VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Webová aplikace: Vizualizace chování trhu

Web Application: Visualization of Market Behavior

2020 Jan Bubik

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jan Bubik**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Webová aplikace: Vizualizace chování trhu
Web Application: Visualization of Market Behavior

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je navrhnout a naimplementovat webovou aplikaci/komponentu, která umožní analyzovat a vizualizovat data vybraného trhu. Součástí práce bude rovněž jednoduchá správa dat, exporty a importy v JSON formátu. Předpokládá se využití následujících technologií a jazyků: React, Redux, TypeScript.

Body zadání:
1. Představení vybrané technologie, jejich vlastností a přínosu pro řešení.
2. Návrh a implementace aplikace.
4. Vytvoření experimentů a provedení testů s ohledem na vybranou aplikační doménu.
5. Zhodnocení experimentů a dosažených výsledků.

Seznam doporučené odborné literatury:

[1] M. Pilgrim: HTML5: Up and Running, O'Reilly Media; 1 edition (August 24, 2010), ISBN-13: 978-0596806026
[2] D. Crockford: JavaScript: The Good Parts, O'Reilly Media; 1st edition (May 2008), ISBN-13: 978-0596517748

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Ing. Petr Gajdoš, Ph.D.**

Datum zadání: 01.09.2019
Datum odevzdání: 30.04.2020

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 5. února 2020 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 5. února 2020

.....

Rád bych poděkoval doc. Ing. Petru Gajdošovi, Ph.D. za cenné rady, věcné připomínky a vstřícnost při konzultacích a vypracování bakalářské práce.

Abstrakt

Cílem práce je navrhnout a implementovat komponentu a webovou aplikaci, umožňující sběr, analýzu a vizualizaci dat vybraného trhu. Práce se zabývá architekturou systému, použitými technologiemi, popisem sbíraných dat, použitým postupem pro analýzu dat a vzhledem k implementaci také obecnou architekturou rozšíření pro webový prohlížeč.

Klíčová slova: Rozšíření webového prohlížeče, chování trhu, vizualizace dat, webová aplikace

Abstract

The aim of the thesis was to design and implement a component and a web application, enabling collection, analysis and visualisation of selected market data. The thesis deals with system architecture, used technologies, description of data collection, method used for data analysis, and due to implementation it deals with the general architecture of web browser extension

Keywords: Web extension, market behavior, data visualisation, web application

Obsah

Seznam použitých zkratek a symbolů

HTML	–	Hyper Text Markup Language
CSS	–	Cascading Style Sheets
MVT	–	Model-View-Template
API	–	Application programming interface
SQL	–	Structured query language
URL	–	Uniform resource locator
MIME	–	Multipurpose Internet Mail Extensions
TCP	–	Transmission Control Protocol
XML	–	Extensible Markup Language
FPS	–	Frames per second

Seznam obrázků

1 Úvod

Analýza dat, zpozorovaných v různých časových úsecích, představila problém ve statistickém modelování. Mnoho tradičních statistických metod závisí na předpokladu, že pozorovaná data jsou souborem stejně rozdělených nezávislých náhodných veličin. Takový předpoklad ale v těchto datech neplatí. Sousedící veličiny zde mají mezi sebou jasnou vazbu, a tak znemožňují použití některých statistických metod. Tento druh dat se nazývá časová řada a přístup zodpovídající jejich matematické a statistické otázky pak analýza časových řad. Problém detailně popisují R. H. Shumway a D. S. Stoffer v jejich knize [34].

Příklady užití této časové řady mohou být nalezeny v mnoha odvětvích. Ve zdravotnictví se vyskytují například ve formě analýzy biologického signálu, kde se s využitím strojového učení analyzuje signál z mozku (viz [6]). Další užití může být například analýza vývoje chřipky za určité období. Mnoho příkladů užití lze nalézt například v knize od autorů S. Bisgaard a M. Kulahci [2]. Tato práce se bude zabývat dalším užitím této časové řady, a to analýzou chování trhu.

