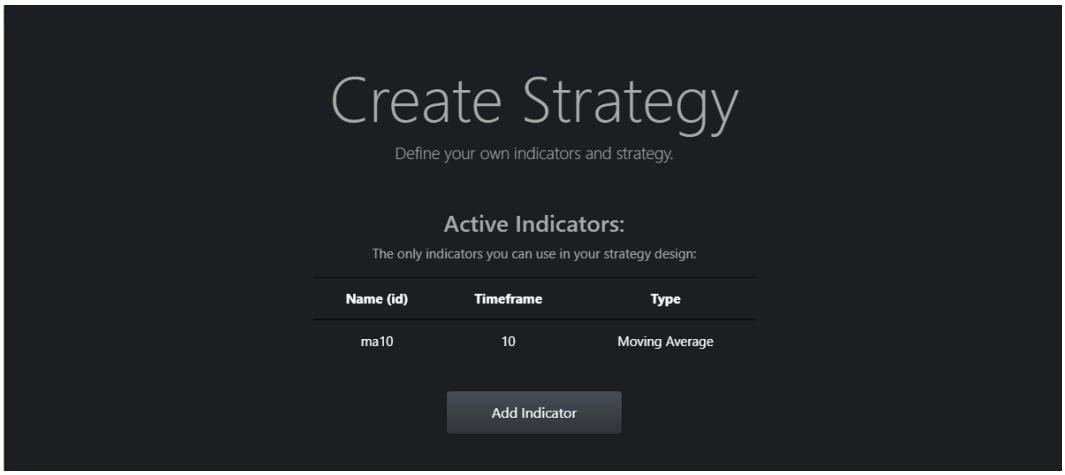
Připojení k Metatrader serveru vytvoříme pomocí instance třídy Client. Pokyny serveru se posílají metodou sendMsg přes REQ socket. Na PULL socketu máme nastavený “on message” handler, zpracovávající příchozí odpovědi. Rozlišujeme dva typy zpráv - Action a Rates. V prvním případě získáváme pole aktuálně otevřených pozic, které se pokaždé aktualizuje. V druhém případě přidáváme do bufferu aktuální hodnoty ceny.



Obrázek 9: Po připojení k MetaTraderu

6.2 Tvorba strategie

Základními objekty technické analýzy, pomocí kterých jsou definovány naše strategie, jsou indikátory. Při vytváření strategie je zapotřebí definovat vlastní indikátory. Přidané indikátory jsou znázorněny v tabulce Active Indicators (obrázek 10). Žádné jiné indikátory ve strategii použít nelze.

