UNIVERZITA PARDUBICE

Fakulta elektrotechniky a informatiky

Webový nástroj pro tvorbu a trénink automatizovaného obchodníka za použití neuronových sítí na platformě Java

Bc. David Boucník

Diplomová práce

2019

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2017/2018**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. David Boucník**
Osobní číslo:	**I16206**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Webový nástroj pro tvorbu a trénink automatizovaného obchodníka za použití neuronových sítí na platformě Java**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

Cílem práce je navrhnout a realizovat webovou aplikaci pro tvorbu a trénování automatického burzovního obchodníka. Automatický obchodník bude reprezentován neuronovou sítí s uživatelem definovanými parametry. Při trénování budou použity časové řady reprezentující sledované veličiny na uživatelem vybraných trzích. Obchodník následně bude v provozním režimu navrhovat obchodní příkazy. Vytvořené obchodníky bude možné dlouhodobě adaptovat na časové řady veličin z vybraných trhů a průběžně sledovat výsledky z jejich tréninku. Teoretická část: provést rešerši známých aplikací, které umožňují inteligentní automatické obchodování, a provést čtenáře aktuálně populárními frameworky pro tvorbu webových aplikací a aplikací využívajících neuronové sítě na platformě Java. Následovat bude popis konkrétně vybraných frameworků pro práci s umělými neuronovými sítěmi a pro deep learning. Praktická část: použít vybrané technologie k vývoji nástroje umožňujícího uživatelům vytvořit a trénovat obchodníky. Nástroj bude testován na vhodných scénářích. Součástí bude technická dokumentace vytvořeného nástroje a uživatelská příručka.

Rozsah grafických prací: **10**

Rozsah pracovní zprávy: **45**

Forma zpracování diplomové práce: **tištěná**

Seznam odborné literatury:

HAYKIN, Simon S. Neural networks and learning machines. 3rd ed. New York: Prentice Hall, 2009. ISBN 978-0-13-147139-9. LI DENG, DONG YU. Deep Learning. Boston: Now Publishers Inc, 2014. ISBN 978-1-60-198814-0.

Vedoucí diplomové práce: **doc. Ing. Petr Doležel, Ph.D.**
Katedra řízení procesů

Datum zadání diplomové práce: **30. října 2017**

Termín odevzdání diplomové práce: **18. května 2018**

Ing. Zdeněk Němec, Ph.D.
děkan

L.S.

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2017

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 23. 8. 2019 David Boucník

PODĚKOVÁNÍ

Především chci ale poděkovat vedoucímu práce panu doc. Ing. Petru Doleželovi, Ph.D. za trpělivost, cenné rady, shovívavost a velkou ochotu během vedení práce.

Rád bych poděkoval Radkovi Charvátovi, Jiřímu Jechovi a Ondřeji Marešovi za korekturu a rady.

ANOTACE

Cílem práce je navrhnout a realizovat webovou aplikaci pro tvorbu a trénování automatického burzovního obchodníka. Automatický obchodník bude reprezentován neuronovou sítí s uživatelem definovanými parametry. Při trénování budou použity časové řady reprezentující sledované veličiny na uživatelem vybraných trzích. Obchodník následně bude v provozním režimu navrhovat obchodní příkazy. Vytvořené obchodníky bude možné dlouhodobě adaptovat na časové řady veličin z vybraných trhů a průběžně sledovat výsledky z jejich tréninku. Teoretická část: provést rešerši známých aplikací, které umožňují inteligentní automatické obchodování, a provést čtenáře aktuálně populárními frameworky pro tvorbu webových aplikací a aplikací využívajících neuronové sítě na platformě Java. Následovat bude popis konkrétně vybraných frameworků pro práci s umělými neuronovými sítěmi a pro deep learning. Praktická část: použít vybrané technologie k vývoji nástroje umožňujícího uživatelům vytvořit a trénovat obchodníky. Nástroj bude testován na vhodných scénářích. Součástí bude technická dokumentace vytvořeného nástroje a uživatelská příručka.

KLÍČOVÁ SLOVA

Java, Spring framework, neuronové sítě, genetický algoritmus, kvantitativní obchodování

TITLE

Web Tool for Training of Automated Stockbroker using Neural Networks and Java Platform.

ANNOTATION

Aim of this thesis is to design and to implement web application for creation and training of automated stock exchange trader. Automated trader will be represented by a neural network with user-defined parameters. Time series representing selected stock exchange markets will be used for training. Trained trader in operating regime will be capable to propose trades. It will be possible to adapt created traders in long term and observe results of training. Theoretical part of thesis: do research of known applications which allow intelligent automated trading. Reader will be guided through popular java web frameworks and through java neural network frameworks. Selected neural network frameworks and deep learning frameworks will be described. Practical part of thesis: develop tool for creating and training of traders using selected technologies. This tool will be tested in suited scenarios. Technical documentation and user guide will be part of the thesis.

KEYWORDS

Java, Spring Framework, neural networks, genetic algorithm, quantitative trading

Obsah

Seznam zkratek

MVC	Model-View-Controller
API	Application Programming Interface
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
XML	Extensible Markup Language
CSV	Comma-separated values
JDK	Java Development Kit
NPM	Node Package Manager
HTTP	Hypertext Transfer Protocol
JDBC	Java Database Connectivity API
JPA	Java Persistence API
JSP	Java Server Pages
ORM	Objektově-relační mapování
REST	Representational State Transfer
IT	Information Technology
JSF	Java Server Faces
GWT	Google Web Toolkit
JSR	Java Specification Request
AJAX	Asynchronous Javascript and XML
DOM	Document Object Model
SKIL CE	Skymind Intelligence Layer Community Edition
JVM	Java Virtual Machine
DL4J	Deep Learning For Java
ARFF	Attribute-Relation File Format
RBM	Restricted Boltzmann Machines
RBF	Radial Basis Function
C#	C Sharp
NEAT	NeuroEvolution of Augmenting Topologies
MACD	Moving Average Convergence Divergence
RSI	Relative Strength Index

Seznam obrázků

Seznam tabulek

Úvod

Automatizované obchodování je dnes nástroj využívaný řadou organizací působících ve finančním sektoru po celém světě. Pro realizaci automatizovaného obchodování je využívána řada systémů postavených na různých algoritmech a technologiích.

V rámci první kapitoly budou tedy přiblíženy základní pojmy a principy z oblasti automatizovaného obchodování a umělé inteligence. Dále budou také v této kapitole formulovány cíle práce.

Vzhledem k tomu, že webové nástroje pro podporu automatizovaného obchodování již existují, tak bude i jim věnována jedna kapitola této práce. Z dostupných zdrojů nelze sice tyto nástroje detailně popsat po stránce použitých technologií a algoritmů, budou tedy alespoň stručně představeny.

Protože cílem této práce je vytvořit webový nástroj na platformě Java, tak je následující kapitola zaměřena právě na popis existujících populárních frameworků použitelných k tomuto účelu.

Po výše zmíněných kapitolách bude následovat praktická část popisující vytvořenou aplikaci pro trénink automatizovaného obchodníka za použití vhodných frameworků a algoritmů.

Součástí práce je i uživatelská příručka vytvořeného nástroje.

1.1 Základní pojmy

Následující kapitola definuje základních pojmy nezbytné k pochopení smyslu a cílů této práce:

- Umělá inteligence – jedná se o vědecký obor, který se zabývá přetvářením strojů a systémů tak, aby při řešení problémů používaly takových postupů, které bychom při jejich použití člověkem považovali za výraz inteligence.[22]
- Strojové učení – podmnožina umělé inteligence zabývající se tvorbou programů, které jsou schopné se samy učit a na základě toho měnit svoje chování bez toho, aby byly explicitně programovány. [23]
- Neuronová síť – tento pojem je v této práci používán pro označení umělé neuronové sítě. Umělé neuronové sítě jsou matematické modely inspirované biologickými neuronovými sítěmi obsaženými v živých organismech. Základním kamenem těchto sítí jsou umělé modely nervové buňky – umělé neurony. [22]
- Backtesting – způsob ověření funkčnosti obchodní strategie tak, že daná strategie je aplikována na historická data. [24]
- Algoritmické obchodování – obchodování při němž obchodník nadefinuje pravidla, na základě kterých následně program zadává obchodní příkazy již bez zásahu člověka. Výhodou tohoto způsobu obchodování je například to, že program je odstíněn od lidských emocí a je také schopný reagovat daleko rychleji na změnu stavu trhu. [25]
- Kvantitativní obchodování – jedná se o obchodování, které je založeno na matematických analýzách a výpočtech. Zpravidla je využíváno finančními institucemi a fondy pro obchodování ve velkém objemu. [26]
- Platforma Java – softwarové prostředí pro běh programů. Sestává z rozhraní Java API a virtuálního stroje JVM. Java API je množina komponent rozdělených do jednotlivých knihoven, které běžící program využívá. Virtuální stroj JVM funguje jako most k hardwaru. Díky tomuto konceptu jsou Java programy nezávislé na hardwarové platformě, pod kterou běží. [27]
- Genetický algoritmus – jedná se algoritmus uplatňující principy Darwinovy evoluční teorie v informatice. Pracuje s populací sestávající z jedinců představujících jednotlivá zakódovaná řešení daného problému. Na začátku algoritmu jsou jedinci v populaci vygenerováni náhodně. V každé iteraci algoritmu jsou provedeny následující kroky:
    - určí biologickou zdatnost (tj. kvalitu řešení daného problému) jednotlivých jedinců pomocí funkce fitness,

- biologicky nejlépe zdatní jedinci jsou označeni za elitu a jsou přímo přeneseni do další generace,
- ostatní členové další generace jsou doplněni křížením vybraných rodičů z předcházející generace,
- u jedinců, kteří vznikají křížením bývá aplikována i mutace tj. drobné pozměnění hodnoty vybraných genů.

Po provedení poslední iterace je za řešení označen jedinec, který nejlépe vyhovuje funkci fitness. [40]

1.2 Cíle práce

Cílem této práce je navržení a implementace webového nástroje pro vytváření a trénink automatických obchodníků. Každý obchodník v tomto nástroji bude tvořen neuronovou sítí, jejíž struktura a parametry budou volitelné. K tréninku sítě budou použita historická data z burzovních trhů vybraných uživatelem reprezentovaná časovými řadami. Uživatel se vytrénovaných obchodníků bude moci v tomto nástroji dotázat na návrh obchodního příkazu. Během tréninku obchodníků bude možné sledovat jejich průběžné výsledky.

1.3 Hypotéza: Pomocí genetického algoritmu je možné vytrénovat neuronovou síť, která bude schopna obchodovat se ziskem

Praktická část práce vychází z předpokladu, že neuronovou síť, která bude schopna v roli obchodníka obchodovat se ziskem, lze vytvořit následujícím přístupem:

- optimalizace hodnot vah bude provedena genetickým algoritmem,
- jako funkce fitness bude použit backtesting,
- pro zkvalitnění rozhodování neuronové sítě bude přiváděno více vstupních řad a také časové řady odvozené z historických dat.

2 Rešerše aplikací pro automatické inteligentní obchodování

Společnost CB Information Services, Inc. provedla průzkum, ve kterém se snažila zmapovat firmy finančního sektoru používající pro poskytování služeb umělou inteligenci. Během průzkumu byly tyto společnosti rozděleny do skupin dle bližšího zaměření jejich činnosti, konkrétně se jedná o:

- komerční bankovnictví,
- nabídky úvěrů,
- pojišťovnictví,
- účetnictví a osobní finance,
- regulace,
- investiční management (obsahuje kvantitativní obchodování). [16]

Do posledního jmenovaného zaměření spadá také kvantitativní obchodování, tedy obchodování řízené algoritmem na základě určitých pravidel a vstupních dat. [17] Fungování firem s tímto zaměřením je tedy založeno na vývoji a používání softwaru stejného typu, jehož vytvoření je předmětem této práce.

V následujících kapitolách budou stručně představeny softwarové projekty výše zmíněných firem.

2.1 Pit.ai

Softwarové řešení pro automatické obchodování americké společnosti Pit.AI Technologies je založeno na myšlence, že umělá inteligence může nabídnout lepší řešení pro investiční řízení než člověk anebo tradiční algoritmické přístupy. Pro tuto myšlenku hovoří více argumentů, zejména však fakt, že pro zpracování velkého množství finančních dat má stroj oproti člověku přirozeně výhodu. Dalším argumentem je také to, že se vstupem počítačů na burzovní trhy došlo k vytvoření prostředí, kde má člověkem řízené obchodování ještě nižší šanci uspět. Tvůrci řešení Pit.ai také věří v to, že automatizace obchodních transakcí může být vhodným řešením pro minimalizaci obchodních nákladů a tvrdí, že výpočetní síla je nyní dostatečně levná a výkonná k tomu, aby způsobila revoluci v celém finančním odvětví. [1]

Cílem tohoto řešení je vytvořit inteligentní nástroj pro řízení investic a eliminovat poplatky pro správu investičního fondu. [1]

Projekt Pit.ai je zaměřen především na vyvinutí agenta disponujícího umělou inteligencí, který se sám učí investovat a je schopen hlubších vhledů do rozsáhlých souborů dat z trhů, než jeho lidský protějšek. Při vývoji jsou používány nejmodernější, nepublikované technologie založené na strojovém učení s uplatněním nových technik a netradičního přístupu k finanční matematice. Při vývoji tohoto produktu bylo dosud třeba mimo standardní záležitosti vývoje softwaru i posunout hranice samotných metod strojového učení, pomocí čehož bylo dosaženo dobrých výsledků. [1]

Vývoj je inspirován čtyřmi základními principy, jedná se o principy granularity, škálovatelnosti, automatizaci a adaptability. [1]

Princip granularity říká, že agent by měl být schopen rozeznat krátkodobé i dlouhodobé příležitosti na trhu. Tyto příležitosti mohou být velmi komplexní a těžce rozpoznatelné, což je činí neviditelné z pohledu běžných přístupů standardně používaných většinou investičních společností. [1]

Princip škálovatelnosti by měl zaručit možnost použití technologie na různých druzích aktiv s různou délkou jejich držení, různým cílem, na různých trzích a bez většího úsilí či komplikací. [1]

Důležitým principem, na který je při vývoji projektu brán zřetel, je automatizace s jejíž pomocí je zaručeno to, že agenti disponující umělou inteligencí mohou jednat bez zásahu člověka a tím být i imunní vůči jeho případné zaujatosti. [1]

Zavedením principu adaptability by agenti měli být schopni průběžně monitorovat data a v čase se postupně přizpůsobovat novým podmínkám, které na trhu aktuálně panují. [1]

2.2 Alpaca

Společnost AlpacaJapan Co., Ltd. se sídlem v japonském Tokiu přichází s řešením automatizovaného obchodování přinášejícím možnost automatizovat obchody s algoritmy sestavenými přímo uživatelem bez nutnosti jejich programování. [2]

V rámci softwarového řešení Alpaca je také dostupný modul usnadňující rozhodování při provádění obchodů využívající předpovědi. Tento modul využívá ke svému fungování potenciál vysokorychlostního datového úložiště a nástrojů umělé inteligence založených na hlubokém učení. Samotné provádění obchodů je možné plně automatizovat pomocí uživatelem vytvořeného algoritmu během několika málo sekund. [2]

Funkcionalita vytváření algoritmu disponuje vizuálním uživatelským rozhraním. Uživateli je zobrazen graf reprezentující historický vývoj cen na trhu. Uživatel pomocí myši vyznačí v grafu body reprezentující okamžiky nákupů a prodejů. Takto zadané informace jsou následně zpracovány pomocí hlubokého učení a na základě tohoto procesu je uživatelský algoritmus sestaven. Poté je možné upravit ještě některé parametry algoritmu a následně jej používat v polo automatickém nebo plně automatickém režimu. V polo automatickém režimu aplikace je vytvořený algoritmus použit pro generování upozornění uživateli doručované do prohlížeče nebo mobilního telefonu a ten může následně obchod provést nebo nechat upozornění bez reakce. V plně automatickém režimu aplikace nechá algoritmus rovnou provádět obchodní příkazy. Pro vstupy algoritmu je možné využít množství měnových párů, k jeho otestování jsou k dispozici historická data až za více než 10 let. Běh algoritmu je možné uskutečnit na minutových nebo okamžikových datech. K dnešním dnům eviduje Alpaca ve své databázi přibližně 15 000 algoritmů. [3]