Trh je místo, kde lidé prodávají, nakupují, či směňují zboží a služby. Nabízející a poptávající jsou mezi sebou v kontaktu, a to ve formě přímé, nebo skrze prostředního člena. Data trhu lze znázornit pomocí časové řady. Vždy se určitá akce vykoná v nějakém časovém bodě. Z dat vybraného trhu lze tedy jednoduše sestavit časovou řadu a pomocí strojového učení provést například předpověď.

Cílem této práce je tedy vyvinout nástroj, který nám umožní získat data vybraného trhu ve formě časové řady a tuto časovou řadu analyzovat a vizualizovat. Tento nástroj musí umožňovat zpracování libovolných, časově uspořádaných dat. Data trhu se mění velice rychle, je tudíž nezbytné, aby nástroj umožňoval získávání dat v reálném čase. Příkladem potřebných dat jsou volně dostupná data z webových stránek burz, které tyto data zprostředkovávají pro lidi obchodující na burze. Zdrojem dat pro tento nástroj je tedy webová stránka dané burzy. Získaná data se ukládají a jsou analyzována a vizualizována aplikací, která je součástí této práce.

Tato práce je rozdělena na čtyři hlavní kapitoly a závěr. Nejprve se práce zabývá různými technologiemi pro vývoj webových komponent a teoretickými východisky. V další kapitole je rozebrán návrh celého systému, tedy návrh komponenty pro získávání dat a aplikací pro analýzu a vizualizaci dat. Dále je v práci ukázána implementace jednotlivých částí, které byly probrány při návrhu. V předposlední částí práce jsou určité experimenty, které ukazují jak se systém chová nad skutečnými daty. Také je zde systém podroben výkonnostnímu testování. V závěru jsou pak uvedeny dosažené výsledky a možná budoucí rozšíření.

2 Teoretická východiska a použité technologie

V této kapitole budou obsažena teoretická východiska práce v závislosti na jejím cíli, tedy přístup k řešení předpovědi časové řady a budou zde rozebrány a popsány různé technologie určené pro vývoj webových komponent.

2.1 Technologie pro vývoj webových komponent

V této kapitole budou popsány různé technologie pro vývoj a chod webových komponent. Některé z těchto technologií budou mezi sebou porovnány.

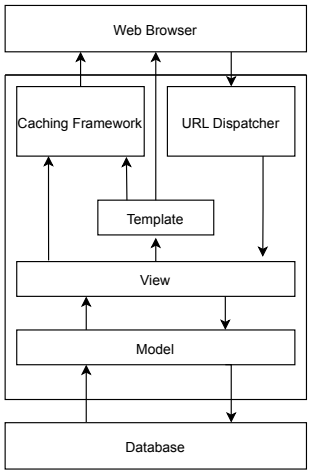
Webové komponenty komunikují s uživatelem skrz webový prohlížeč. Každý webový prohlížeč podporuje HTML, CSS a JavaScript, což jsou tři základní technologie, které jsou využity v téměř každé komponentě.

HTML je značkovací jazyk pro vyjádření struktury dokumentu, který má být prezentován v rámci služby WWW. Dokument se skládá z elementů, které jsou vyznačeny tagy. Element dává vymezenému textu určitý význam (např. nadpis nejvyšší úrovně). HTML dále umožňuje vytvářet hypertextové vztahy buď v rámci téhož dokumentu, nebo mezi různými dokumenty (nezávisle na jejich umístění s využitím URL). [18]

CSS je soubor pravidel pro grafickou úpravu webových dokumentů. Pomocí kaskádových stylů lze definovat formát vlastnosti (písmo, zarovnání, barvy atd.) samostatně pro každý logický prvek struktury dokumentu (nadpisy, běžný text, odkazy atd.), definice je uplatněna jednotně v celém dokumentu. Jde o účinný prostředek pro zajištění jednotného vzhledu celého webu založený na principu oddělení popisu struktury (HTML) od její vizualizace (CSS). [5]

JavaScript je multiplatformní, skriptovací, objektově orientovaný jazyk, který byl představen v roce 1995. Je nezbytnou součástí každé moderní webové aplikace. [17]