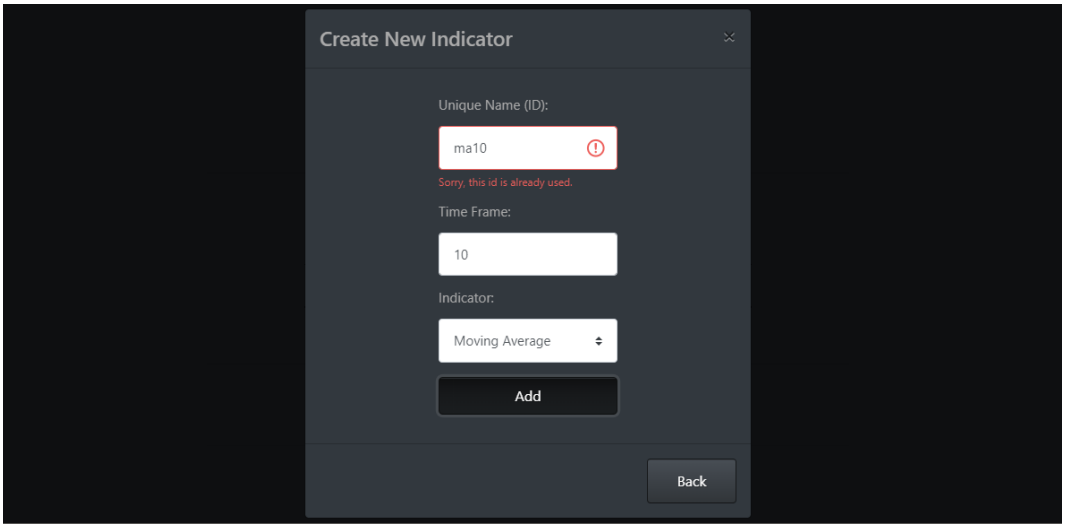


Obrázek 10: Tvorba strategie

6.2.1 Přidání indikátoru

Je nutné, aby každý nově definovaný indikátor měl unikátní id. V opačném případě je uživatel požádán, aby zvolil jiný název (id). Vlastnost timeframe určuje, z kolika posledních načtených hodnot ceny se bude hodnota daného indikátoru vypočítávat. Poslední nastavitelnou hodnotou je typ indikátoru.

Komponenta IndicatorAddForm nám umožňuje přidat uživatelsky definované indikátory. Indikátory se ukládají pomocí globálních akcí do globálního app-storu (s využitím knihovny use-global-hook). Poté se buďto uloží spolu se strategií do souboru, případně se (inicializované) ihned použijí při spuštění strategie. Pokud se uživatel rozhodne vrátit zpět a pokračovat ve výběru dříve definovaných



Obrázek 11: IndicatorAddForm: Komponenta pro definice indikátorů

strategií nebo tvorbou nové strategie, dříve nadefinované indikátory se zahodí. Indikátory jsou definovány následujícím způsobem - jak lze vyčíst ze zdrojového kódu č. 6, vrací se předpřipravená funkce, která si v lokálním prostředí uchovává vazbu n (požadovaná velikost pole) a ve chvíli, kdy je volána s argumentem db (hodnoty cen za posledních n časových jednotek), je vrácena hodnota indikátoru. Tímto způsobem si lze jednoduše definovat libovolné vlastní indikátory.

```
1 exports.average = n => {
2   return db => {
3     if (db.length != n)
4       throw new Error('Wrong size of array.')
5
6     return db.reduce((a, b) => a + b, 0) / n
7   }
8 }
```

Zdrojový kód 6: Indikátor typu Moving Average

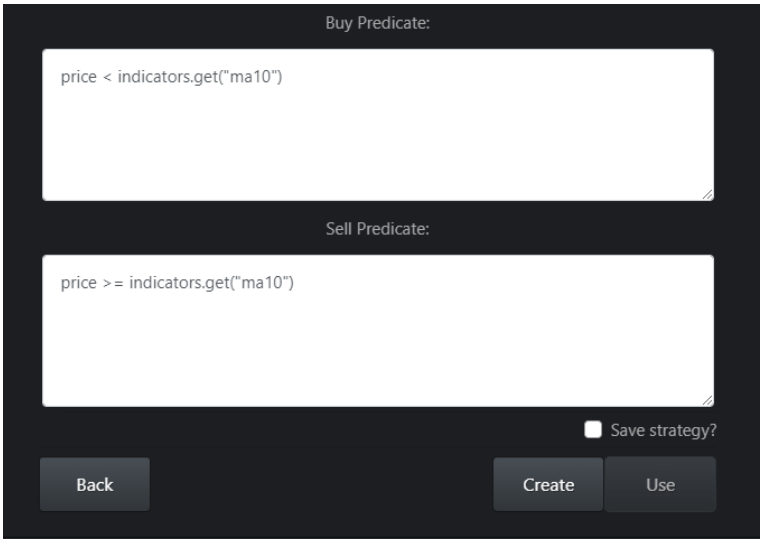
6.2.2 Definice strategie

Podobně jako u indikátorů i strategie mají svůj unikátní název. Dále mají nastavitelné vlastnosti otevíraných pozic - jako SL/TP a velikost obchodovaného objemu (lot size).