Modul softwarového řešení pro predikci vývoje trhu AlpacaForecast je založen na hlubokém učení a na vysokorychlostním úložišti dat. [4]

Krátkodobé předpovědi v rozsahu 20 až 30 minut využívají též technologii založenou na hlubokém učení, uživatel má možnost upravit model neuronové sítě pro danou úroveň přesnosti a frekvenci dat. Středně až dlouhodobé předpovědi v řádu několika týdnů jsou prováděny také hlubokým učením, model neuronové sítě pro předpověď cen je navržen na míru zákazníkovi. Pro historická data z finančních trhů bylo vyvinuto speciální úložiště zvané MarketStore. Předpovědní model je dále tvořen specializovanými komponentami na platformě Docker běžícím v Cloudových řešeních Amazon Web Service nebo Microsoft Azure. [4]

MarketStore je optimalizován pro časové řady, podporuje velmi rychlé dotazování, které je tolik důležité pro tvorbu krátkodobých předpovědí. Do MarketStore se mohou okamžikové hodnoty ukládat v reálném čase s tím, že k historickým datům, ale i k datům uložených v reálném čase je možné přistupovat stejným způsobem. Celý modul MarketStore je napsán v jazyce Go a umožňuje binární vyhledávání. [4]

2.3 Algoriz

Aplikace Algoriz je softwarové řešení využívající umělou inteligence od firmy Algoriz Inc. sídlící v americkém New Yorku. Tato aplikace nabízí prostředky pro rychlé vytvoření, testování a automatizaci obchodních strategií založených na tradičních anebo alternativních

zdrojích dat. K různým zdrojům dat lze přistupovat přes jedno jediné intuitivně zpracované uživatelské rozhraní. Za použití zcela stejného přístupu bez ohledu na zdroj dat lze v angličtině vytvořit a otestovat obchodní strategie. Uživatel také může zapnout upozorňování na vzniklé obchodní příležitosti. [18]

K procesu tvorby strategie lze také do aplikace nahrát do uživatelského účtu uživatelská data a následně je používat. Je možné je nechat přímo v aplikaci anebo je nahrát do soukromého cloudového úložiště. Budování obchodní strategie uživatel provádí nad vlastními daty nebo daty poskytnutými od poskytovatele aplikace. [18]

3 Aktuálně populární frameworky pro tvorbu webových aplikací na platformě Java

Firma ZeroTurnaround vytvořila tzv. Java Web Frameworks Index, tedy ukazatel reflektující popularitu jednotlivých frameworků pro tvorbu webových aplikací na platformě Java. Tento ukazatel čerpá informace o popularitě z několika následujících zdrojů:

- StackOverflow,
- LinkedIn,
- GitHub,
- Google.

Tyto zdroje budou v následujících odstavcích stručně představeny.[10]

Webová stránka StackOverflow je místem, kde vývojáři pokládají dotazy týkající se problémů souvisejících s použitím dané technologie, z počtu těchto položených dotazů je odvozována míra popularity dané technologie.[10]

Sociální síť LinkedIn používána mimo jiné profesionály v IT obsahuje cenné informace o technologiích, se kterými daný člověk pracuje nebo v minulosti pracoval, počet profilů profesionálů zmiňujících zkušenost s danou technologií je použita k určení míry popularity.[10]

GitHub je služba poskytující nejrozsáhlejší hosting Git repozitářů, je zde k nalezení velké množství veřejně dostupných zdrojových kódů aplikací. Tyto zdrojové kódy patří většinou tutoriálům nebo malým projektům, jejich předmětem je ověření funkčnosti konceptů v daném frameworku. Míra popularity tohoto zdroje je odvozena z množství použití artefaktů frameworku v konfiguračních souborech Maven nebo Gradle projektů.[10]

Nejznámějším způsobem měření oblíbenosti je vyhledávání Google. Tvůrci indexu vycházeli z předpokladu, že pokud se lidé o daný framework zajímají, pak se o něm snaží vyhledat informace. Do indexu byl tedy zařazen proto, že indikuje zájem lidí. Míra oblíbenosti pro vyhledávač Google je odvozena z počtu dotazů typu „název frameworku framework“ (např. tedy „spring framework“) za poslední měsíc.[10]

Dle výše zmíněného žebříčku patří mezi nejrozšířenější frameworky pro tvorbu webových aplikaci na platformě Java tyto nástroje:

- Spring MVC a Spring Boot,

- JSF,
- GWT.

Tyto technologie budou v následujících kapitolách stručně představeny.

3.1 Spring MVC a Spring Boot

3.1.1 Spring Framework a Spring MVC

Spring Framework poskytuje vývojový a konfigurační model pro tvorbu aplikací jakéhokoli typu a pro libovolný způsob nasazení. Důraz je v této technologii kladen na možnost nechat vývojáře soustředit se na aplikační logiku a zcela je oprostit od detailů týkajících se např. samotného nasazení aplikace. [13]

Jádrem Spring Frameworku jsou komponenty zabezpečující dependency injection, práci s událostmi, správu zdrojů, lokalizaci a internacionalizaci aplikace, ale také například data binding, konverzi typů nebo aspektově orientované programování. [13]

K testování aplikace jsou ve Spring Frameworku k nalezení nástroje pro mockování objektů, nastavení testovacího kontextu, testování MVC aplikace a testovací webový klient. [13]

Pomocí nástrojů jako transakce, podpora pro objekty zdrojových dat, JDBC, objektově-relační mapování a marshalling umí Spring Framework pracovat s daty. [13]

Součástí frameworku jsou i prostředy pro tvorbu webových aplikací WebFlux a Spring MVC. Mezi podporované jazyky Spring Frameworkem se řadí vedle jazyku Java také Kotlin a Groovy.[13]

Spring MVC je původní webový framework obsažený ve Spring Frameworku od jeho samotného počátku. Spring MVC je založen na Servlet API, jeho zdrojovým modulem je spring-webmvc a formálně je pojmenován jako Spring Web MVC. [15]

Centrem fungování Spring MVC je centrální řadič, v tomto případě DispatcherServlet obsahující sdílené obsloužení requestů. Sama logika aplikace obsahující mapování requestů, obsluhu chyb a podobně je však roztříštěna v jednotlivých takzvaných kontrolerech, které DispatcherServlet nalezne v konfiguraci Spring Frameworku. Podobně jako jiné servlety je i DispatcherServlet registrován standardně v konfiguračním souboru webové Java aplikace zvaném web.xml.[15]

3.1.2 Spring Boot

Spring Boot usnadňuje tvorbu samostatných aplikací rychle připravených ke spuštění a nasazení. Spring Boot již sám o sobě obsahuje množinu spring komponent a knihoven třetích stran, při použití spring boot je tedy zapotřebí zpravidla pouze minimální konfigurace spring frameworku. [14]

Mimo vytvoření samostatné Spring aplikace nabízí Spring i možnost použití vestavěného webserveru Tomcat, Jetty či Undertow. Spring Boot poskytuje množinu spring komponent a komponent třetích stran, které umí také automaticky konfiguraovat. Dále najdeme v této technologii i funkce jako jsou metriky, míry kondice aplikace a vnější konfigurace bez jakékoli nutnosti generování kódu nebo složité XML konfigurace. Vzhledem k tomu, že je tato technologie komunitním projektem, má i svojí komunitu na komunikátoru Gitter.[14]

3.2 JSF

Technologie Java Server Faces definuje aplikační rozhraní pro reprezentaci komponent uživatelského rozhraní, události aplikace, validace uživatelského vstupu a mimo jiné také navigaci uživatele ve webové aplikaci. Mezi další funkce této technologii lze také zařadit nástroje pro internacionalizaci. Součástí JSF jsou i JSP tagy pro definici JSF uživatelského rozhraní uvnitř JSP stránky. [11]

JSF je vytvořeno jako velice flexibilní nástroj schopný nabídnout možnosti pro vytvoření stávajících a standardních typů uživatelských rozhraní za použití různých značkovacích jazyků, protokolů a pro různá klientská zařízení. Jednotlivá JSF komponenta obsahuje funkcionalitu, jejíž prezentace je vždy dle stávajícího klienta. Pro renderování HTML kódu obsahuje vlastní knihovnu JSP tagů. Za povšimnutí stojí také fakt, že JSP tagy mohou být definovány pro konkrétní koncové zařízení.[11]

JSF byly vyvinuty v rámci JSR-314 se zaměřením především na jednoduchost použití. JSF disponuje snadným spojením prezentační vrstvy a funkčního kódu, aplikace je postavena ze samostatných a zároveň snadno propojitelných komponent. JSF je zavedeným standardem pro serverová rozhraní.[11]

3.3 GWT

Google Web Toolkit je sadou open source vývojových nástrojů pro vytvoření a optimalizaci webových aplikací používaných tisíci vývojáři po celém světě. Cílem projektu je to, aby

vývojáři bez znalosti detailů webových prohlížečů, protokolu HTTP a JavaScriptu byli schopni vyvinout vysoce výkonné webové aplikace. [12]

Javascript je programovací jazyk běžící v prohlížeči. Dokáže manipulovat jak HTML strukturou webové stránky, tak CSS vlastnostmi jednotlivých elementů a pracovat s daty. [33]

V knihovně GWT je k nalezení řada API a komponent tvořících jádro tohoto nástroje. Díky tomu je možné napsat aplikace založené na AJAXu v Javě a následně nechat zkompilovat do Javascriptu, který je optimalizovaný a schopný běžet ve všech prohlížečích včetně mobilních webových prohlížečů pro Android a iPhone. [12]

Vytvoření aplikace založené na AJAXu je tedy z tohoto pohledu efektivnější z důvodu, že GWT nabízí vyšší míru abstrakce, která vývojáře ušetří detailů práce s DOM a síťovou komunikací. Vedle již existujících standardních komponent GWT je samozřejmě možné vytvořit vlastní pracující například i s ručně napsaným JavaScript kódem. [12]

Kompilátor je jeden z hlavních nástrojů v GWT. Během kompilace jsou prováděny optimalizace kódu, inlinování metod, optimalizace práce s řetězci a vynechávání fragmentů nepoužitého kódu. Dále může rozdělit kód celé aplikace do menších částí, které jsou stahovány do prohlížeče postupně a díky čemuž je prvotní start aplikace rychlejší. [12]

Diagnostika problémů výkonu webových aplikací je pokryta nástrojem Speed Tracer pro Google Chrome. Výkonově problematická místa GWT aplikace v Javascriptu nebo CSS lze tedy diagnostikovat přímo tímto doplňkem prohlížeče. [12]

Za povšimnutí také stojí plugin pro vývojové prostředí Eclipse jako podpora pro vývojáře používající GWT. [12]

4 Aktuálně populární frameworky pro tvorbu aplikací využívajících neuronové sítě na platformě Java

Dle přehledu zabývajícím se dostupnými nástroji podporujícími vývoj aplikací využívajících umělou inteligenci na platformě Java uveřejněném na webu projektu Deeplearning4j lze mezi frameworky pro tvorbu aplikací využívajících neuronové sítě zařadit tyto projekty:

- Deeplearning4j,
- Neuroph,
- Encog,
- Smile,
- OpenNLP,
- CoreNLP. [21]

V následující části práce budou výše zmíněné projekty stručně představeny s vyjímkou OpenNLP a CoreNLP, které neuronové sítě využívají pouze pro zpracování přirozeného jazyka.

4.1 Eclipse Deeplearning4j

Eclipse Deeplearning4j je open source knihovna pro podporu hlubokého učení v prostředí Java a Scala, disponuje plnou integrací s Hadoop a Apache Spark. Cílem této knihovny je zpřístupnit firmám prostředky umělé inteligence běžících na distribuovaných procesorech a grafických čipech. Projekt Eclipse Deeplearning4j je také podporován týmem Skymind, který si stanovil za úkol spojovat Deeplearning4j s dalšími knihovnami jako Tensorflow nebo Keras do Skymind Intelligence Layer Community Edition (SKIL CE). SKIL CE je tedy nástroj použitelný k rychlému trénování a nasazení počítačové umělé inteligence sloužící jako most mezi prostředím Python a JVM. [8]

4.1.1 Základní koncepce Eclipse Deeplearning4j

Strojové učení obvykle sestává ze dvou základních fází. První z nich má za úkol připravit data, tedy najít, transformovat a načíst vhodná data k učení. Pro vytvoření datové roury, která zajišťuje tyto operace, slouží v Eclipse Deeplearning4j knihovna DataVec. Algoritmy nacházející se v jádru knihovny DL4J, zabezpečují druhou fázi obsahující samotné učení vytvořeného modelu. [9]

Trénink modelu a příprava dat jsou zde na rozdíl od jiných podobných nástrojů chápány jako zcela oddělené procesy. Větší flexibility se zachováním jednoduchosti je docíleno načtením

vstupních dat pomocí nástroje DataVec na místo prostého odkázání se na umístění souboru s daty. Před samotným začátkem učení je třeba data normalizovat, nebo upravit jejich variabilitu a podobně. Zde může použití DataVec knihovny ušetřit práci a předejít případným chybám při implementaci vlastních transformačních algoritmů. Je možné pracovat s daty v podobě obrázků, CSV dokumentů, ARFF, prostého textu, integračního nástroje Apache Camel nebo jinými. DataVec data čte pomocí implementací rozhraní RecordReader nebo RecordReaderDataSetIterator. Po vytvoření instance třídy DataSetIterator je možné data využít při tréninku neuronové sítě. [9]

Normalizovaná data do škály od -1 po 1 fungují obvykle pro trénování neuronových sítí nejlépe. Příčinou tohoto faktu je učení sítí klesáním podle gradientu a použití aktivačních funkcí vracejících funkční hodnoty ve škále od -1 do 1. Nastavení příslušné normalizace je provedeno jednoduchým přiřazením instance DataNormalization jako preprocesoru do DataSetIteratoru. [9]

ImagePreProcessingScaler normalizuje obrázková data, NormalizerMinMaxScaler je vhodný pro data identických rozměrů, NormalizerStandardize je zase vhodný pokud rozměry stejné nejsou. Uživatelskou normalizaci si vývojář může vytvořit implementováním rozhraní DataNormalization. NormalizerStandardize si pro svou práci vytváří statistiky vstupních dat, pro úspěšné pozdější obnovení modelu je zapotřebí jej opatřit při ukládání i těmito normalizačními statistikami. [9]

Instance typu DataSetIterator vrací jednotlivé objekty DataSet. Tyto objekty obsahují charakteristiky a popisky dat použitých pro trénování. DataSet obsahuje několik instancí INDArray, jedna slouží pro reprezentaci vlastností dat, další pro uložení popisků a další dvě pro případné masky časových řad. INDArray je reprezentací n-rozměrného pole, může tedy například jako matice zachytit sérii hodnot více znaků, typicky tedy dávku dat vytvořenou jako podmnožinu trénovacích dat. Díky této možnosti není problém využít při učení objem dat převyšující dostupnou operační paměť. [9]

Více hodnot daných znaků v jedné instanci DataSet zaručuje to, že při tréninku modelu klesáním podle gradientu je v rámci dávky přítomna pro učení potřebná změna míry chyby. Pokud by v DataSetu byla pro každý znak přítomna jenom jedna hodnota, pak by to narušilo samotný proces učení klesáním podle gradientu. [9]

Každá dávka vstupních dat by měla obsahovat reprezentativní vzorek vstupních dat. V případě, že je model trénovaný ke klasifikaci dat, pak by každá dávka měla obsahovat všechny třídy v zastoupení, které přibližně odpovídají celé množině vstupních dat. [9]