Django je open source framework pro tvorbu webových aplikací. Je napsán v Pythonu a nezávisle se drží architektury MVC (Model View Controller). Hlavní myšlenkou tohoto frameworku je znovupoužitelnost jednotlivých komponent a jejich vzájemné propojování. Pro zajištění konverze dat mezi relační databází a třídami využívá Django objektově relační mapování. Popis datového modelu aplikace je výhradně v souboru models.py. Zpracování webových požadavků a následné vracení odpovědi obstarává Python funkce, která se nazývá view. Tyto funkce jsou uloženy v souboru aplikace views.py. Pro zobrazování stránek a dynamického generování jejich obsahu se používají šablony - Teamplates. Tyto šablony jsou psány v HTML a pro práci s nimi se používá frameworkem definované API.



Obrázek 1: Architektura Django frameworku [9]

Více informací o Django frameworku lze nalézt například v oficiální dokumentaci [10], nebo v knize od W. S. Vincenta [36].

RESTful API může být vytvořeno pouze za pomocí frameworku Django, nicméně Django REST framework přináší určité výhody a celkové ulehčení práce. **Django REST framework** nám dovoluje velice rychle vyvíjet RESTful API za pomocí Django modelů. Disponuje „Web browsable API“, které umožňuje testovat API přímo v prohlížeči bez využití jiných aplikací třetích stran. Framework je velice rozsáhlý a v této práci je použita pouze malá část jeho funkcí. Více informací o Django REST frameworku lze nalézt na webových stránkách [11].

Alternativou založenou na jazyce Python je **Flask**. Flask není tak rozsáhlý jako Django. Jedná se o velice malý framework, který je jednoduchý a rychlý. Tak, jako v Django a Django REST frameworku v něm lze vytvářet RESTful API.

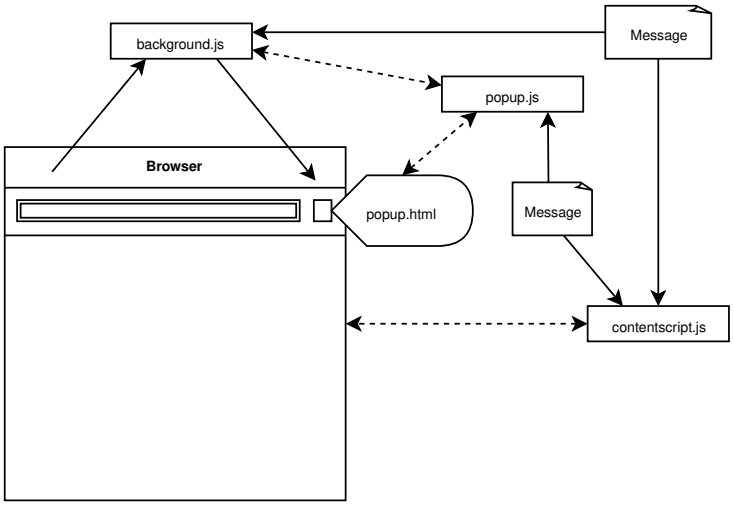
React je knihovna skriptovacího jazyka JavaScript, vyvíjena Facebookem, určená pro tvorbu libovolných uživatelských rozhraní. Hlavním pilířem této knihovny je skládání rozhraní z oddělených kusů, neboli komponent. Celkově je tak kód lépe čitelný, snadno udržovatelný a rozšiřitelný. Jednotlivé komponenty jsou psány použitím JSX, což je rozšíření syntaxe jazyka JavaScript. Pro detailnější pohled na React lze využít oficiální dokumentace [30], nebo knižních zdrojů [37].

Alternativou Reactu je například populární **Angular**. Oproti Reactu, který je knihovnou jazyka Javascript, Angular je framework. Obě dvě technologie mohou být využity pro mobilní i webové aplikace. React je ale oproti Angularu u složitějších aplikací rychlejší. Nelze jednoznačně určit, která technologie je vhodnější pro použití. Při rozhodování se jedná spíše o osobní preferenci a dosavadní zkušenost. [1]

Rozšíření prohlížeče jsou malé komponenty, které dovolují upravovat a rozšiřovat defaultní funkci prohlížeče. Jsou psána pomocí technologií HTML, CSS a JavaScriptu. Soubory rozšíření jsou zabaleny do balíku s příponou crx, který si uživatel stáhne a do prohlížeče nainstaluje. Rozšíření se stahují z oficiálních internetových obchodů (např. Chrome - [21], Firefox - [13]) a spravují přímo v prohlížeči.