Komponenta StrategyAddForm poskytuje uživatelské rozhraní pro samotnou definici strategie. Komponenta obsahuje také funkce pro uložení strategie do souboru. Jelikož jsou strategie definovány jako stavové automaty, definujeme jejich chování pomocí predikátů tímto způsobem - obrázek 13. Jedná se o javascriptové výrazy, které se vyhodnotí na true/false. Tímto způsobem můžeme jednoduše

Obrázek 12: Definice strategie

definovat libovolné strategie zřetězením definovaných predikátů, vycházejících z hodnot indikátorů a aktuálních hodnot ceny v daném čase. V případě, že uživatel zadá neplatný javascriptový výraz, je upozorněn chybovou hláškou (stejně jako v případě již existujícího názvu indikátoru).



Obrázek 13: Definice přechodových funkcí

Zde je třeba upozornit na jeden implementační detail - definici samotných strategií. Jelikož se jedná o javascriptový kód, který není nijak analyzován, může jít o potencionální bezpečnostní riziko. Pokud by se zadal výraz, který by se následně vyhodnotil například na objekt, pole nebo cokoliv jiného než false, undefined nebo null, docházelo by k přechodu mezi stavy pokaždé, když by došlo k vyhodnocení tohoto výrazu (jelikož se jedná vždy o zobecněnou pravdivostní

hodnotu). V opačném případě by zase k přechodu mezi stavy nedošlo vůbec. V nejhorším scénáři by tedy mohlo dojít k pozměnění uživatelsky definovaných strategií přímo v souboru, ve kterém jsou uloženy. Ale jelikož se uživateli vždy zobrazuje definice strategie před samotným spuštěním, bylo by zřejmé, že s kódem není něco v pořádku. Hlavním bezpečnostním opatřením je ale zejména prvotní otestování strategie na demo účtu. V takovém případě by strategii na základě špatných výsledků vůbec nepoužil.

6.2.3 Třída Strategy

Přechody mezi stavy jsou vyvolány skrze metodu updateState. Z kódu č. 7 lze vyčíst, že změna stavu proběhne pouze v případě, že se vyhodnotí predikát následujícího přechodového stavu na true, a při téže příležitosti se aplikuje funkce spojená s aktuálním stavem. Pokud dojde ke změně stavu, vrací se hodnota aktuálního stavu.

```
1  updateState(price, indicatorsValuesMap) {
2    for (const t of this.transitions) {
3      if (this.state.name == t.name) {
4        for (const ns of t.nextStates) {
5          if (ns.name == this.state.name) continue;
6          if (ns.predicate(price, indicatorsValuesMap)) {
7            this.state = this.states.find(obj => {
8              return obj.name == ns.name
9            })
10           this.state.applyTransition()
11           return this.state
12         }
13       }
14     }
15   }
16   return false
17 }
```

Zdrojový kód 7: Aktualizace stavu strategie

6.2.4 Třída State

Jednoduchá třída reprezentující stav strategie. Obsahuje pouze svůj název a funkci, která se má při přechodu aplikovat (odeslání obchodního pokynu).