4.1.2 Vytváření modelů neuronových sítí

Uživatelé knihovny Eclipse Deeplearning4j mají k dispozici vysokoúrovňové nástroje pro vytváření modelů neuronových sítí. Návrhový vzor builder je zde implementován pro snadnou konfiguraci sítě deklarativním přístupem. [9]

```
double rateOfLearning = 0.1;
int numberOfInputs = 3;
int numberOfHiddenNodes = 5;
int numberOfOutputs = 2;
MultiLayerConfiguration multiLayerConfiguration =
    new NeuralNetConfiguration.Builder()
        .optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT)
        .updater(new Nesterovs(rateOfLearning,  momentum: 0.95))
        .list(
            new DenseLayer.Builder()
                .nIn(numberOfInputs)
                .nOut(numberOfHiddenNodes)
                .activation(new ActivationReLU())
                .build(),
            new OutputLayer.Builder(LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD)
                .activation(new ActivationSoftmax())
                .nIn(numberOfHiddenNodes)
                .nOut(numberOfOutputs).build()
        )
        .backprop(true).build();
```

Obrázek 1 - Příklad konfigurace neuronové sítě v Deeplearning4j

Z Obrázku 1 obsahujícího fragment kódu konfigurující vícevrstvou neuronovou síť je jasně patrné, že knihovna Eclipse Deeplearning4j odděluje konfiguraci optimalizačních a aktualizačních algoritmů modelu odděleně. Tato skutečnost znamená vyšší flexibilitu při konfiguraci vhodného modelu pro danou množinu dat. Pro konfiguraci například rekurentních nebo konvolučních sítí jsou k dispozici vedle typů vrstev DenseLayer a OtputLayer viditelných na Obrázku 1 také:

- GravesLSTM,
- ConvolutionLayer,
- RBM,
- EmbeddingLayer. [9]

4.1.3 Trénink modelu sítě

Po nakonfigurování modelu sítě je nejsnazší způsob, jak zahájit trénování sítě voláním metody fit() za použití instance DataSetIterator jako agumentu. Volání metody fit() provede jednu

epochu tréninku, projde tedy postupně celou množinou dat z DataSetIteratoru. Postupné volání metody fit() provádí postupně libovolné množství epoch. [9]

Jinou možností je použít instanci třídy EarlyStoppingTrainer. Instanci této třídy lze při konfiguraci předat informace o množství epoch nebo čase vyhrazeném na trénování, ta se následně postará o samotné volání metody fit(). Navíc provádí i vyhodnocení výkonu modelu po provedení každé trénovací epochy a nejlépe hodnocené modely ukládá pro pozdější použití. [9]

4.1.4 Testování a ladění modelu sítě

Dalším krokem po tréninku modelu neuronové sítě je obvykle testování. Pro tento účel se používá jiná množina dat než k samotnému tréninku, čímž je zabráněno přeučení sítě na trénovací data. K tomuto účelu je v knihovně Deeplearning4j třída Evaluation, která testování dopředných a rekurentních modelů sítí však pojímá mírně odlišně, avšak základní princip zachovává. Pomocí modelu sítě a testovacích dat provede sérii testů, na základě kterých přesnost modelu vyhodnotí. [9]

Pro sledování procesu tréninku je možné zaregistrovat různé typy listenerů. Příkladem takového listeneru je ScoreIterationListener, který jenom vypisuje aktuální míru chyby daného modelu sítě. Jiný typ listeneru HistogramIterationListener spustí speciální webové grafické rozhraní umožňující uživateli detailně sledovat stávající konfiguraci modelu. [9]

4.2 Neuroph

Framework Neuroph je odlehčený nástroj sloužící jako podpora pro práci s neuronovými sítěmi na platformě Java. Umožňuje tvorbu jednoduchých sítí běžně používaných typů, obsahuje vhodně navrženou open source knihovnu obsahující malé množství základních tříd reprezentujících základní typy neuronových sítí. Obsahuje i grafické uživatelské rozhraní pro rychlejší a snazší práci s neuronovými sítěmi. Toto uživatelské rozhraní nabízí funkcionality vytváření, trénování a ukládání neuronových sítí. [5]

Ze své povahy je tento framework vhodný pro začátečníky z důvodu svého snadného použití, které nepředpokládá, že uživatel tohoto frameworku má širší teoretické nebo programátorské znalosti. Neuroph je také vhodný pro výzkumné projekty mimo jiné proto, že není příliš rozsáhlý, je velmi flexibilní a disponuje dobrou dokumentací. Neuroph je licencován pod Apache licence 2.0. [5]

Projekt Neuroph byl původně vyvíjen jako bakalářská práce, následně jako magisterská závěrečná práce v roce 2008. Nyní je tento projekt open source a od verze 2 je používán pro výuku kurzu Inteligentní systémy na Fakultě organizačních věd Bělehradské univerzity.[6]

4.2.1 Cíl

Posláním projektu Neuroph bylo vytvořit framework pro práci s neuronovými sítěmi disponující těmito vlastnostmi:

- Intuitivnost, snadná použitelnost, malé rozměry – pro rychlé osvojení a použití i pro začátečníky.
- Flexibilita – pro snadnou rozšiřitelnost a širokou možnost znovupoužití.
- Dobrou dokumentaci – nezdokumentovaná funkcionalita frameworku jako by neexistovala.
- Rychlá použitelnost – použití frameworku má být možné téměř okamžitě.
- Obsahuje potřebné nástroje – framework má pokrývat potřebné nástroje pro vytvoření, trénink, testování a nasazení neuronových sítí.
- Zábavnost – používaní frameworku má být zábava. [6]

4.2.2 Funkcionality

Aktuálně Neuroph obsahuje řadu funkcionalit, mezi podporované architektury neuronových sítí se řadí:

- Adaptivní lineární neuron,
- Jednoduchý perceptron,
- Vícevrstvý perceptron se zpětným šířením chyby,
- Hopfieldova síť,
- obousměrná asociativní paměť,
- Kohonenova síť,
- Hebbova síť,
- Maxnet,
- Kompetitivní síť,
- RBF síť,
- Neuro-Fuzzy inferenční systém.

Mezi další funkcionality frameworku se řadí:

- 10 základních tříd pro snadné použití nebo rozšíření,
- podpora učení s učitelem a bez učitele,
- specializované vývojové prostředí pro jazyk Java založené na NetBeans,
- podpora rozpoznávání obrazu,
- optické rozpoznávání znaků,
- ukázka modelu pro předpověď vývoje trhu,
- vizuální znázornění učení,
- normalizace dat,
- jednoduchá podpora měření výkonu. [7]

4.3 Encog

V roce 2008 byl Jeffem Heatonem zahájen vývoj frameworku za pomocí jazyků C# a Java, jehož účelem byla podpora strojového učení, genetického programování, NEAT/HyperNEAT a dalších technik využívajících neuronové sítě. Díky podpoře neuronových sítí vešel framework Encog do širšího povědomí a je využíván řadou vývojářů. Díky tomu, že samotný framework je napsaný pomocí programovacích jazyků C# a Java je možné aby si jej vývojář pro svou práci i částečně upravil. Encog postrádá podporu pro počítačové vidění, výhodou však zůstává již výše zmíněná podpora pro genetické algoritmy a NEAT. [19]

Encog podporuje následující metody strojového učení:

- metodu podpůrných vektorů,
- neuronové sítě,
- Bayesovské sítě,
- skryté Markovovy modely,
- genetické programování,
- genetické algoritmy.[19]

Implementace většiny výše zmíněných metod podporují vícevláknové vykonávání a jsou tedy schopné využít efektivně vícejádrové systémy.[19]

4.4 Smile

Projekt Smile je rychlý a jednoduše použitelný nástroj strojového učení na platformě Java. Mezi hlavní přednosti tohoto nástroje patří jeho rychlost implementovaných pokročilých algoritmů a jejich nízké nároky na operační paměť. Dle existujících výkonnostních testů citovaných na

webových stránkách projektu mohou aplikace používající Smile běžet na stejném hardware výrazně rychleji než obdobné implementace v prostředí jazyka R, Python, Spark nebo za použití frameworku H2O či XGBoost. Ve výše zmíněném výkonnostním testu byl vítězný framework Smile schopen vykonat požadované operace dokonce i několikanásobně rychleji než v pořadí druhý XGBoost. [20]

Další silnou stránkou frameworku Smile je jeho kompletnost, díky které je schopen podporovat množinu operací související se strojovým učením. Tato množina operací zahrnuje:

- klasifikaci,
- regresi,
- shlukování,
- genetické algoritmy,
- asociační analýzu,
- výběr rysů,
- efektivní hledání nejbližšího souseda,
- a další. [20]

Součástí frameworku je také podpora nástrojů matematiky a statistiky. Konkrétně se jedná o:

- lineární algebru včetně LU rozkladu,
- Choleského rozklad,
- QR rozklad,
- Spektrální analýzu,
- Singulární rozklad,
- Řídké matice,
- T-test,
- F-test,
- Chí-kvadrát test,
- testy pro zjištění korelace,
- Kolmogorovův-Smirnovův test,
- rozdělení pravděpodobnosti,
- generátory čísel,
- podporu interpolace,
- a další. [20]

5 Vytvoření webové aplikace

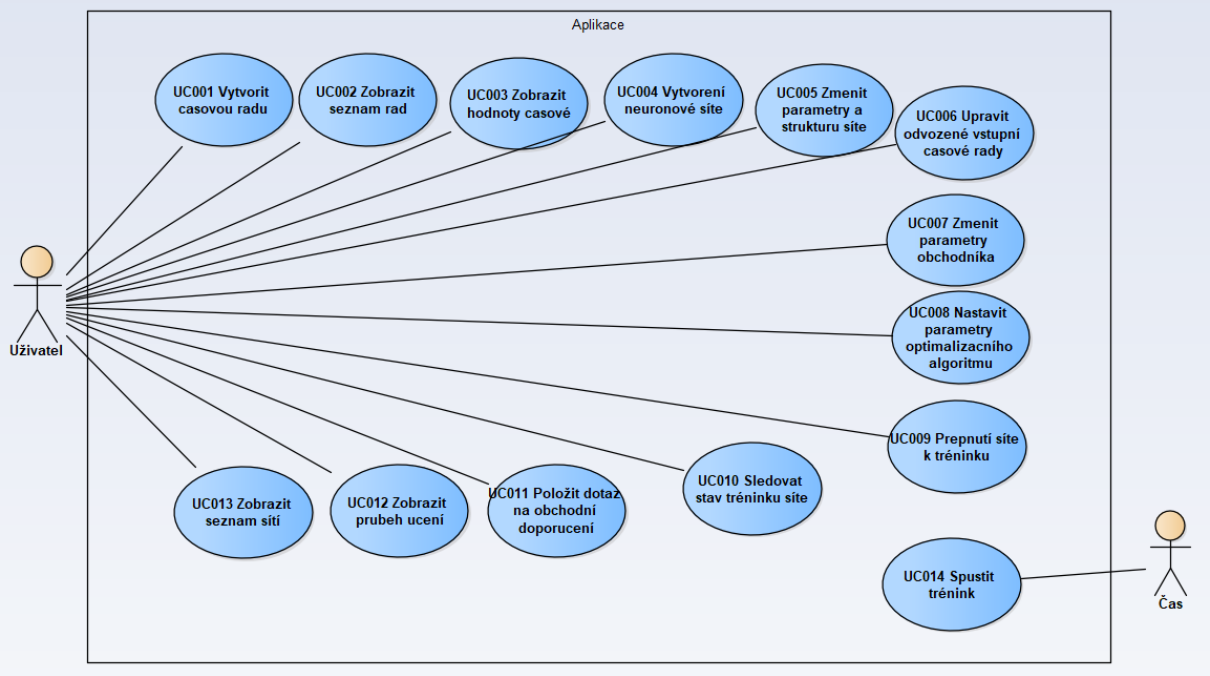
Předmětem této kapitoly bude analýza popisující funkce, které by vytvořená aplikace měla mít pro naplnění cílů práce. Výstupem analýzy jsou diagramy případů užití, které budou také popsány. Následovat bude podkapitola věnující se představení implementace aplikace a její část:

- datová vrstva,
- servisní vrstva,
- REST API,
- klientská část.

V příslušných podkapitolách budou také vysvětleny použité a dosud nepopsané technologie a koncepty.

5.1 Analýza požadavků

Na začátku každého softwarového projektu by měla stát řádně zpracovaná analýza. Jejím výstupem by měla být definice toho, co uživatelé očekávají od výsledného softwarového produktu. Toto je reprezentováno diagramy. V této kapitole budou uvedeny a dále popsány diagramy případů užití.



Obrázek 2 - Diagram případů užití

Jak je patrno z uvedeného diagramu užití, tak v implementované aplikaci mohou akce spouštět pouze dva aktéři. Buďto se jedná o uživatele, jímž je člověk přihlášený ve webové aplikaci, anebo o čas čímž je rozuměno uplynutí určitého časového intervalu.

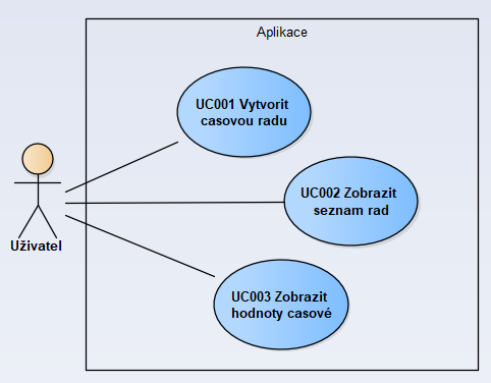
Případů užití je výrazně více než aktérů a lze je rozdělit do tří skupin:

- akce související s časovými řadami,
- vytvoření a úpravy neuronové sítě a souvisejících objektů,
- akce nad trénovanými a vytrénovanými sítěmi.

Následně budou jednotlivé skupiny případů užití blíže představeny.

5.1.1 Akce související s časovými řadami

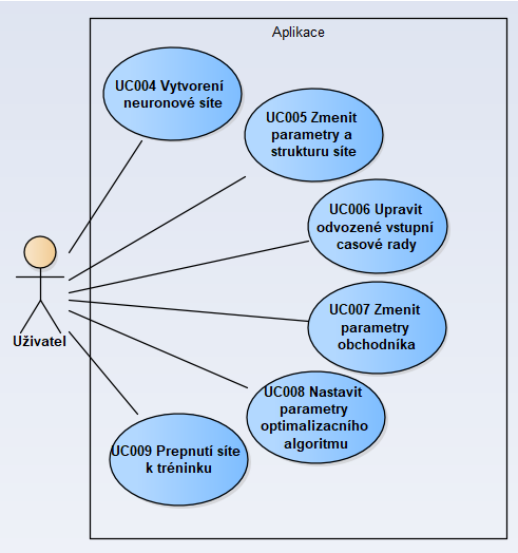
Časové řady dat z burzy budou používány jako vstupy do neuronových sítí během učení.



Obrázek 3 - Případ užití akcí souvisejících s časovými řadami

Uživatel tedy bude potřebovat způsob, jak vytvořit jejich instance a uložit jejich hodnoty uvnitř aplikace. Tento problém zachycuje případ užití UC001. Po vytvoření jednotlivých instancí časových řad bude uživatel potřebovat zobrazit jejich přehled, čímž se zabývá UC002. UC003 se dále zabývá možností graficky zobrazit hodnoty časových řad.

5.1.2 Vytvoření a úpravy neuronové sítě a souvisejících objektů



Obrázek 4 - Diagram případů užití vytvoření a úprav neuronových sítí a souvisejících objektů

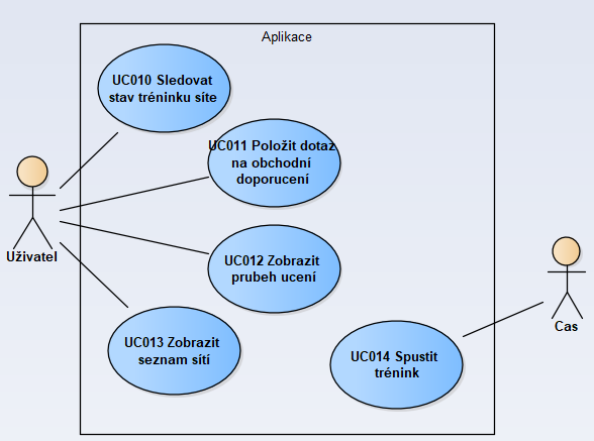
Ve chvíli kdy má uživatel připravená data k trénování neuronových sítí, může přistoupit k dalšímu kroku. Tím je vytvoření instance neuronové sítě reprezentované jako případ užití UC004.