Rozšíření se skládá z několika klíčových souborů:

1. Manifest
    - Tento JSON soubor obsahuje základní metadata. Je to jediný soubor, který je nezbytný v jakémkoliv rozšíření. Metadata, která tento soubor obsahuje, jsou například: jméno, oprávnění, odkazy na skripty a verze.
2. Background script
    - Background script se načítá jakmile se načte webový prohlížeč a slouží jako event handler pro eventy prohlížeče jako celku.
3. UI prvky
    - Rozšíření může obsahovat různé komponenty uživatelského rozhraní. Tyto komponenty jsou psány jako klasické HTML webové stránky.
4. Content script
    - Content script čte a modifikuje DOM z jedné konkrétní stránky prohlížeče. S ostatními částmi rozšíření (Background script, UI prvky) může komunikovat pomocí zasílání zpráv.



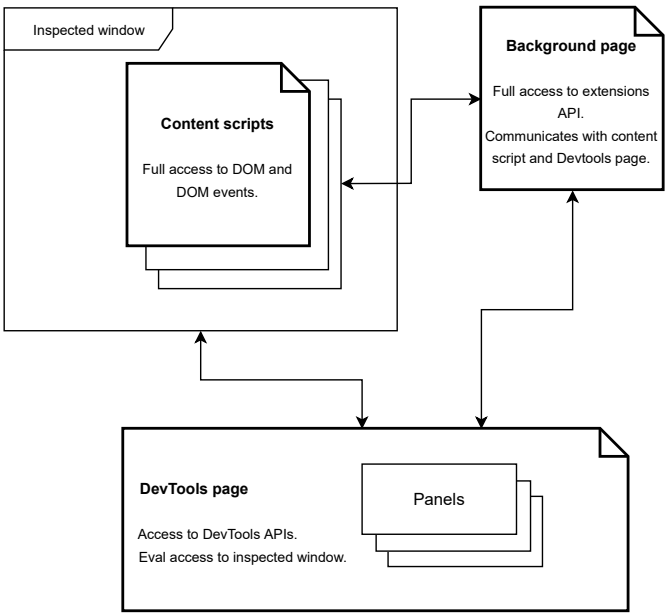
Obrázek 2: Architektura rozšíření se znázorněnou komunikací mezi jednotlivými skripty [20]

Rozšíření vyvinuto s využitím WebExtension API je kompatibilní s prohlížeči Firefox, Chrome, Edge, a Opera Přestože jsou rozšíření mezi prohlížeči kompatibilní, mají mezi sebou určité nesrovnalosti (viz [8]):

- Podporované funkce Javascript API se u různých prohlížečů liší.

- Manifest.json má v různých prohlížečích různé klíče.
- K Javascript API je přistupováno pod různými jmennými prostory.

Krom klasického rozšíření prohlížeče lze vyvíjet také rozšíření pro vývojářské nástroje (DevTools). Rozšíření DevTools je vyvíjeno prakticky stejně, jako klasické rozšíření. Navíc je zde možnost využít speciální DevTools API. Jako UI prvek je zde možnost vytvořit svůj vlastní DevTools panel, nebo panel postranního panelu - sidebar pane. Jak už název napovídá, sidebar pane se váže k specifickému panelu. Pro více informací ohledně vývoje univerzálního rozšíření do prohlížeče je k dispozici dokumentace na stránkách MDN Web Docs [22], nebo pak dokumentace k vývoji rozšíření pro prohlížeč Chrome [16].



Obrázek 3: Architektura rozšíření DevTools [19]

Webové API je jakýsi vzor HTTP požadavků a odpovědí, používaných pro přístup k webové stránce. Tato webová stránka je určená spíše libovolným počítačovým programům, než webovým prohlížečům používaným lidmi. [4]

HTTP protokol, který webové API využívá, je internetový protokol určený původně pro výměnu hypertextových dokumentů ve formátu HTML.