6.2.5 Třída Orders

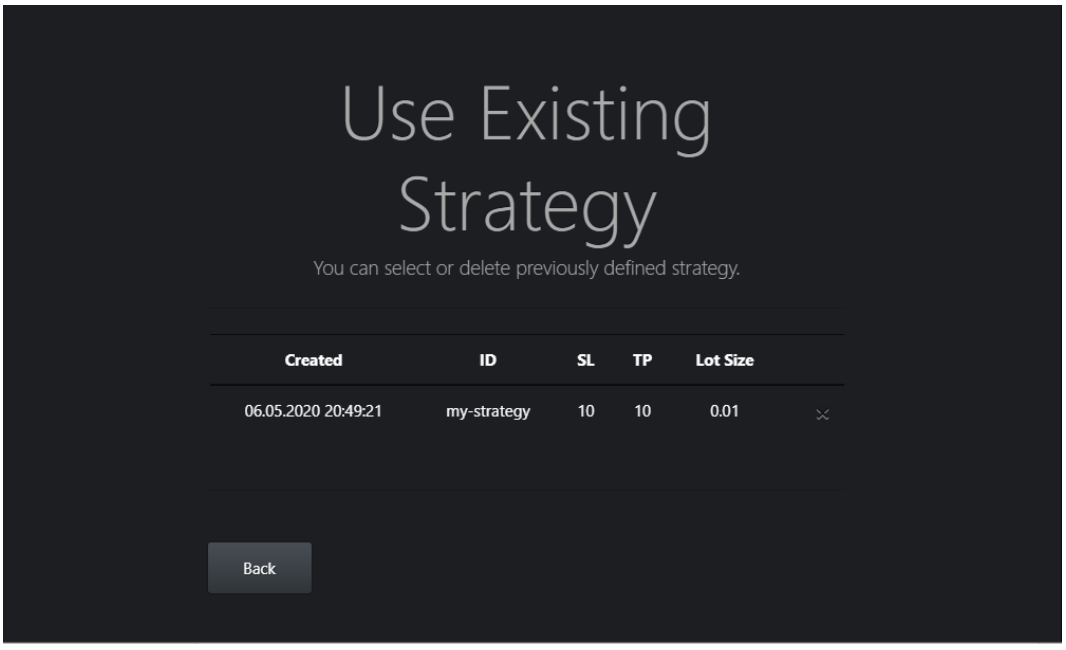
Množina pomocných funkcí, provádějících obchodní pokyny.

```
{
  name: "INIT",
  nextStates: [
  {
    name: "SELL",
    predicate: (price, indicators) => price < indicators.get("ma100")
  },
  {
    name: "BUY",
    predicate: (price, indicators) => price >= indicators.get("ma100"
        )
  }]
}, ...
```

Zdrojový kód 8: Ukázka možných přechodů pro stav init

```
exports.getOpenedTrades =
  client => {
    let req = "GET_OPEN_TRADES"
    client.sendMsg(req)
  }
```