Vytvořené síti lze jak přidat tak odebrat časové řady, které budou použity na vstupu. Dále lze také upravit počet a rozměr skrytých vrstev sítě. Funkcionalita umožňující provádět tyto operace je definována v případu užití UC005.

Během trénování nemusí být na vstup sítě přivedeny pouze přímo hodnoty časových řad, ale i hodnoty z nich odvozené (např. klouzavé průměry). V rámci UC006 by měly být implementovány funkcionality umožňující přidání, odebrání nebo úpravu odvozené vstupní časové řady. Během trénování je biologická zdatnost jedince měřena výkonem v backtestingu, při němž je simulováno provádění obchodů. To jakým způsobem se neuronová síť v roli obchodníka chová, lze ovlivnit nastavením parametrů zpřístupněných v rámci případu užití UC007. Trénink neuronových sítí je řízen optimalizačním algoritmem. Pro změnu způsobu trénovaní a ovlivnění výsledku tréninku lze měnit řadu parametrů tohoto algoritmu, případ užití UC008 zachycuje funkcionalitu, která toto umožňuje. Ve chvíli kdy jsou všechna nastavení neuronové sítě a souvisejících objektů finální, lze přistoupit k samotnému tréninku sítě. Pro zahájení tréninku přepne uživatel síť do stavu signalizujícího připravenost sítě k tréninku, jak je dokumentováno případem užití UC009.

5.1.3 Akce nad trénovanými a vytrénovanými sítěmi

V případu užití UC014 přichází na scénu další aktér. Tímto aktérem je čas, periodicky má totiž docházet každých 10 sekund ke spuštění tréninku pro sítě, které jsou ve stavu udávajícím připravenost k tréninku.



Obrázek 5 - Diagram případů užití akcí nad trénovanými a vytrénovanými sítěmi

Případ užití UC010 říká, že v době kdy je prováděn trénink neuronové sítě, by měl uživatel mít možnost sledovat jeho stav. Tohoto by mělo být dosaženo jak funkcionalitou zobrazující samotný stav sítě, tak i možností prohlížet textový výstup z tréninku.

Pro již vytrénovanou síť má být uživatel schopný manuálně vložit vstupy a dotázat se na její výstup, který aplikace pro uživatele interpretuje. V praxi tedy uživatel do aplikace zadá hodnoty časových řad a aplikace mu zobrazí, zdali by pro takové hodnoty provedla obchod a popřípadě jaký, což je dokumentováno v případu užití UC011.

V případu užití UC012 se hovoří o funkcionalitě zpřístupňující průběžné výsledky optimalizačního algoritmu. Uživatel bude moci graficky zobrazit průměrné a nejlepší skóre populace v jednotlivých generacích. Téměř samozřejmostí je také možnost definovaná v případu užití UC013 a sice zobrazit všechny neuronové sítě v aplikaci a případně procházet jejich nastavení.

5.2 Implementace

K implementaci aplikace byla využita řada technologií a konceptů. Některé z nich nesouvisí přímo s tvorbou webových aplikací na platformě Java a tak nebyly jejich definice pokryty v teoretické části práce. Z tohoto důvodu budou tyto technologie a koncepty představeny zde a následně bude popsáno, jakou roli v tvorbě aplikace hrály:

- **Single page aplikace** – webová aplikace, která pro své fungování načte jedinou HTML stránku a následně dynamicky mění její části na základě interakce s uživatelem [28]
- **MySQL** – jedná se o nejpopulárnější systém řízení báze dat, který je open source. Nyní je vyvíjený pod společností Oracle a vyniká rychlostí, spolehlivostí a snadností použití. [29]
- **ORM** – zkratka pro objektově-relační mapování, tedy nástroj usnadňující spolupráci prostředí objektově orientovaného jazyka a relační databáze.[30]
- **JPA** – zkratka pro Java Persistence API, standardizované rozhraní pro řízení relační databáze z prostředí jazyka Java. [41]
- **Hibernate** – implementace objektově-relačního mapování pro jazyk Java, která respektuje specifikaci JPA. Tento nástroj se vyznačuje vysokým výkonem, škálovatelností a spolehlivostí. [31]
- **React.js** – Javascriptová knihovna vytvořená za účelem tvorby interaktivních uživatelských rozhraní. Za použití této knihovny je celé rozhraní deklarativním způsobem postaveno pomocí jednotlivých znovupoužitelných komponent. [32]
- **REST API** – aplikační rozhraní využívající protokol HTTP. Toto rozhraní běží zpravidla na serveru, jeho služby jsou využívány klientskou částí webové aplikace. [34]
- **DOM –** neboli document object model je datová reprezentace webového dokumentu v prohlížeči. Sestává z hierarchicky uspořádaných uzlů a objektů. [41]
- **Virtual DOM –** jedná se o funkci knihovny React.js. React.js vykreslování jednotlivých komponent neprovádí přímo do DOMu prohlížeče, ale do své virtuální reprezentace stránky zvané virtual DOM. Když je vykreslování komponent do virtual DOM dokončeno, tak se přistoupí k synchronizaci virtuálního DOMu a DOMu prohlížeče, kdy jsou do DOMu prohlížeče přeneseny pouze rozdíly. Tímto způsobem je minimalizována drahá interakce s DOMem prohlížeče a běh aplikace je urychlen.
- **Technická analýza** – analytický pohled na burzovní trh zohledňující předchozí historické výsledky trhu a ukazatele, které jsou na těchto historických výsledcích postaveny. [35]
- **Jednoduchý klouzavý průměr** – prostý aritmetický průměr hodnot spadajících do daného časového okna. [39]
- **Exponenciální klouzavý průměr** – jedná se o klouzavý průměr, pro jehož výpočet se používají váhy, jejichž hodnoty se mění exponenciálně. Hodnota tohoto indikátoru je výsledkem následujícího matematického vztahu:

$$EMA_t = Cena_t * exp + EMA_{t-1} * (1 - exp)$$

V tomto vztahu platí, že

$$exp = 2/(n + 1)$$

A n je perioda klouzavého průměru. [36]

- **MACD** – jedná se o pomalejší indikátor založený na klouzavých průměrech. Je definován jako rozdíl klouzavých průměrů, konkrétně se jedná o klouzavé průměry s délkou okna 12 a 26. [37]
- **Maven** – nástroj usnadňující kompilaci a správu závislostí Java projektů. [43]
- **Bootstrap** – CSS knihovna umožňující uživatelské rozhraní webu sestavit z již připravených komponent. Podporuje moderní přístupy k tvorbě uživatelských rozhraní. [44]
- **Index relativní síly (RSI)** – indikátor patřící k pomalejším, změny signalizuje později. Udává vnitřní relativní sílu trhu a pro jeho výpočet lze použít následující vztah:

$$RSI = 100 - (100/(1 + RS))$$

V tomto vztahu platí, že

$$RS = \frac{počet\ kladných\ změn\ ceny\ za\ periodu}{počet\ záporných\ změn\ ceny\ za\ periodu}$$

[38]

Výstupy analýzy byly implementovány jako webová aplikace. Data aplikace jsou uložena v databázi MySQL, konkrétně byla použita edice Community Edition. Tato databáze byla zvolena z důvodu její popularity, dlouhé historie a především z toho vyplývajícího množství internetových zdrojů pro řešení jakýchkoli potíží během jejího používaní. Serverová část aplikace je napsána v jazyce Java za použití frameworku Spring-Boot. Prostředníka mezi serverovou částí aplikace a databází tvoří ORM nástroj Hibernate, který data z databázového schématu mapuje do tříd datové vrstvy. Databázové schéma bylo vygenerováno automaticky nástrojem Hibernate na základě implementace datových tříd. Každá třída datové vrstvy má také svoji repozitářovou třídu založenou na modulu Spring Data JPA zpřístupňující programové rozhraní se základními operacemi pro čtení, úpravu a zápis dat.

Nad datovou vrstvou pracují třídy spadající do servisní vrstvy. Servisní vrstva obsahuje většinu logiky aplikace a je využívána třídami implementující REST API. Pro práci s umělými neuronovými sítěmi je použit framework Encog představený v teoretické části práce. Důvodem pro jeho volbu byla jeho snadná použitelnost, konfigurace, rozšiřitelnost ale také fakt, že

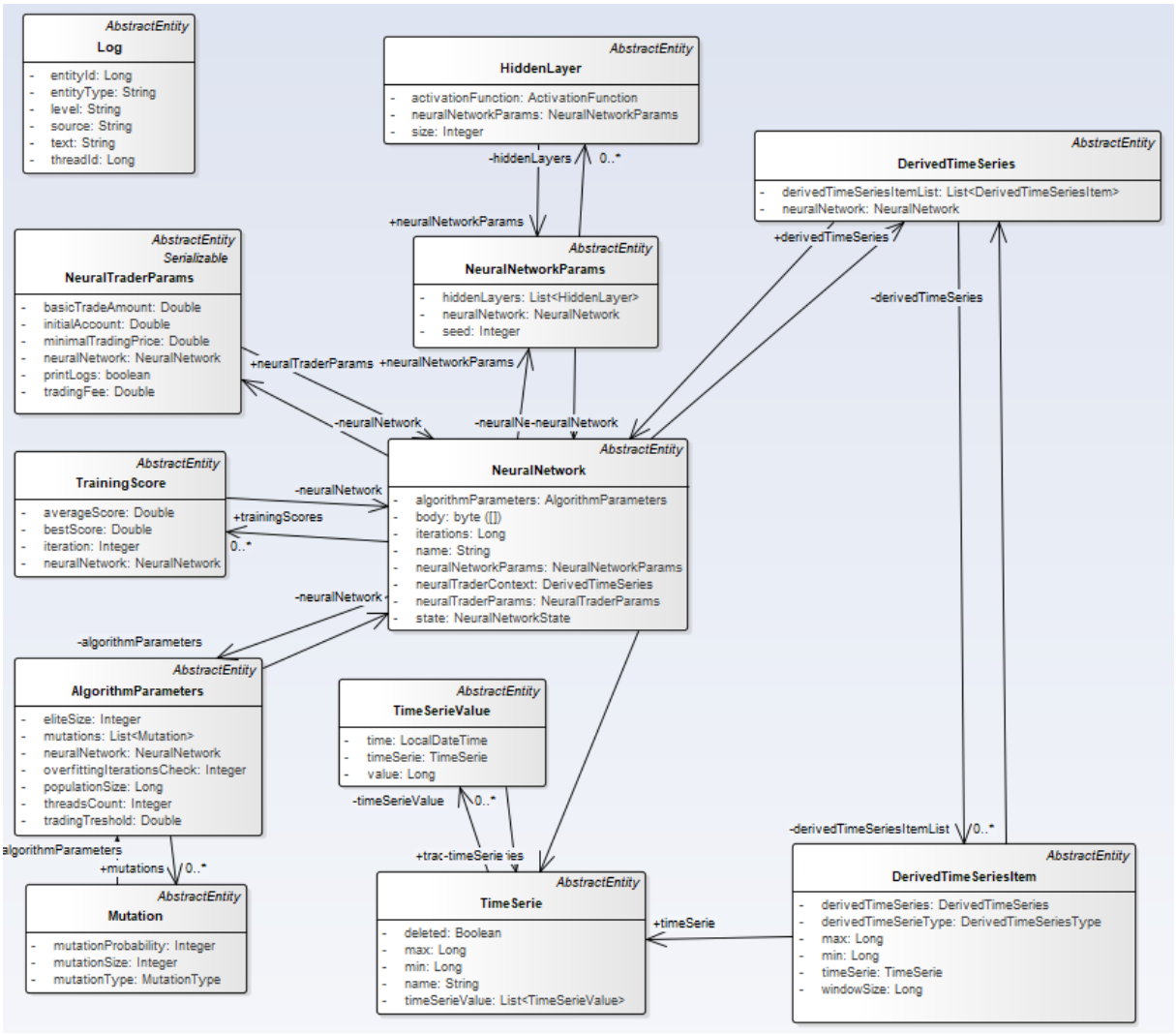
obsahuje jednoduše parametrizovanou implementaci genetického algorimu pro optimalizaci sítě.

Servisní vrstva zároveň také třídy REST API izoluje od vrstvy datové, čímž je tvoří zcela nezávislými. Ve výsledku tedy stávající implementace umožňuje větší změny v datové vrstvě bez nutnosti zasahovat do implementace vrstvy REST API a naopak je také možné provést větší změny v třídách RESTového API bez zásahu do tříd datové vrstvy.

REST API zpřístupňuje skrze HTTP protokol klientské části aplikace potřebné metody pro vytvoření, editaci a čtení datových objektů aplikace.

Klientská část aplikace je napsána za použití jazyka Javascript a knihovny React.js. Jednotlivé obrazovky aplikace jsou vykresleny z komponent knihovny React.js. Pro potřebná data k zobrazení je vždy provedeno volání REST API na serverové části aplikace. Akce provedené uživatelem na jednotlivých obrazovkách upravující data aplikace také vyústí ve volání příslušných metod REST API. Pro usnadnění vykreslování grafů v klientské části aplikace byla využita knihovna Recharts pro React.js. Tabulkové seznamy v aplikaci jsou vytvořeny za pomocí knihovny React Table.

5.2.1 Datová vrstva



Obrázek 6 - Diagram tříd datové vrstvy aplikace

Mezi základní kameny datové vrstvy patří abstraktní třída AbstractEntity obohacující datové třídy o běžná pole jako je jejich id, čas vytvoření a poslední úpravy. Všechny datové třídy jsou od ní odvozeny.

Výše uvedený diagram tříd datové vrstvy byl vytvořen na základě importu přímo ze zdrojových kódů. Obousměrné vztahy mezi třídami mohou působit poněkud nezvykle, jsou totiž nadefinovány tak, aby bylo možné přistupovat k souvisejícím entitám v obou směrech, což samotný kód aplikace v určitých směrech zjednodušuje. Za účelem zaměření se na podstatné vlastnosti tříd bylo u diagramu tříd této vrstvy také vynecháno zobrazování jiných než privátních atributů. V následujícím textu budou postupně stručně popsány významy jednotlivých tříd a funkce jejich privátních atributů.

NeuralNetwork reprezentuje instanci neuronové sítě z pohledu uživatele, obsahuje následující atributy:

- iterations – počet tréninkových iterací sítě,
- name – název sítě,
- state – stav sítě, po vytvoření sítě nabývá hodnoty CREATED, pokud je síť připravena k uskutečnění tréninku, tak je hodnota tohoto pole READY_FOR_TRAINING. Hodnota IN_TRAINING signalizuje síť, která je právě trénována a TRAINED reprezentuje síť již vytrénovanou,
- body – tělo vytrénované neuronové sítě v binární podobě včetně hodnot vah a aktivačních funkcí.

NeuralNetworkParams definuje pro danou neuronovou síť detailnější popis jejích parametrů:

- hiddenLayers – reference na jednotlivé instance vnitřních vrstev sítě,
- seed – násada použitá pro generování iniciálních náhodných hodnot vah sítě.

HiddenLayer představuje jednu vnitřní vrstvu sítě, obsahuje:

- activationFunction – aktivační funkce dané vrstvy,
- size – rozměr vrstvy, tj. počet umělých neuronů v dané vrstvě.

Měření biologické zdatnosti dané neuronové sítě při tréninku probíhá formou backtestingu, kde síť zadává obchodní příkazy. Parametry pro ovlivnění chování neuronové sítě při zadávání těchto příkazů jsou uloženy v NeuralTraderParams v těchto atributech:

- initialAccount – počáteční hodnota obchodního účtu v backtestingu,
- minimalTradingPrice – minimální velikost obchodu,
- basicTradeAmount – základní velikost obchodu, jedná se o konstantu, kterou je vynásoben výstup z neuronové sítě a tím vypočtena velikost obchodu,
- printLogs – při zapnutí této volby se během každého backtestingu zaznamenávají detailní logy o činnosti,
- tradingFee – poplatek za provedení obchodu sítí během backtestingu.