Zdrojový kód 9: Dotaz na aktuálně otevřené pozice



Obrázek 14: Výběr z uložených strategií

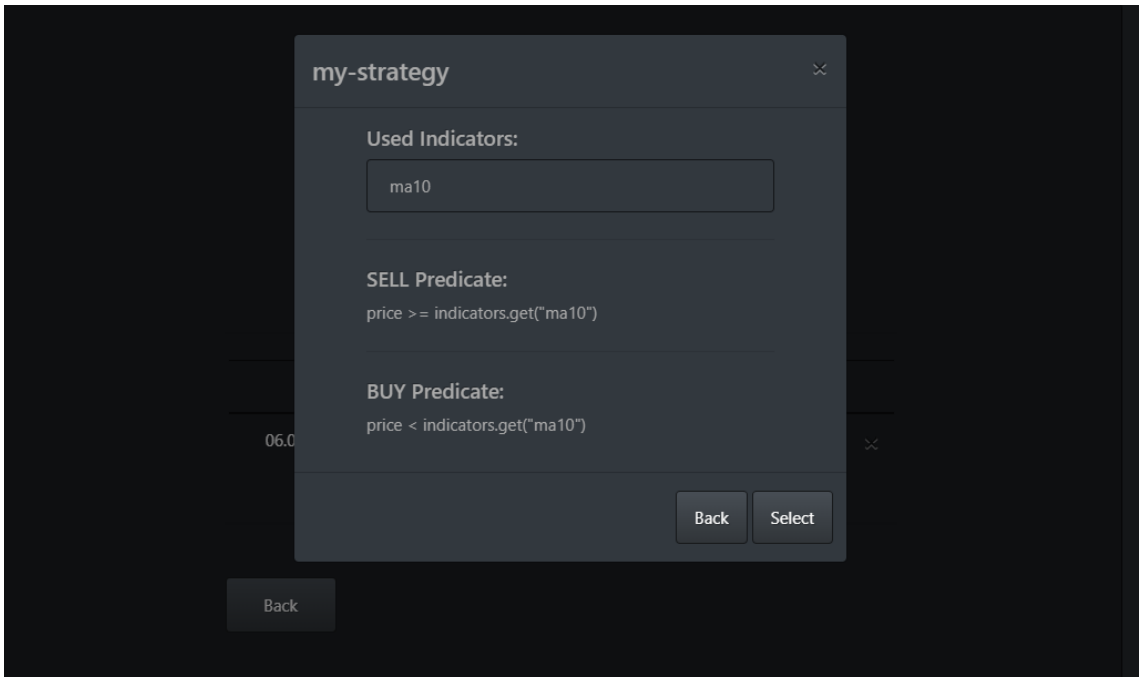
6.2.6 Vytvoření a uložení strategie

Definovanou strategii je možné perzistentně uložit. K tomu slouží check input v pravém dolním rohu. V případě, že nechceme ukládat, ale použít rovnou, lze pouze zakliknout Create a dále Use.

6.3 Načtení uložených strategií

Na stránce Use Existing Strategy (obrázek 14) vidíme dříve uložené strategie, které je možné ze seznamu vymazat nebo použít. Po rozkliknutí se zobrazí informace o použitých indikátorech spolu s definicí predikátů dané strategie (obrázek 15). Strategie jsou uloženy ve formátu JSON, takže se dají modifikovat a vytvářet i z "vnějšku" aplikace. Nicméně stále platí, že je třeba, aby se zachovala správnost javascriptové syntaxe.

6.4 Výběr a spuštění strategie

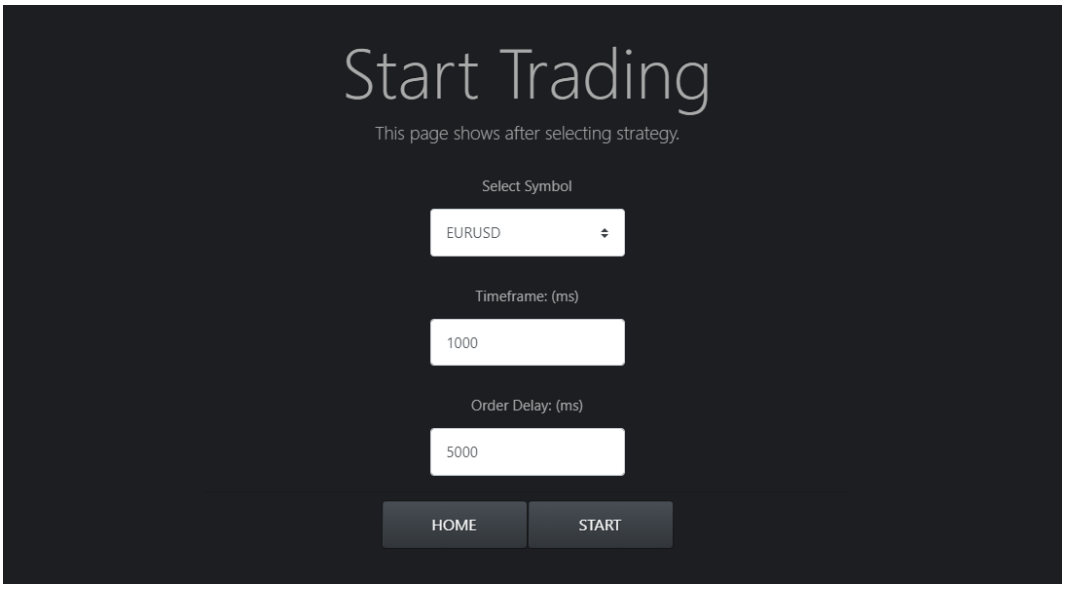


Obrázek 15: Výběr konkrétní strategie

Po kliknutí na tlačítko Select nás aplikace přesměruje na stránku Start Trading (obrázek 16). Zde máme možnost specifikovat konkrétní obchodovaný instrument (symbol) spolu s rychlostí komunikace mezi serverem a klientem (timeframe). Pro zajištění spolehlivé komunikace je vhodné používat interval větší než 500ms.