Trénink neuronové sítě je prováděn genetickým algoritmem. Chování tohoto algoritmu ovlivňuje řada parametrů, pro danou neuronovou síť jsou hodnoty těchto parametrů uloženy v AlgorithmParameters za použití těchto atributů:

- eliteSize – procentuální velikost elity,

- overfittingIterationsCheck – během procesu tréninku je možné, že dojde k přeučení sítě. Z tohoto důvodu je v aplikaci proti tomuto zavedena detekce přeučení, která vždy po dokončení iterace zkoumá, zda došlo k přeučení. Během této detekce je brán v potaz pouze určitý počet posledních iterací, počet těchto iterací je definován právě v atributu overfittingIterationsCheck,
- populationSize – velikost populace genetického algoritmu,
- threadsCount – počet vláken, který má být použit k učení dané sítě,
- tradingTreshold – výstupem ze sítě je vždy jedna hodnota na základě které se rozhoduje, zda dojde k nákupu či prodeji. Hodnota tohoto atributu udává prahovou hodnotu, která pokud je absolutní hodnotou výstupu ze sítě překročena, tak se pokusí zadat obchodní příkaz,
- mutations – mutace aplikované při křížení jedinců.

Pro definici parametrů mutací genetického algoritmu je použita třída Mutation s následujícími atributy:

- mutationProbability – procentuální pravděpodobnost provedení dané mutace,
- mutationSize – procentuální množství mutovaných genů,
- mutationType – způsob, jakým je samotná mutace prováděna.

Vstupem do neuronové sítě mohou být i odvozené časové řady. Pro reprezentaci těchto odvozených řad můžeme tedy v diagramu tříd najít třídu DerivedTimeSeries. Tato třída sama o sobě nenese žádná data, ale pouze odkazy na příslušnou neuronovou síť a žádnou nebo více instancí DerivedTimeSeriesItem, která obsahuje následující atributy:

- derivedTimeSerieType – způsob odvození časové řady, může obsahovat hodnoty např. EXPONENTIAL_MOVING_AVERAGE pro exponenciální klouzavý průměr nebo DELAYED_VALUE pro zpožděnou hodnotu ze zdrojové řady a další,
- windowSize – délka okna pro odvozování časové řady, je relevantní jenom pro některé typy odvozených časových řad
- min, max – minimum a maximum hodnot v odvozené časové řadě.

Vzhledem k tomu, že vstupními daty pro neuronové sítě jsou časové řady, tak bylo zapotřebí tyto objekty datově reprezentovat. K tomuto účelu slouží především třída TimeSerie a její atributy:

- min, max – minimum a maximum hodnot v časové řadě,

- name – jméno časové řady,
- deleted – signalizuje to, že je z pohledu uživatele řada smazaná. Důvodem zavedení tohoto pole je fakt, že mazaná časová řada může být využívána jinými objekty v aplikaci (existujícími sítěmi, sítěmi v trénování, odvozenými časovými řadami). Na místo složitého zajišťování bezpečného mazání časové řady je tedy řada raději ponechána v databázi a uživateli skryta.

Pro uložení jednotlivých hodnot časové řady byla zavedena třída TimeSerieValue a doplněna o následující privátní atributy:

- time – čas ke kterému se hodnota vztahuje,
- value – hodnota časové řady.

K zaznamenání průběžných výsledků učení pro jednotlivé iterace je datová vrstva doplněna o třídu TrainingScore disponující:

- averageScore – průměrná hodnota biologické zdatnosti jedince v populaci,
- bestScore – biologická zdatnost nejlepšího jedince,
- iteration – iterace algoritmu pro kterou je daný záznam uložen.

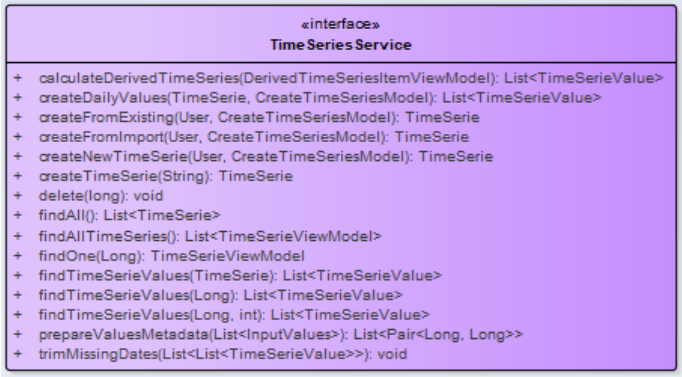
Třída Log uchovává různé textové zprávy z běhu aplikace. Jedná se jak o diagnostické informace, tak zprávy z průběhu učení, které jsou později uživateli zobrazovány. Používá tyto atributy:

- entityId, entityType – identifikátor a typ objektu ke kterému se záznam vztahuje,
- level – úroveň důležitosti zprávy,
- source – zdroj, který zprávu uložil, typicky se jedná o název třídy,
- text – znění zprávy,
- threadId – identifikátor vlákna, které log uložilo.

5.2.2 Servisní vrstva

Servisní vrstva tříd aplikace zpřístupňuje řady operací a obsahuje tedy většinu logiky aplikace. Její detailní popis by byl značně rozsáhlý a zřejmě by ani neměl větší význam z pohledu fungování aplikace. Z tohoto důvodu budou v této podkapitole popsány pouze vybrané operace vybraných servisních tříd. K detailnějšímu popisu bude přistoupeno u servisních tříd, které implementují logiku pokrývající trénink neuronových sítí.

Smyslem třídy TimeSeriesServiceImpl je zaobalovat operace pro práci s časovými řadami.

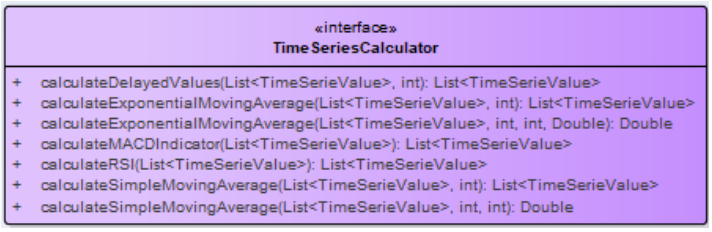


Obrázek 7 - Diagram rozhraní implementovaného třídou TimeSeriesServiceImpl

Vedle klasických operací pro vytvoření, editaci a čtení obsahuje i následující operace:

- calculateDerivedTimeSeries – slouží pro vypočet odvozených časových řad,
- readValuesFromFile – načte samotné hodnoty časové řady z CSV souboru,
- createFromImport – provede vytvoření nové časové řady s hodnotami naimportovanými z CSV souboru,
- prepareValuesMetadata – vytvoří metadata hodnot ke zvoleným časovým řadám. Metadata obsahují minimum a maximum řady, tyto hodnoty jsou dále používány k normalizaci vstupních dat pro neuronovou síť,
- trimMissingDates – před začátkem učení je zkontrolováno, zda hodnoty vybraných vstupních časových řad pokrývají stejné časové okamžiky. Pokud tomu tak není, pak pomocí této operace dojde k tomu, že jsou data k učení očištěna o časové okamžiky, které nejsou přítomny v hodnotách všech časových řad. Výsledkem je tedy časový průnik vstupních řad.

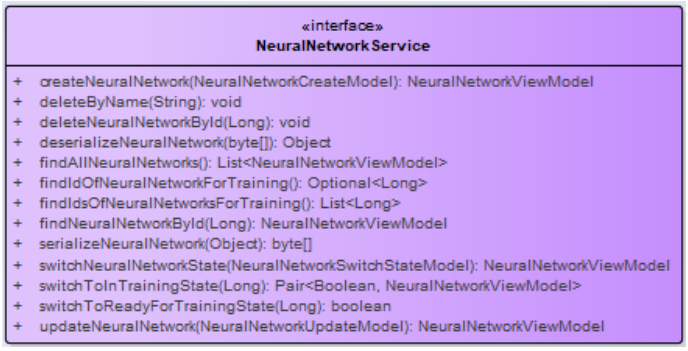
Odvozené časové řady vznikají výpočtem nástrojů technické analýzy.



Obrázek 8 - Diagram tříd rozhraní třídy TimeSeriesCalculatorImpl

Pro tyto výpočty nad časovými řadami byla vytvořena třída TimeSeriesCalculatorImpl s následujícími operacemi:

- calculateDelayedValues – zkopíruje hodnoty časové řady a posune je v čase o určitý počet dní. Tato operace je použita v případě, že uživatel použije jako odvozený typ časové řady zpožděnou hodnotu,
- calculateExponentialMovingAverage – vypočítá hodnoty exponenciálního klouzavého průměru pro časovou řadu s určitou šířkou okna,
- calculateMACDIndicator – na základě hodnot vstupní časové řady vypočítá hodnoty MACD indikátoru,
- calculateRSI – určí hodnoty indexu relativní síly pro hodnoty dané časové řady,
- calculateSimpleMovingAverage – provede výpočet hodnot jednoduchého klouzavého průměru nad danou časovou řadou s určitou délkou okna.



Obrázek 9 - Diagram tříd rozhraní implementovaného třídou NeuralNetworkServiceImpl

Servisní třída NeuralNetworkServiceImpl obsahuje základní operace pro čtení, editaci a vytváření neuronových sítí. Dále jsou v ní definovány následující operace:

- deserializeNeuralNetwork – v databázi jsou vytrénované neuronové sítě uloženy jako binární data. Úkolem této metody je přečtená binární data z databáze převést na instanci neuronové sítě,
- serializeNeuralNetwork – jedná se o inverzní operaci k deserializeNeuralNetwork, převede tedy instanci vytrénované neuronové sítě do binární podoby pro uložení do databáze,
- switchToInTrainingState – přepne neuronovou síť do stavu IN_TRAINING, tato operace je volána před zahájením tréninku.

Jakmile je dokončen trénink neuronové sítě, tak se uživatel může dotázat na návrh obchodního příkazu. Uživatel tedy manuálně zadá hodnoty vstupních časových řad a následně obdrží informaci o tom, co za obchod by vytrénovaná síť v dané situaci vykonala. V servisní třídě TradeProposalService je umístěna operace podporující tuto funkcionalitu. Konkrétně se jedná

o requestTradeProposal, která na základě uživatelských vstupů vrátí zprávu s obchodním doporučením.

Pokud je uživatelem vytvořená instance sítě již finální, pak ji pro zahájení učení může přepnout do stavu připravena k učení. Od této chvíli nemůže uživatel již měnit nastavení sítě. Uvnitř třídy NeuralNetworkJobRunner je nastaven časovač a každou sekundu je prohledána databáze a pokud je nalezena nějaká síť připravená k učení, pak je její trénink zahájen. K zahájení tréninku dojde nejprve tím, že je vytvořena instance třídy NeuralNetworkJob a ta je předána jinému vláknu, které vykoná její metodu run().

```
NeuralNetworkJobData trainingData = neuralNetworkJobDataFactory.createTrainingData(neuralNetworkViewModel);
TraderScore traderScore = getNeuralNetworkJobService().prepareTraderScore(trainingData, neuralTraderParamsViewModel,
        algorithmParametersViewModel);
OptimizationAlgorithm algorithm = getNeuralNetworkJobService().prepareTrainingAlgorithm(neuralNetworkParamsViewModel,
        neuralNetworkViewModel, traderScore, algorithmParametersViewModel);

log(getParametersLogMessage(algorithmParametersViewModel), neuralNetworkViewModel);

for (int i = 0; i < neuralNetworkViewModel.getIterations(); i++) {
    algorithm.iteration();

    MLMethod currentNetwork = algorithm.getMethod();
    validateNeuralTrader(currentNetwork, neuralNetworkJobDataFactory.createValidationData(neuralNetworkViewModel),
            neuralTraderParamsViewModel, algorithmParametersViewModel);
    log(String.format("Iterace #%d Skóre:%d", (i + 1), Math.round(algorithm.getError())), neuralNetworkViewModel);
    if (getNeuralNetworkJobService().isOverfitting(traderScore, getValidationScore(), neuralNetworkParamsViewModel.getNeuralNetworkId(),
            algorithmParametersViewModel.getOverfittingIterationsCheck())){
        break;
    }
    traderScore.incrementIteration();
}

algorithm.finishTraining();
```

Obrázek 10 - Fragment metody run() třídy NeuralNetworkJob

Uvnitř této metody nejprve dojde k načtení vstupních časových řad z databáze a vytvoření odvozených časových řad. Pro trénink a validaci jsou data rozdělena v poměru 80:20. Vytvoří se parametrizovaná instance genetického algoritmu z frameworku Encog. Tomu je předána implementace funkce fitness v třídě NeuralTrader.

```
public int scoreTrader() {
    validateNeuralTrader();

    for (int indexOfValue = 0; indexOfValue < neuralNetworkJobData.getInputValues().get(0).size(); indexOfValue++) {
        MLData input = new BasicMLData(neuralNetworkJobData.getInputValues().size());

        for (int indexOfSeries = 0; indexOfSeries < neuralNetworkJobData.getInputValues().size(); indexOfSeries++) {
            Long singleValue = this.neuralNetworkJobData.getInputValues().get(indexOfSeries).get(indexOfValue).getValue();
            input.setData(indexOfSeries, normalizedFields.get(indexOfSeries).normalize(singleValue));
        }

        MLData output = this.network.compute(input);
        double value = output.getData( index: 0);

        executeTransactionIfProposedByNetwork(value, neuralNetworkJobData.getTradedValues().get(indexOfValue));
    }
    final Long finalTradedTimeSeriePrice = neuralNetworkJobData.getTradedValues().get(neuralNetworkJobData.getTradedValues().size() - 1).getValue();
    final double finalAssetPrice = (assetVolume.doubleValue() * finalTradedTimeSeriePrice.doubleValue());
    if (getNeuralTraderParams().isPrintLogs()) {
        String message = String.format("Na konci simulace má síť %d aktiv, %d na účtě, cena aktiv je: %d, skóre sítě je: %d.",
                assetVolume, Math.round(account), Math.round(finalAssetPrice),
                Math.round(finalAssetPrice + account - getNeuralTraderParams().getInitialAccount()));
        getLogService().log(LOGGING_SOURCE, message, LOGGING_ENTITY_TYPE, neuralTraderParams.getNeuralNetworkId());
    }
    return (int) Math.round((account + finalAssetPrice) - getNeuralTraderParams().getInitialAccount());
}
```

Obrázek 11 - Implementace funkce fitness v metodě scoreTrader() třídy NeuralTrader

Skóre sítě je určeno pomocí backtestingu. To znamená, že jsou historická data časových řad a odvozených časových řad procházena od nejstaršího záznamu po nejnovější. Pro každý takto procházený časový okamžik jsou jemu příslušející data přivedena na vstup neuronové sítě. Jejím výstupem je jedna hodnota, ze které je odvozeno pomocí parametru tradingTreshold, co síť doporučuje. Konkrétně se jedná o vyhodnocení následujících pravidel:

Tabulka 1 - Pravidla pro vyhodnocení výstupu neuronové sítě

Znění pravidla	Doporučená akce
Pokud je absolutní hodnota výstupu větší menší než tradingTreshold	Neprováděj žádný obchod
Výstup je větší než 0	Proveď nákup
Výstup je menší než 0	Proveď prodej

Objem obchodu je určen vynásobením parametru basicTradeAmount a výstupu sítě. Po každé iteraci je provedena validace vytrénovaného modelu. Následně jsou také zkoumána trénovací a validační skóre sítě a pokud dojde k detekování přeučení, tak je namísto přechodu na další iteraci trénink zcela ukončen. V případě detekce přeučení je o tom samozřejmě uživatel informován pomocí zprávy v logu.

Na konci tréninku je ještě jednou provedena funkce fitness pro nejlepší síť s tím, že všechny operace prováděné sítí jsou detailně logovány. Potom je nejlepší síť uložena do atributu body datové třídy NeuralNetwork.

5.2.3 REST API

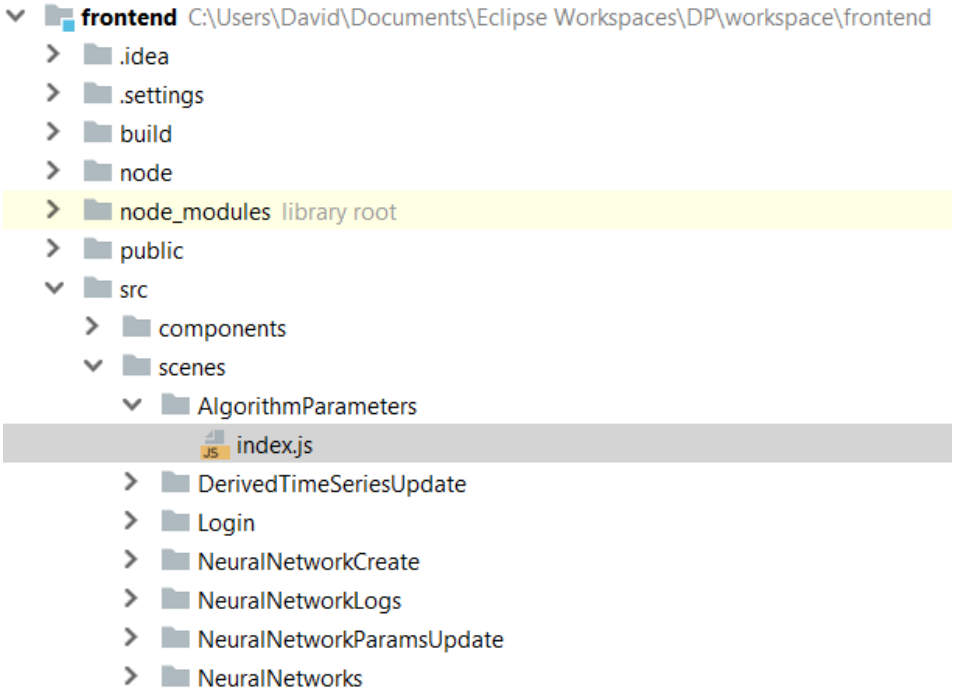
RESTové rozhraní je v serverové části aplikace implementováno pomocí tříd, jejichž název obsahuje příponu resource. Všechny tyto třídy jsou umístěny v balíčku cz.upce.dp.resources a typicky se vztahují k jedné třídě z datové vrstvy a zpřístupňuje pro ni dvě operace. Jedna z nich je umístěna na metodě HTTP protokolu GET a umožňuje klientovi, aby k ID, které v dotazu poskytne, obratem získal data daného záznamu. Druhá typická operace je umístěna na metodě PUT a pomocí jejího volání je provedena editace dat načtených předchozí metodou. NeuralNetworkResource implementuje navíc i metody DELETE pro mazání záznamů a CREATE pro vytvoření záznamu pro novou neuronovou síť a související objekty jako jsou parametry optimalizačního algoritmu a podobně. Jednotlivé třídy implementující REST API jsou v následující tabulce stručně popsány.

Tabulka 2 - Popis jednotlivých tříd implementující REST API

Název třídy, adresa	Popis
AlgorithmParametersResource /api/algorithmParameters	Čtení a úprava parametrů optimalizačního algoritmu
DerivedTimeSeriesResource /api/derivedTimeSeries	Čtení a úprava odvozených vstupních časových řad
LogResource /api/log	Čtení logů, např. zprávy z průběhu učení sítě
NeuralNetworkParamsResource /api/neuralNetworkParams	Čtení a úprava struktury sítě a jejich vrstev
NeuralTraderParamsResource /api/neuralTraderParams	Čtení a úprava parametrů neronového obchodníka
TradeProposalResource /api/tradeProposal	Poskytuje data k vykreslení formuláře obchodního doporučení, vrací zprávu obchodního doporučení
TrainingScoreResource /api/trainingScore	Čtení statistik učení sítě pro jednotlivé generace
UserInfoResource /user-info	Čtení informací o přihlášeném uživateli
NeuralNetworkResource /api/neuralNetwork	Vytvoření, mazání, editace a čtení instance neuronové sítě
NeuralNetworkResource /api/neuralNetwork/ switchState	Přepnutí stavu sítě

5.2.4 Klientská část aplikace

Jak bylo již zmíněno na začátku kapitoly věnující se implementaci, tak klientská část aplikace je vytvořena za použití knihovny React.js. Pro tuto single page aplikaci je vstupním bodem soubor index.js, který do kořenového elementu stránky vykreslí instance třídy App. Tato třída je již klasická komponenta pro React.js obsahující funkci render, pomocí které vrací elementy pro virtuální DOM. Třída App využívá knihovnu React router pro přepínání mezi jednotlivými obrazovkami aplikace. Každá obrazovka má svůj podadresář v adresáři src/scenes obsahující soubor index.js, jehož funkce render danou obrazovku vykresluje.



Obrázek 12 - Adresářová struktura klientské části aplikace

Kostru obrazovky mají všechny obrazovky společnou, pro vykreslování využívají tedy jednotlivé obrazovky sdílenou komponentu Layout, která kostru stránky vykreslí. Fungování obrazovky klientské části aplikace sestává typicky z následujících kroků:

- Po přistoupení uživatele na danou stránku jsou ve funkci componentDidMount() načtena data přes REST API ze serverové části a nastavena do state aplikace, následně dojde k překreslení stránky pomocí přijatých dat,
- uživatelské změny hodnot v polích na stránce jsou propagovány do state aplikace,
- po odeslání formuláře jsou data ze state odeslána opět do serverové části aplikace pomocí metody POST, pokud akcí má být vytvořen nový záznam nebo pomocí metody PUT pokud se jedná pouze o aktualizaci existujícího záznamu v aplikaci,

- následně serverová část provede validaci odeslaného formuláře a vrátí buďto chybovou hlášku anebo zprávu o úspěšném provedení operace,
- klientská část přijatou zprávu ze serveru zobrazí.

Na některých obrazovkách aplikace bylo třeba prezentovat data vizuálně pomocí vykreslování grafů. K tomuto účelu byla použita knihovna Recharts. V aplikaci se také nachází několik obrazovek vykreslujících stránkované tabulkové seznamy. Tyto seznamy byly vytvořeny za pomocí knihovny React-table. Výše zmíněné obrazovky využívající knihovny Recharts a React-table budou představeny jako součást následující kapitoly věnované představení hotové aplikace.

Vizuální stránku aplikace zabezpečuje volně dostupná šablona postavená na open source CSS knihovně Bootstrap 3.

6 Představení hotové aplikace

V následující kapitole bude představena vytvořená aplikace z uživatelského pohledu. Na místo prostého popisu jednotlivých obrazovek budou funkce aplikace vysvětleny na praktické ukázce. V rámci této ukázky budou provedeny následující testovací scénáře:

- import dat k učení do aplikace,
- vytvoření a nakonfigurovaní instance neuronové sítě,
- spuštění optimalizace sítě,
- prohlížení průběhu učení,
- použití vytrénované sítě k dotazu na návrh obchodního příkazu.

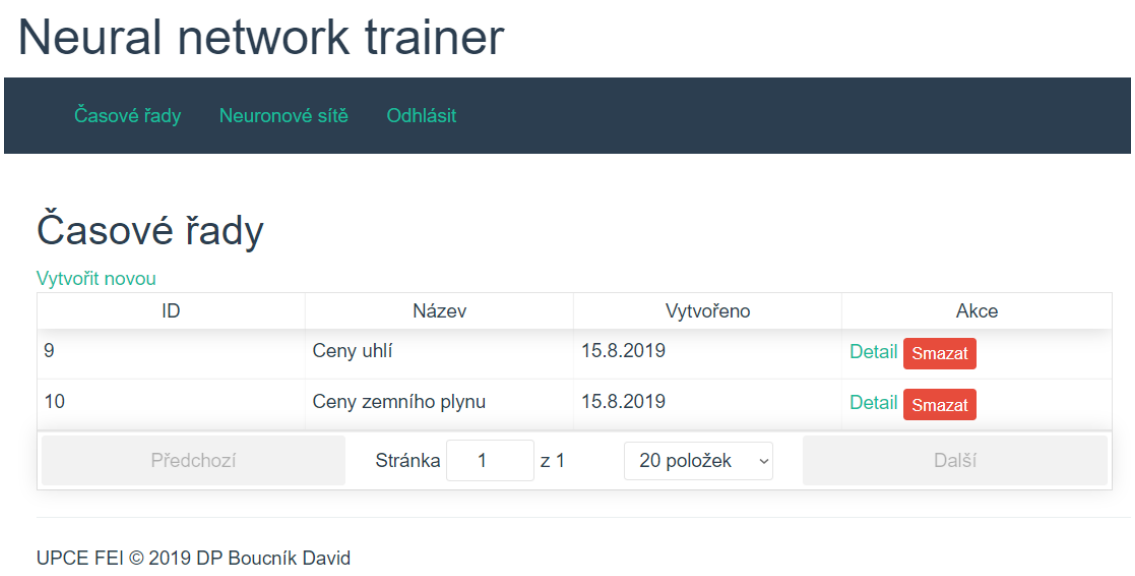
Jednotlivé kroky ukázky budou doplněny také o obrázky obrazovek. Návod k instalaci a spuštění aplikace je vložen do příloh.

6.1 Uživatelská příručka

Prvním krokem v používání aplikace je typicky nahrání dat k učení do aplikace. V tomto příkladu byla použita data volně dostupná na webu kurzy.cz. Jedná se konkrétně o zavírací denní ceny z následujících komoditních trhů:

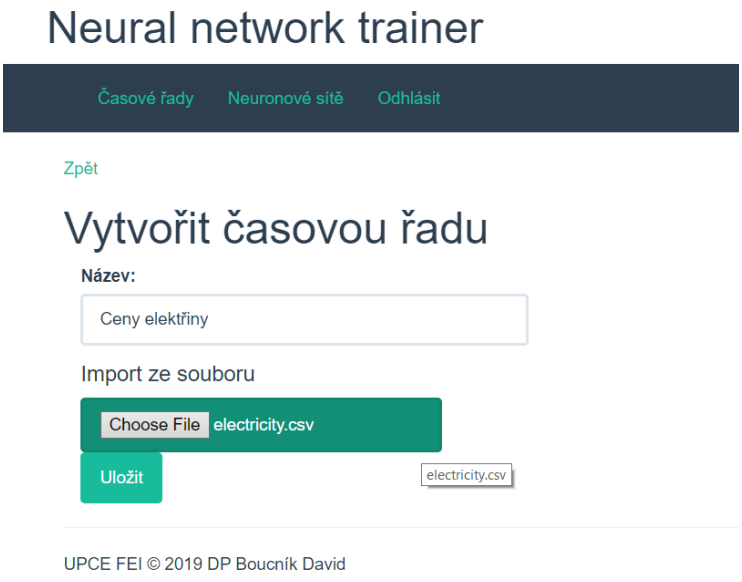
- zemní plyn,
- elektřina.

Výše uvedená data byla manuálně vykopírována z webové stránky kurzy.cz, vložena do CSV dokumentu a následně naimportována. Samotný import dat je proveden tak, že po přihlášení je třeba kliknout na tlačítko „Časové řady“ v horním panelu aplikace. Uživateli se následně vykreslí seznam již existujících časových řad v aplikaci. Nad levým horním rohem tabulky s časovými řadami je odkaz „Vytvořit novou“.



Obrázek 13 - Část obrazovky výpisu existujících časových řad

Tento odkaz zavede uživatele na formulář pro vytvoření nové časové řady pomocí importu.



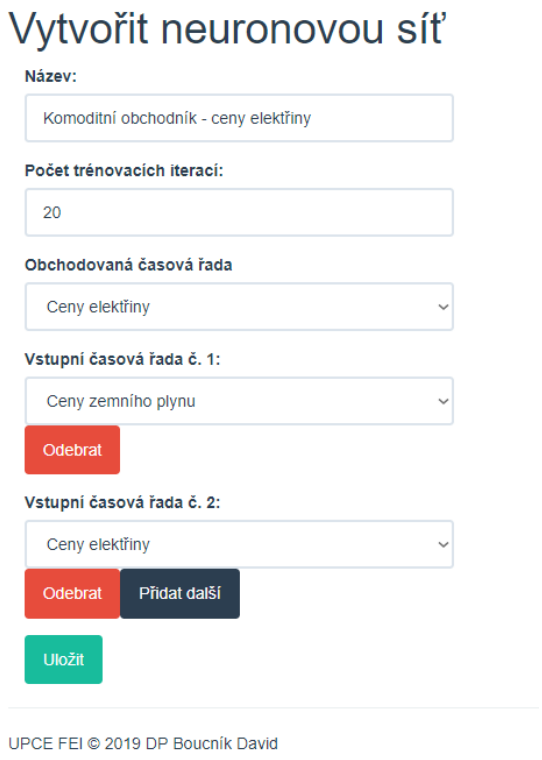
Obrázek 14 - Formulář pro vytvoření nové časové řady importem

Importovaný CSV soubor se zdrojovými daty by měl obsahovat dva sloupce:

- Datum ve formátu „dd.MM.rrrr“,
- Desetinné číslo představující hodnotu časové řady pro daný datum.

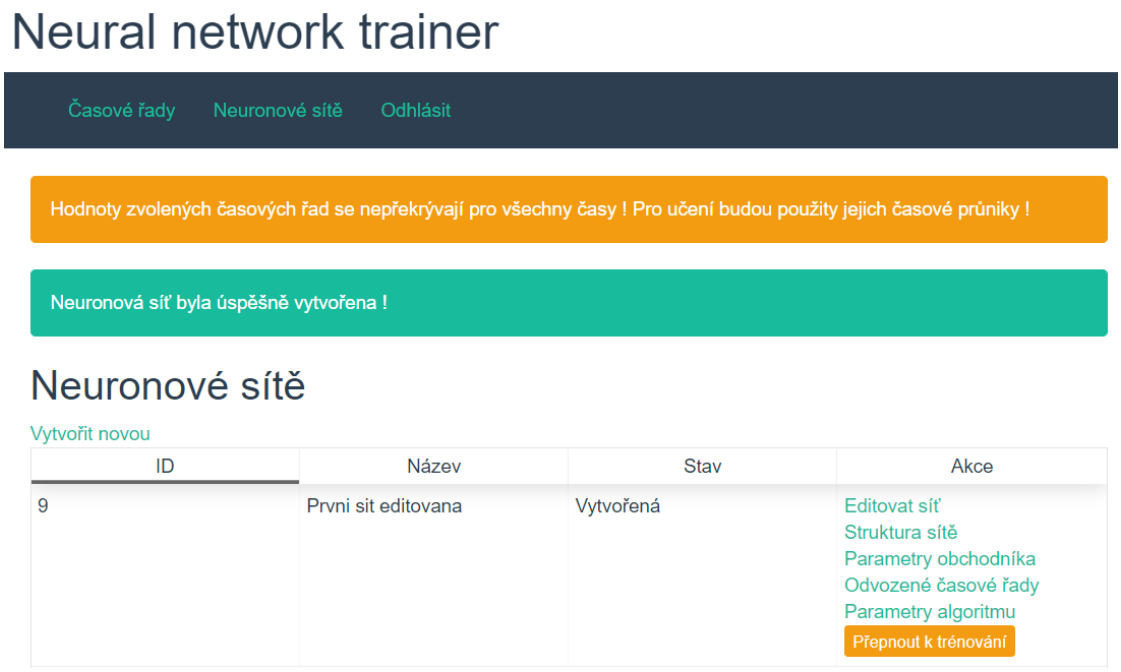
Zde uživatel zadá název nové časové řady, vybere CSV soubor obsahující data a následně stiskne tlačítko „Uložit“, čímž vše potvrdí a odešle formulář. Uživateli je poté zobrazena zpráva o dokončení importu. Tímto způsobem byla do aplikace nahrána potřebná historická data

a následně bylo přikročeno k dalšímu kroku. Tímto krokem je vytvoření samotné instance neuronové sítě. Stisknutím tlačítka „Neuronové sítě“ dojde k přesměrování do výpisu existujících neuronových sítí v aplikaci. Nad levým horním rohem seznamu neuronových sítí je k vidění odkaz „Vytvořit novou“ pro vstup do formuláře k vytvoření nové instance neuronové sítě.