Klient posílá každých <timeframe> milisekund dotaz na server s požadavkem na získání aktuální hodnoty daného instrumentu a dále pole aktuálně otevřených

pozic. Jakmile mu server odpoví, předají se tyto informace třídě StrategyManager. Ta si aktualizuje aktivní indikátory a rozešle jejich nové hodnoty mezi strategie spolu s cenou. V aktuální verzi aktivně obchoduje pouze jedna strategie, nicméně aplikace je připravena na případné budoucí rozšíření pro souběžné obchodování více strategií. Poslední nastavitelná vlastnost Order Delay zamezuje situaci, ve které by se v případě rychlého zákmitu ceny (změny hodnoty) odeslalo několik obchodních pokynů v krátkém intervalu. Při odeslání obchodního pokynu se spustí timeout, který zamezí jakémukoliv obchodování po dobu určenou právě hodnotou Order Delay. Značná část případných komunikačních komplikací je už ošetřena díky knihovně ZeroMQ. Nicméně v případě, že by klient nebyl připojen, je uživateli zobrazena výzva pro opětovné připojení.



Obrázek 16: Specifikace parametrů pro obchodování

6.5 Automatické obchodování

Po spuštění strategie se zobrazí následující informace - aktuální hodnota daného instrumentu, aktuální stav strategie a seznam otevřených pozic (obrázek 17).