Obrázek 15 - Formulář pro vytvoření nové instance neuronové sítě v aplikaci

Název neuronové sítě by zvolen „Komoditní obchodník - ceny elektřiny“, což popisuje její určení. Počet trénovacích iterací byl nastaven na 20. Jako časová řada, na které bude síť provádět obchody, byla zvolena „Ceny elektřiny“. Vstupní časové řady představují časové řady, které budou použity jako přímé vstupy do neuronové sítě. V tomto případě byly jako vstupní časové řady zvoleny historické ceny zemního plynu a elektřiny. Pro zvolení právě těchto časových řad jako vstupů je přistoupeno na základě toho, že se jedná o substituenty tzn. produkty, které jsou z hlediska poptávky zaměnitelné. Z tohoto důvodu se dá předpokládat, že jejich vývoj cen na sebe vzájemně působí. Díky těmto vstupům by mohla být tedy nalezená obchodní strategie sítě kvalitnější.



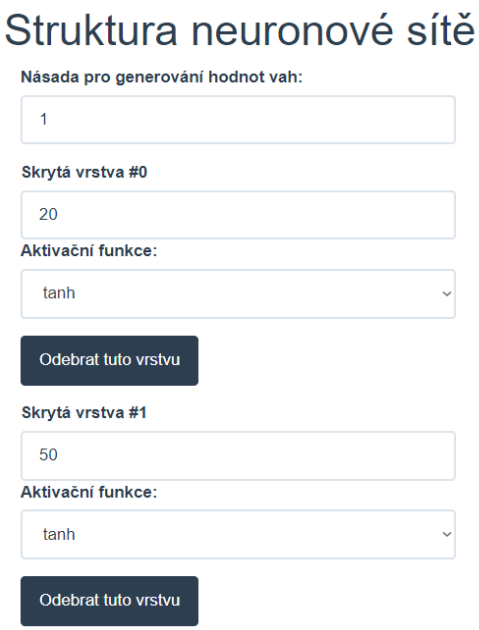
Obrázek 16 - Upozornění při vytvoření časové řady za použití časově nepřekrývajících se časových řad

Po odeslání formuláře dojde k vytvoření nové instance neuronové sítě. Pokud bylo k vytvoření neuronové sítě použito časových řad, které se časově stoprocentně nepřekrývají, tak dojde k následujícímu:

- uživateli se po odeslání formuláře k vytvoření neuronové sítě zobrazí upozornění,
- pro učení bude ze zvolených časových řad vybrán jejich časový průnik.

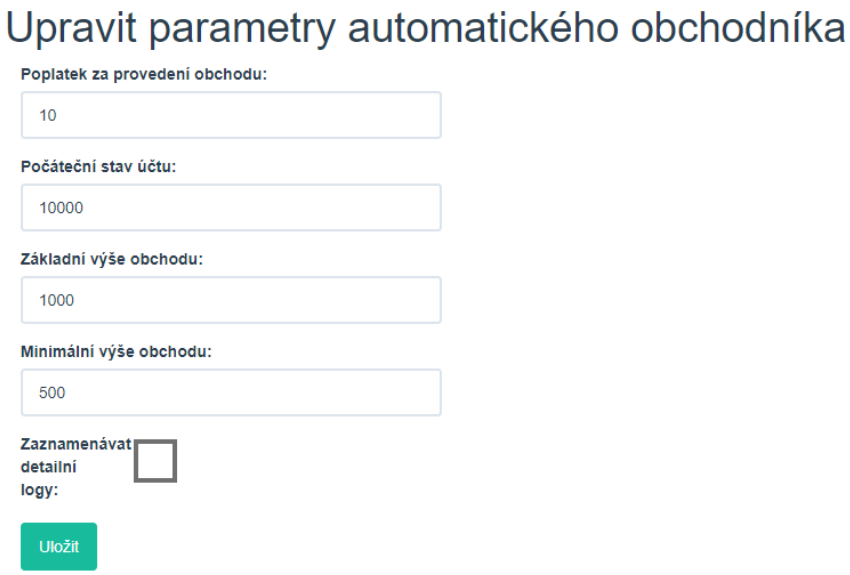
S vytvořením nové instance neuronové sítě dojde k tomu, že byly vytvořeny i objekty související s neuronovou sítí jako je například definice struktury sítě, parametry optimalizačního algoritmu a další. Vlastnosti těchto objektů jsou nastaveny na výchozí hodnoty. Uživatel tedy nemusí vždy při vytvoření instance sítě nastavovat zdlouhavě všechny volby manuálně, místo toho pouze změní volby, kde výchozí hodnoty nevyhovují danému účelu.

Schopnost neuronové sítě adaptovat se na daný problém, je do značné míry ovlivněna její strukturou. Tato struktura je reprezentována různým počtem vrstev, každá z nich s různým počtem umělých neuronů.



Obrázek 17 - Formulář pro nastavení struktury neuronové sítě

K nastavení struktury sítě se uživatel dostane pomocí odkazu „Struktura sítě“ v seznamu neuronových sítí. Před optimalizací sítě jsou její hodnoty vygenerovány náhodně. Generování hodnot je provedeno pomocí prostředků jazyka Java s násadou volitelnou polem v tomto formuláři. Hodnota násady byla ponechána ve výchozím stavu s hodnotou 1. Pro jednotlivé skryté vrstvy sítě byla vždy použita aktivační funkce hyperbolický tangens. Rozměry skrytých vrstev byly definovány v pořadí od nejnižší po nejvyšší vzdálenosti od vstupní vrstvy jako hodnoty: 20, 50, 50, 10.



Obrázek 18 - Formulář pro úpravu parametrů obchodníka

Během vyhodnocování funkce fitness neuronová síť zadává obchodní příkazy. Chování sítě při zadávání těchto příkazů lze ovlivnit nastavením voleb ve formuláři pro úpravu parametrů automatického obchodníka. Na tento formulář se uživatel dostane odkazem „Parametry obchodníka“ ve výpisu sítí. Jednotky hodnot nastavovaných v tomto formuláři odpovídají jednotkám v importovaném CSV souboru obchodované časové řady, v tomto případě se jedná o Americký dolar.

Během skutečného obchodování na burze jsou obchodní příkazy podávány prostřednictvím tzv. brokera. Tento prostředník za svoje služby dostává zaplaceno obvykle formou poplatků. Po zadání obchodního příkazu sítí ve funkci fitness je z jejího účtu stržena také částka, která by v reálu odpovídala poplatku brokerovi. V popisovaném příkladě byla jako poplatek za provedení obchodu zvolena hodnota 10 Amerických dolarů.

Pole „počáteční stav účtu“ nastavuje, s jak velkým finančním obnosem neuronová síť ve funkci fitness začíná. Pro tento účel byla použita hodnota 10 000 Amerických dolarů.

Výstupem neuronové sítě je v aplikaci jedno jediné číslo v intervalu mezi -1 a 1. Pro určování výše obchodu je tento výstup vynásoben konstantou zadanou uživatelem v aplikaci do pole „Základní výše obchodu“. Zde bylo použito hodnoty 1000 Amerických dolarů.

Aby byl obchodní příkaz sítí proveden, musí být také vyšší než je uživatelem zadaná „Minimální výše obchodu“, jež byla nastavena na 500 Amerických dolarů.

Obrázek 19 - Formulář pro nastavení odvozených časových řad

Hodnoty vstupních časových řad nejsou jediným vstupem sítě. Dalším neméně důležitým vstupem jsou odvozené časové řady, které vznikají pro potřeby učení pomocí odvození z časových řad existujících v aplikaci. Aplikace podporuje následující způsoby odvození:

- klouzavý průměr,
- exponenciální klouzavý průměr
- MACD indikátor,
- index relativní síly (RSI),
- zpožděná hodnota.

Pro některé typy odvození je možné také nadefinovat i šířku okna, která je typicky použitelná u klouzavých průměrů jako informace o tom, nad kolika hodnotami se má průměr počítat. V popisovaném příkladu jsou použity odvozené časové řady uvedené v následující tabulce:

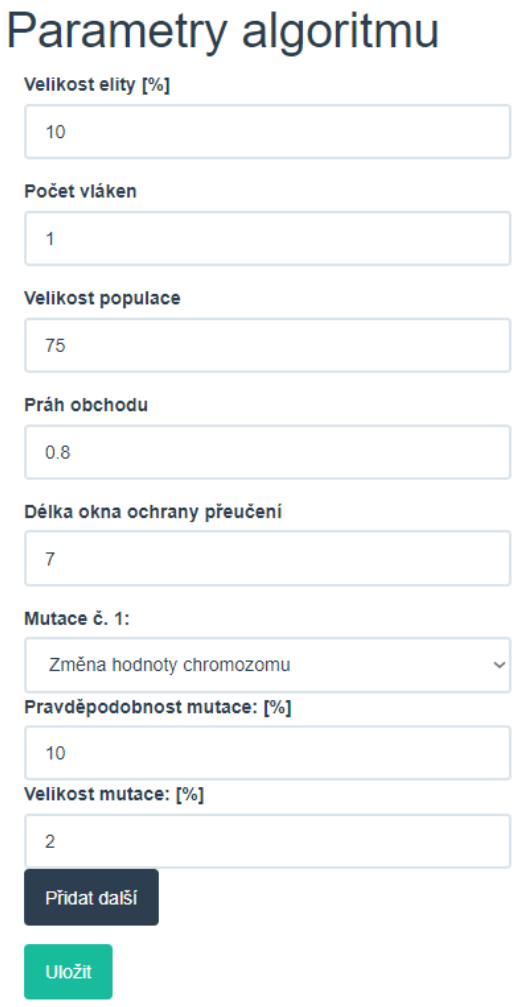
Tabulka 3 - Nastavené volby odvozených časových řad

Zdrojová časová řada	Způsob odvození	Šířka okna
Ceny zemního plynu	Klouzavý průměr	30
Ceny elektřiny	Klouzavý průměr	30
Ceny zemního plynu	Exponenciální klouzavý průměr	60
Ceny elektřiny	Exponenciální klouzavý průměr	60
Ceny zemního plynu	Klouzavý průměr	90
Ceny elektřiny	Klouzavý průměr	90

Optimalizace hodnot vah sítě je prováděna pomocí genetického algoritmu. Tento algoritmus má řadu parametrů, které může uživatel měnit. Na formulář pro změnu parametrů algoritmu se uživatel dostane přes odkaz „Parametry algoritmu“ v seznamu sítí. Hodnoty těchto parametrů byly nastaveny na tyto konkrétní hodnoty:

- Velikost elity – procentuální vyjádření velikosti skupiny považovaných za elitu generace, nastaveno na hodnotu 10.
- Počet vláken – počet vláken ve kterých má být učení této sítě spuštěno, nastaveno na 1.
- Velikost populace – počet jedinců v jedné generaci genetického algoritmu, nastaveno na 75.

- Práh obchodu – výstupem sítě je vždy jedno desetinné číslo v intervalu od -1 do 1. Pokud absolutní hodnota tohoto čísla překračuje práh obchodu, pak je to identifikováno jako pokyn k provedení obchodu. V případě, že je výstup kladný, tak se výstup interpretuje jako nákup, záporný výstup je interpretován jako prodej. Ponechána výchozí hodnota 0,8.
- Délka okna ochrany přeučení – v každé iteraci jsou porovnány výsledky sítě na trénovací a validační množině dat pro detekci přeučení. Délka okna ochrany přeučení udává, kolik iterací je při této kontrole bráno v potaz. Použita byla výchozí volba 7.
- Mutace – seznam mutací aplikovaných při křížení. Pravděpodobnost mutace udává, s jakou pravděpodobností bude mutace na geny aplikována. Velikost mutace definuje procentuální míru pozměnění genu při mutaci. Lze zvolit buďto mutaci pozměňující hodnoty genů anebo upravující pořadí genů. Zde byla beze změny ponechána výchozí hodnota pro mutaci změnou chromozomů s pravděpodobností 10 % a velikostí 2 %.



Obrázek 20 - Formulář pro nastavení parametrů algoritmu

Po nastavení parametrů algoritmu byla síť přepnuta k trénování pomocí tlačítka „Přepnout k trénování“ v seznamu sítí. Od této chvíle již není možné síť editovat. Na místo toho je na pozadí aplikace spuštěna optimalizace sítě. Uživatel má v tuto chvíli možnost sledovat stav učení několika způsoby.

Ve výpisu sítí může sledovat stav v jakém se síť dále nachází:

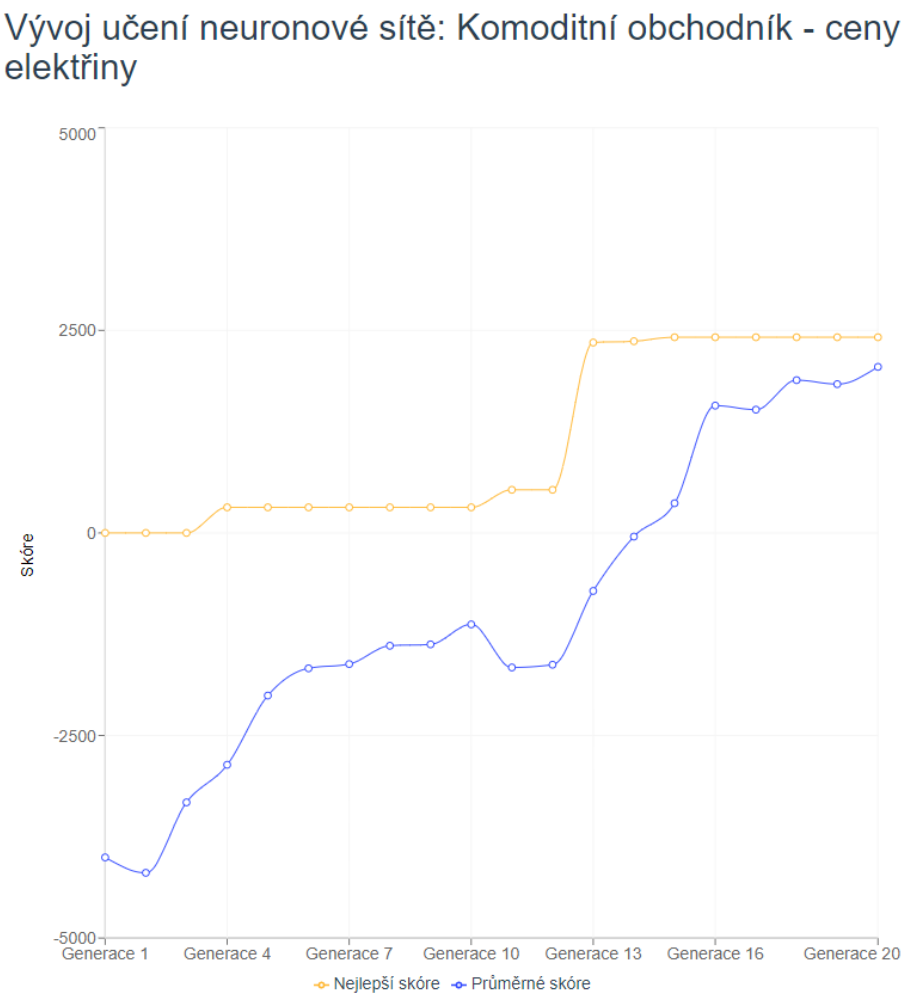
- Připravena k učení – signalizuje, že síť byla uživatelem přepnuta k učení, ale učení ještě nebylo zahájeno.
- V procesu učení – učení sítě již bylo zahájeno, ale zatím není dokončeno.
- Trénink dokončen – učení je dokončeno, síť je připravena k použití.

Pomocí odkazu „Log neuronové sítě“ v seznamu sítí se může uživatel dostat na obrazovku zobrazující seznam všech zpráv z učení. Tyto zprávy obsahují informace o:

- zahájení tréninku,
- použitých parametrech,
- výsledcích v jednotlivých iteracích,
- dokončení tréninku,
- detailním chování optimalizované sítě na trénovacích datech.

Obrázek 21 - Výpis zpráv z tréninku neuronové sítě

Ve chvíli, kdy je trénink sítě dokončen, může uživatel přejít pomocí odkazu „Průběh učení“ na stránku, kde je graficky zobrazen průběh učení. Na této obrazovce je vykreslen graf obsahující informace o hodnotě fitness nejlepšího a průměrného jedince v každé iteraci algoritmu.



Obrázek 22 - Grafická vizualizace průběhu optimalizace neuronové sítě

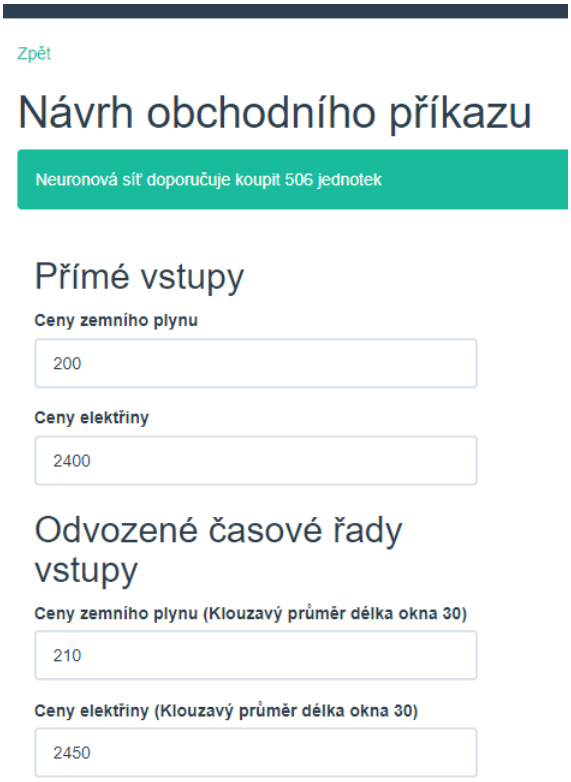
Již vytrénované sítě se lze také dotázat na návrh obchodního příkazu. K této funkcionalitě se uživatel dostane přes odkaz „Návrh obchodního příkazu“ v seznamu neuronových sítí. Do formuláře této funkcionality lze zadat hodnoty časových řad a odvozených časových řad manuálně. Tyto hodnoty jsou po odeslání formuláře přivedeny na vstup sítě. Výstup sítě je uživateli obratem prezentován ve formě textového návrhu obchodního příkazu.

Pro dotaz na návrh obchodního příkazu vytrénované sítě byly použity vstupy zdokumentované v následující tabulce (Tabulka 4).

Tabulka 4 - Manuálně vložené vstupy neuronové sítě pro návrh obchodního příkazu

Název stupu	Hodnota
Ceny zemního plynu	200
Ceny elektřiny	2400
Ceny zemního plynu (Klouzavý průměr délka okna 30)	210
Ceny elektřiny (Klouzavý průměr délka okna 30)	2450
Ceny zemního plynu (Klouzavý průměr délka okna 60)	220
Ceny elektřiny (Klouzavý průměr délka okna 60)	2500
Ceny zemního plynu (Klouzavý průměr délka okna 90)	230
Ceny elektřiny (Klouzavý průměr délka okna 90)	2550
Hodnota obchodované řady	200

Jak je z uvedené tabulky (Tabulka 4) patrné, tak hodnoty vstupů jsou voleny pro vytvoření dojmu, že vývoj cen je právě na sestupu.



Obrázek 23 - Formulář pro dotaz na návrh obchodního příkazu

Tento stav je sítí vyhodnocen jako vhodná dobu k nákupu, její výstup je interpretován jako pokyn k nákupu 506 jednotek elektřiny.

Závěr

Teoretická část práce pokryla rešerši známých řešení pro inteligentní automatické obchodování. Představeno bylo několik projektů komerčně úspěšných a vyvíjených v různých částech světa. V teoretické části aplikace byly také představeny aktuálně populární frameworky pro tvorbu webových aplikací na velmi rozšířené platformě Java. Většina z popsaných webových frameworků má již dlouholetou historii a řadu uživatelů. Dále byl čtenář v rámci teoretická části práce uveden do problematiky aktuálně populárních Java frameworků pro práci s neuronovými sítěmi. Některé z těchto frameworků vynikají schopností využít potencionál hardwaru, jiné především snadností použití. Poznatky z teoretické části práce byly následně zúročeny při rozhodování o použití technologií v praktické části.

Během praktické části práce byla provedena analýza uživatelských požadavků na aplikaci. Následovala implementace webové aplikace, v níž byl použit framework Spring Boot. Pro práci s neuronovými sítěmi byl pro svou snadnost použití zvolen Encog. Během implementace jednotlivých funkcionalit však vyšlo najevo, že framework Encog ne ve všech směrech zcela vyhovuje (např. vytvoření sítě s aktivačními funkcemi pro každou vrstvu zvlášť nebylo zcela podporováno). Pro řešení optimalizace sítě byl použit genetický algoritmus, který se na použitých datech projevil jako vyhovující. Je nutné ale také podotknout, že existují optimalizační algoritmy (např. diferenciální evoluce), které by mohly na určitých datech dospět k o něco lepším výsledkům.

Diplomová práce splnila požadavky kladené na teoretickou a praktickou část práce. Vzhledem k tomu, že vývoj komerčně úspěšných aplikací je zpravidla v rukou několika profesionálních týmů, tak se nedá očekávat, že by praktická část této práce mohla dosahovat podobné úrovně. Jako možné rozšíření aplikace, které by vedlo ke zvýšení její použitelnosti, se jeví přidání optimalizačního modulu. Tento modul by na základě uživatelem definovaných pravidel upravoval strukturu existující sítě a hledal její optimální strukturu.

Vyvinutou aplikaci lze využívat jako podporu pro rozhodování při správě osobních investic. Při tomto použití je ale zcela nezbytné mít neustále na paměti fakt, že obchodní strategie vedoucí k zisku na historických datech nemusí vždy vést k zisku také v budoucnosti.

Tato práce může být zdrojem informací a inspirace jak pro čtenáře zabývajícího se vývojem podobné aplikace, tak i aplikace, která je postavena na technologiích, o kterých tato práce pojednává.

Literatura

[1] PIT.AI TECHNOLOGIES, INC., MINING TRADING STRATEGIES WITH AI [Online] 2018. [Citace: 24. března 2018] https://pit.ai

[2] ALPACAJAPAN CO., LTD., AUTOMATE YOUR WINNING TRADE IDEAS WITH AN ALGORITHM BUILT JUST FOR YOU [Online] 2018. [Citace: 30. března 2018] https://www.alpaca.ai

[3] ALPACAJAPAN CO., LTD., ALPACAALGO [Online] 2018. [Citace: 30. března 2018] https://www.alpaca.ai/alpacaalgo/

[4] ALPACAJAPAN CO., LTD., OUR CHALLENGE IS MARKET FORECASTING [Online] 2018. [Citace: 30. března 2018] https://www.alpaca.ai/alpacaforecast/

[5] SEVARAC, Zoran**,** JAVA NEURAL NETWORK NEUROPH [Online] 2018. [Citace: 16. dubna 2018] http://neuroph.sourceforge.net/index.html

[6] SEVARAC, Zoran, JAVA NEURAL NETWORK NEUROPH : ABOUT NEUROPH PROJECT [Online] 2018. [Citace: 16. dubna 2018] http://neuroph.sourceforge.net/about_project.html

[7] SEVARAC, Zoran, JAVA NEURAL NETWORK NEUROPH : FEATURES [Online] 2018. [Citace: 16. dubna 2018] http://neuroph.sourceforge.net/features.html

[8] SKYMIND, DEEP LEARNING FOR JAVA: OPEN-SOURCE, DISTRIBUTED, DEEP LEARNING LIBRARY FOR THE JVM [Online] 2018. [Citace: 5. května 2018] https://deeplearning4j.org

[9] SKYMIND, CORE CONCEPTS IN DEEPLEARNING4J [Online] 2018. [Citace: 8. listopadu 2018] https://deeplearning4j.org/docs/latest/deeplearning4j-concepts

[10] SHELAJEV, Oleg**,** JAVA WEB FRAMEWORKS INDEX: FEBRUARY 2017 [Online] 2018. [Citace: 25. února 2019] https://zeroturnaround.com/rebellabs/java-web-frameworks-index-by-rebellabs/

[11] ORACLE, JAVASERVER FACES TECHNOLOGY OVERVIEW [Online] 2019. [Citace: 2. března 2019] https://www.oracle.com/technetwork/java/javaee/overview-140548.html

[12] GOOGLE, GWT PROJECT OVERVIEW [Online] 2019. [Citace: 5. března 2019] http://www.gwtproject.org/overview.html

[13] PIVOTAL SOFTWARE, SPRING FRAMEWORK [Online] 2019. [Citace: 16. března 2019] https://spring.io/projects/spring-framework

[14] PIVOTAL SOFTWARE, SPRING BOOT [Online] 2019 [Citace: 16. března 2019] https://spring.io/projects/spring-boot

[15] PIVOTAL SOFTWARE, WEB ON SERVLET STACK [Online] 2019. [Citace: 16. března 2019] https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html

[16] CB INFORMATION SERVICES, INC., THE AI IN FINTECH MARKET MAP: 100+ COMPANIES USING AI ALGORITHMS TO IMPROVE THE FIN SERVICES INDUSTRY [Online] 2017. [Citace: 16. března 2019] https://www.cbinsights.com/research/ai-fintech-startup-market-map/

[17] UHEREK, Jiří, ALGORITMICKÉ OBCHODOVÁNÍ [Online] 2014. [Citace: 25. března 2019] https://vskp.vse.cz/44151_algoritmicke_obchodovani

[18] ALGORIZ, INC., ALGORIZ - ALGORITHMIC TRADING - AUTOMATE STOCK BUYING [Online] 2019. [Citace: 25. března 2019] https://algoriz.com

[19] HEATON RESEARCH, INC., ALGOENCOG MACHINE LEARNING FRAMEWORK [Online] 2019. [Citace: 25. května 2019]https://www.heatonresearch.com/encog/

[20] SMILE, SMILE - STATISTICAL MACHINE INTELLIGENCE AND LEARNING ENGINE [Online] 2019. [Citace: 26. května 2019] http://haifengl.github.io/smile/index.html

[21] SKYMIND, ARTIFICIAL INTELLIGENCE (AI) FOR JAVA [Online] 2019. [Citace: 28. května 2019] https://deeplearning4j.org/cn/java-ai.html

[22] HAYKIN, Simon. Neural Networks: A Comprehensive Foundation. 2nd edition. New Jersey, USA: Prentice Hall, 1998. ISBN 978-0132733502.

[23] SKYMIND, ARTIFICIAL INTELLIGENCE (AI) VS. MACHINE LEARNING VS. DEEP LEARNING [Online] 2019. [Citace: 20. července 2019] https://skymind.ai/wiki/ai-vs-machine-learning-vs-deep-learning

[24] INVESTOPEDIA, AR TECHNICAL ANALYSIS BASIC EDUCATION [Online] 2019. [Citace: 20. července 2019] https://www.investopedia.com/terms/b/backtesting.asp

[25] INVESTOPEDIA, Algorithmic Trading [Online] 2019. [Citace: 20. července 2019] https://www.investopedia.com/terms/a/algorithmictrading.asp

[26] INVESTOPEDIA, QUANTITATIVE TRADING DEFINITION [Online] 2019. [Citace: 20. července 2019] https://www.investopedia.com/terms/q/quantitative-trading.asp

[27] ORACLE, About the Java Technology [Online] 2019. [Citace: 20. července 2019] https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html

[28] WASSON, Mike, ASP.NET - Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET [Online] 2019. [Citace: 8. srpna 2019] https://msdn.microsoft.com/en-us/magazine/dn463786.aspx

[29] ORACLE CORPORATION AND/OR ITS AFFILIATES, 1.3.1 What is MySQL? [Online] 2019. [Citace: 8. srpna 2019] https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html

[30] HIBERNATE, What is Object/Relational Mapping? [Online] 2019. [Citace: 8. srpna 2019] https://hibernate.org/orm/what-is-an-orm/

[31] HIBERNATE, Your relational data. Objectively. [Online] 2019. [Citace: 8. srpna 2019] http://hibernate.org/orm/

[32] FACEBOOK INC, React - A JavaScript library for building user interfaces. [Online] 2019. [Citace: 8. srpna 2019] https://reactjs.org

[33] REFSNES DATA, What is JavaScript? [Online] 2019. [Citace: 8. srpna 2019] https://www.w3schools.com/whatis/whatis_js.asp

[34] PIVOTAL SOFTWARE INC, Building REST services with Spring [Online] 2019. [Citace: 8. srpna 2019] https://spring.io/guides/tutorials/rest/

[35] PATRIA FINANCE, A.S., Technická analýza [Online] 2019. [Citace: 8. srpna 2019] https://www.patria.cz/slovnik/84/technicka-analyza.html

[36] PATRIA FINANCE, A.S., Klouzavý průměr - Exponenciální [Online] 2019. [Citace: 8. srpna 2019] https://www.patria.cz/slovnik/391/klouzavy-prumer---exponencialni.html

[37] PATRIA FINANCE, A.S., MACD [Online] 2019. [Citace: 8. srpna 2019] https://www.patria.cz/slovnik/389/macd.html

[38] PATRIA FINANCE, A.S., Index relativní síly [Online] 2019. [Citace: 8. srpna 2019] https://www.patria.cz/slovnik/387/index-relativni-sily.html

[39] DOTDASH PUBLISHING FAMILY, Simple Moving Average - SMA Definition [Online] 2019. [Citace: 10. srpna 2019] https://www.investopedia.com/terms/s/sma.asp

[40] HOLLAND, John H. Genetic Algorithms and Adaptation. 1. Boston, MA, USA: Springer, 1984. ISBN 978-1-4684-8941-5.

[41] REFSNES DATA, What is the HTML DOM? [Online] 2019. [Citace: 10. srpna 2019] https://www.w3schools.com/whatis/whatis_htmldom.asp

[42] ORACLE, The Java EE 6 Tutorial: Introduction to the Java Persistence API [Online] 2019. [Citace: 10. srpna 2019] https://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html

[43] THE APACHE SOFTWARE FOUNDATION, Maven – Introduction [Online] 2019. [Citace: 21. srpna 2019] https://maven.apache.org/what-is-maven.html

[44] BOOTSTRAP TEAM, Bootstrap · The most popular HTML, CSS, and JS library in the world. [Online] 2019. [Citace: 21. srpna 2019] https://getbootstrap.com

Seznam příloh

Příloha A – *Instalace a spuštění aplikace*

Pro úspěšné spuštění aplikace musí být na daném počítači předem nainstalován následující software:

- JDK 8,
- NPM ve verzi 5.6.0,
- Yarn Package Manager ve verzi 1.9.4,
- MySQL,
- Maven verze 3.5.3.

Následně je potřeba nakonfigurovat aplikaci před prvním spuštěním v souboru „workspace\backend\src\main\resources\application.properties“ nastavit:

- „spring.datasource.url“ na adresu existující databáze ve spuštěné instanci serveru MySQL,
- „spring.datasource.username“ na uživatelské jméno aplikace pro přihlášení do databázového serveru,
- „spring.datasource.password“ na heslo pro přihlášení do databázového serveru,
- „spring.jpa.hibernate.ddl-auto“ na „update“,
- „cz.upce.dp.user.name“ na uživatelské jméno pro přihlášení do aplikace,
- „cz.upce.dp.user.password“ na uživatelské heslo pro přihlášení do aplikace (tuto hodnotu je dobré po vytvoření iniciálních dat z bezpečnostních důvodu z konfiguračního souboru smazat).

Pro spuštění aplikace je třeba:

- přejít do adresáře „workspace/backend“,
- spustit „mvn clena install -Pinit-data -DskipTests=true“ pro vytvoření databázového schématu s iniciálními daty (pouze pro první spuštění),
- zavolat „spring-boot:run“ pro spuštění REST API
- přejít do adresáře „workspace/frontend“
- zavolat „yarn start“ pro spuštění webového serveru s klientskou částí aplikace

Spuštění všech částí aplikace by mělo být hotovo během několika minut, následně bude možné aplikaci začít používat na adrese „http://localhost:3000/login“.

Příloha B – *CD*

Obsah CD:

BoucnikDavidDP – Zdrojové kódy aplikace

Boucnik_david_dp.pdf – Vlastní text práce