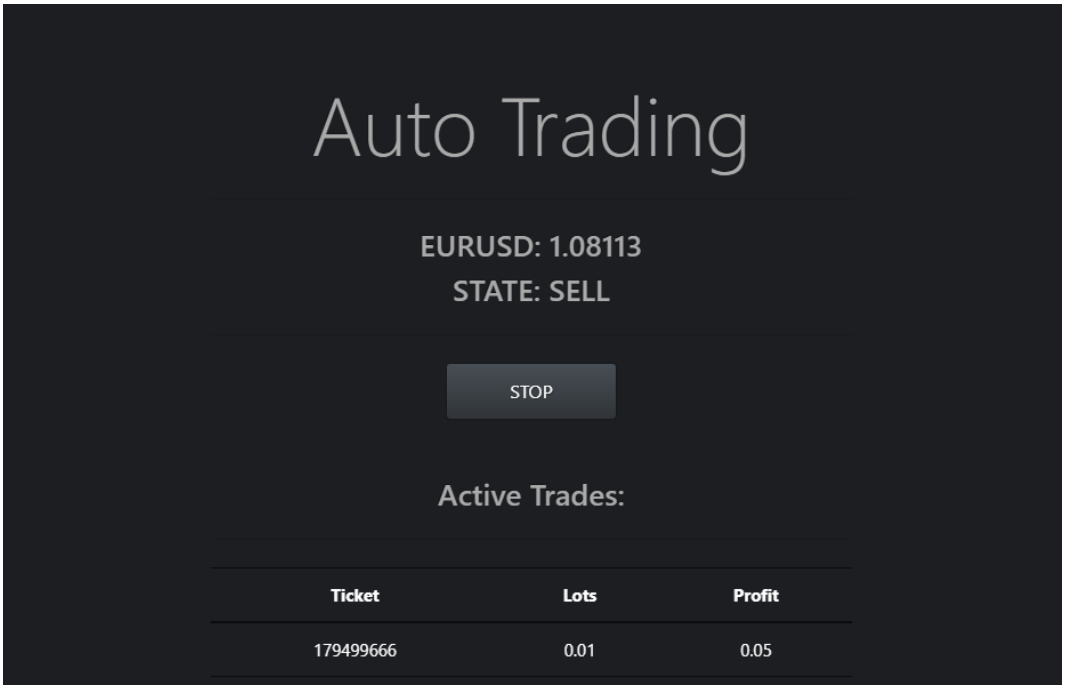
6.5.1 StrategyManager

Jedná se o třídu, která si udržuje seznam všech aktivních strategií spolu s mapou použitých indikátorů včetně jejich aktuálních hodnot. Dále třída obsahuje metodu pro přidání strategie a metodu sendEvent, která spočítá aktuální hodnoty indikátorů a rozešle je spolu s cenou mezi jednotlivé strategie (v současné verzi pouze jedna aktivní strategie). Tato třída prošla řadou změn, zejména v přístupu k vykonávání obchodních pokynů a vyhodnocování efektivity strategií. V prvotních verzích aplikace se efektivita strategií vyhodnocovala procházením

```
this.strategies.forEach(s => {
 const tempMap = new Map()
 for (const i of s.getIndicators().map(i => i.name)) {
  const iVal = this.indicatorValues.get(i)
  if (!iVal) return
  tempMap.set(i, iVal)
 }
 const changedState = s.updateState(lastPrice, tempMap)
 if (changedState)
  result = Object.assign({}, changedState)
})
```

Zdrojový kód 10: StrategyManager: aktualizace strategií

aktuálně otevřených pozic a jejich následném uzavírání při dosažení požadovaných hodnot SL/TP. To umožňovalo za běhu upravovat tyto hodnoty, počítat celkovou efektivitu a na základě této efektivity například zvyšovat nebo snižovat obchodované objemy. To se však ukázalo při praktickém použití jako nerealizovatelné. Docházelo ke zpoždění při komunikaci, s čímž souviselo například chybné počítání efektivity uzavřených pozic.



Obrázek 17: Průběh automatického obchodování

6.5.2 MainLoop

Main loop poskytuje rozhraní mezi frontendem a backendem, kde se předávají všechny uživatelsky nastavitelné hodnoty, potřebné k realizaci automatického

obchodování a funkce pro aktualizaci GUI. Provádí se zde několik nastavení - inicializace uživatelsky definovaných indikátorů a strategií, předání strategií třídě Strategy Manager, nastavení obchodovaných instrumentů atd. Poté se nastaví interval, během kterého se budou posílat pokyny serveru (získání otevřených pozic, aktuální ceny). Ve smyčce se také musí volat GUI funkce pro aktualizaci frontendu. Odkaz na spuštěný interval je předán frontendu, ze kterého je možné probíhající výpočet ukončit.

7 Shrnutí

Aplikace je od začátku koncipována jako jednoduše rozšiřitelný open-source projekt. To znamená, že je například velmi jednoduché definovat si vlastní typy indikátorů a použít je v rámci definice vlastních strategií.

V dalších verzích aplikace se počítá s paralelním během více strategií najednou, možnostmi použití odlišných přístupů pro komunikaci s MetaTraderem (streamování aktuálních hodnot cen), pro snadné backtestování na historických datech. Dále je v plánu rozšíření uživatelské definice strategií spolu s vyhodnocováním efektivity na základě přijatých informacích o uzavřených pozicích přímo od MetaTraderu.

Jelikož je aplikace stavěna jako multiplatformní, budou k dispozici v příloze bakalářské práce verze pro Windows (10) a Linux (Ubuntu 18.04). Je třeba upozornit, že samotná platforma MetaTrader podporuje pouze operační systém Windows. Na Linuxu a MacOS lze spustit MetaTrader pouze pomocí programů typu Wine.

Závěr

Cílem této bakalářské práce bylo seznámení se s obchodováním na Forexu a sestavení funkčního obchodního systému. Vytvořená aplikace komunikuje s obchodní platformou MetaTrader 4, získává obchodní data a automaticky realizuje obchodní pokyny na základě uživatelsky definovaných obchodních strategií. Hlavní předností aplikace je jednoduchá definovatelnost obchodních strategií v porovnání s nástroji, dostupnými v rámci platformy MetaTrader.

Conclusions

The aim of this bachelor's thesis was to get acquainted with Forex trading and to build a functional trading system. Application communicates with the MetaTrader 4 trading platform, obtains trading data and automatically executes trading orders based on user-defined trading strategies. The main advantage of the application is the easy definability of trading strategies in comparison with the tools available within the MetaTrader platform.

A Obsah přiloženého CD/DVD

bin/
: Instalační soubory pro Windows a Linux, včetně předinstalovaných verzí.

doc/
: Text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu (v ZIP archivu), tj. zdrojový text textu, vložené obrázky, apod.

src/
: Kompletní zdrojové texty programu.

readme.txt
: Instrukce pro instalaci a spuštění programu.

Literatura

[1] Professional Automated Trading: Theory and Practice, Eugene A. Durenard, 2013. ISBN-13: 978-1118129852

[2] FX Street: typy forexových brokerů [online], FXstreet.cz s.r.o © 2009-2020 [cit. 2020-05-28]. Dostupné z: https://www.fxstreet.cz/typy-forexovych-brokeru.html

[3] AM Admiral Markets Education [online], FXstreet.cz s.r.o © 2009-2020 [cit. 2020-05-28]. Dostupné z: https://admiralmarkets.com/education/articles

[4] Bossainfo: Forex názvosloví [online], Bossa © 2020 [cit. 2020-05-28]. Dostupné z: https://www.bossainfo.cz/forex-nazvoslovi

[5] Volatilita [online]. Wikipedia © 2020 [cit. 2020-05-28]. Dostupné z: https://cs.wikipedia.org/wiki/Volatilita

[6] Technologie používané při algoritmickém obchodování na světových finančních trzích [online] AVC-ČVUT a FIT © 2020 [cit. 2020-05-28]. Dostupné z: https://youtu.be/VwLVc5Zgw7Q

[7] Flux Flarnie Marchan, What is the Flux Application Architecture? [online]. Dostupné z: https://brigade.engineering/what-is-the-flux-application-architecture-b57ebca85b9e

[8] Darwinex, dwx-zeromq-connector. GitHub [online]. Dostupné z: https://github.com/darwinex/dwx-zeromq-connector

[9] Electron, Framework Electron. GitHub [online]. Dostupné z: https://github.com/electron

[10] ZeroMQ: An open-source universal messaging library. GitHub [online]. Dostupné z: https://github.com/zeromq

[11] NodeJS: JavaScript runtime. GitHub [online]. Dostupné z: https://github.com/nodejs/node

[12] React: A declarative, efficient, and flexible JavaScript library for building user interfaces. GitHub [online]. Dostupné z: https://github.com/facebook/react

[13] Redux: Predictable state container for JavaScript apps. GitHub [online]. Dostupné z: https://github.com/reduxjs/redux

[14] Hot Reload: Tweak React components in real time. GitHub [online]. Dostupné z: https://github.com/gaearon/react-hot-loader

[15] Next-js: The React Framework. GitHub [online]. Dostupné z: https://github.com/vercel/next.js

[16] Nextron: Electron + Next.js. GitHub [online]. Dostupné z: https://github.com/saltyshiomix/nextron

[17] Proton Native, Framework Proton Native. GitHub [online]. Dostupné z: https://github.com/kusti8/proton-native

[18] Use-global-hook, Easy state management for react using hooks. GitHub [online]. Dostupné z: https://github.com/andregardi/use-global-hook

[19] BC-1, První verze bakalářské práce. GitHub [online]. Dostupné z: https://github.com/jkvapil6/bc0

[20] BC-2, Druhá verze bakalářské práce. GitHub [online]. Dostupné z: https://github.com/jkvapil6/bc1

[21] ES6, ECMAScript® 2015 Language Specification © 2020. Dostupné z: http://www.ecma-international.org/ecma-262/6.0/

[22] MQL4, Reference, MetaQuotes Software Corp © 2000-2020 [cit. 2020-05-28]. Dostupné z: https://docs.mql4.com/

[23] MT4, MetaTrader4 MetaQuotes Software Corp © 2000-2020 [cit. 2020-05-28]. Dostupné z: http://www.metatrader4.com/