V současné době je používán i pro přenos dalších informací. Pomocí rozšíření MIME umí HTTP přenášet jakýkoli soubor (podobně jako e-mail). MIME se používá společně s formátem XML pro tzv. webové služby (spouštění vzdálených aplikací).

Protokol funguje způsobem dotaz-odpověď. Uživatel (pomocí programu, obvykle internetového prohlížeče) pošle serveru dotaz ve formě čistého textu (obvykle na port TCP/80), obsahujícího označení požadovaného dokumentu, informace o schopnostech prohlížeče apod. Server poté odpoví pomocí několika řádků textu popisujících výsledek dotazu (zda se dokument poda-

řilo najít, jakého typu dokument je atd.), za kterými následují data samotného požadovaného dokumentu. [25]

Webové API, které je navrženo dle architektury **REST** (Representational State Transfer) se nazývá RESTful API. Tato architektura byla představena v disertační práci Roye Fieldinga [12]. V dokumentaci Microsoftu ohledně návrhu webového rozhraní API [24] jsou některé hlavní zásady RESTful API definovány takto:

- Rozhraní REST API jsou navržená kolem prostředků, což je jakýkoli druh objektu, dat nebo služby, ke kterým může klient přistupovat.
- Prostředek má identifikátor, což je identifikátor URI, který jednoznačně tento prostředek identifikuje.
- Klienti komunikují se službou tak, že si vyměňují reprezentace prostředků. Mnoho webových rozhraní API používá jako formát výměny JSON.
- Rozhraní REST API používají jednotné rozhraní, které pomáhá oddělit implementace klienta a služby. Pro rozhraní REST API postavená na protokolu HTTP zahrnuje jednotné rozhraní použití standardních sloves HTTP k provádění operací s prostředky. Nejběžnější operace jsou GET, POST, PUT, PATCH a DELETE.
- Rozhraní REST API používají bezstavový model požadavků. Požadavky HTTP by měly být nezávislé a může k nim dojít v libovolném pořadí, takže udržování přechodných informací o stavu mezi požadavky není vhodné. Jediným místem, kde se informace ukládají, jsou samotné prostředky a každý požadavek by měl být atomickou operací. Toto omezení umožňuje webovým službám být vysoce škálovatelné, protože není nutné zachovávat žádné vztahy mezi klienty a konkrétními servery. Každý server může zpracovat jakýkoli požadavek od jakéhokoli klienta.
- Rozhraní REST API se řídí hypermediálními odkazy, které jsou obsaženy v reprezentaci.

2.2 Model pro regresi

Pro odhad hodnoty určité veličiny se využívá regresní analýza. Matematických modelů pro regresi je mnoho. Je možné využít například lineární regrese, logistické regrese, nebo polynomické regrese. Lze také užít složitějších modelů na implementaci, například náhodný les, nebo neuronové sítě. Níže budou více osvětleny dva typy modelů.

2.2.1 Lineární regrese

Lineární regrese je lineární přístup k znázornění vztahu mezi závislými a nezávislými proměnnými. Znázornění vztahu mezi jednou závislou a jednou nezávislou proměnnou se nazývá jednoduchá lineární regrese a právě tato je v této práci použita. Model lineární regrese je založen na

proložení několika bodů přímkou. Přímka může být jednoduše popsána rovnicí:

$$y = mx + b$$

Kde:

- x: souřadnice osy X
- y: souřadnice osy Y
- m: směrnice přímky
- b: koeficient y

Optimální m a b můžeme nalézt minimalizací chyby, tedy vzdáleností mezi skutečnou hodnotou a odhadem (hodnotou na přímce lineární regrese). Tato chyba se nazývá střední kvadratická chyba (**MSE** = mean squared error) a rovnice pro změření této chyby je definována takto:

$$MSE = \frac{1}{2}\sum_{i=1}^{n}(y_i - \widetilde{y_i})$$

Kde:

- y_i: skutečná hodnota
- $\widetilde{y_i}$: odhadovaná hodnota

Minimum této rovnice je tam, kde je gradient nulový. Vyřešením dvou jednoduchých parciálních derivací lze nalézt rovnice pro optimální m a b:

$$\frac{\partial MSE}{\partial m} = 0 \qquad\qquad \frac{\partial MSE}{\partial b} = 0$$

Po zderivování tedy:

$$\frac{\partial MSE}{\partial m} => -\frac{\sum x_i y_i}{n} + m\frac{\sum x_i^2}{n} + b\frac{\sum x_i}{n} = 0$$
$$\frac{\partial MSE}{\partial b} => -\frac{\sum y_i}{n} + m\frac{\sum x_i}{n} + b = 0$$

Z výsledných rovnic pak můžeme vyjádřit samostatně m a b:

$$m = \frac{n\sum x_i y_i - \sum y_i \sum x_i}{n\sum x_i^2 - (\sum x)^2}$$

$$b = \frac{\sum y_i - n\sum x_i}{n}$$

Za pomocí těchto výsledných rovnic můžeme jednoduše proložit libovolný počet bodů přímkou. Více informací ohledně lineární regrese, včetně složitějších řešení než jednoduché lineární regrese, lze nalézt například v této knize [33].

2.2.2 Náhodný les

Tato metoda strojového učení vychází z rozhodovacích stromů. Jde o kombinaci určitého počtu těchto rozhodovacích stromů tak, že celkovým výsledkem je výsledek určité matematické operace nad výstupy jednotlivých stromů.

Náhodný les je tvořen množinou vzájemně nepropojených rozhodovacích stromů, jejichž výstupy jsou spojeny v jeden. V každém vnitřním uzlu stromu počínaje kořenem je položena určitá otázka, např. při problému klasifikace obrázků se může jednat o otázku: „Je rgb hodnota pixelu na pozici `x, y` větší, než hodnota pixelu na pozici `x+1, y`?“. Postupně se dostaneme až k listům stromu, které představují finální rozhodnutí dané větve. K univerzálnímu užití tohoto stromu pro klasifikaci, či regresi se musí sestavení rozhodovacího stromu zautomatizovat na základě vstupních dat tak, aby byla minimalizována chyba na každém listu rozhodovacího stromu. Algoritmů pro sestavení rozhodovacího stromu je mnoho. Například ID3, C4.5, MARS, a další. Všechny tyto algoritmy ale sdílí stejnou myšlenku rekurzivního nalezení nejvhodnějšího bodu pro větvení, nicméně postup nalezení tohoto bodu se v algoritmech liší.

Například stromy typu CART jsou vhodné pro kategoriální i regresní úlohy a rostou na základě rekurzivního binárního dělení [3]. Na začátku tvorby stromu patří všechna pozorování souboru do jednoho uzlu neboli kořene. Následně jsou tato pozorování rozdělena do dvou dceřiných uzlů, na základě hodnoty prediktoru, které jsou dále děleny opět binárně na další uzly. Při snaze nalézt správné rozdělení se snažíme o takové rozdělení závislé proměnné Y prediktorem X, aby hodnoty proměnné Y byly uvnitř uzlu co nejhomogennější a zároveň mezi uzly co nejrozdílnější. Který prediktor (a jeho hodnota) nám zajistí nejlepší rozdělení, zjistíme pomocí tzv. kriteriální statistiky (spliting criterium), která určuje homogenitu uzlu. Jako kriteriální statistiku lze užít například minimum kvadratické chyby, Gini index, entropii, nebo klasifikační chybu. [32]

Velkou výhodou těchto rozhodovacích stromů je vysoká interpretovatelnost. Rozhodovací strom je totiž určitý „if-then-else“ diagram, který kombinuje jednotlivé atributy dat do jednoho rozhodnutí. Je tedy velice jednoduché, interpretovat chování tohoto stromu a odvodit si, proč je strom sestaven právě takto. Tato výhoda se může na první pohled zdát banální, avšak pro budoucí vývoj v oblasti umělé inteligence a strojového učení je tzv. vysvětlitelné AI (Explainable AI) velice žádoucí.

Aby náhodný les fungoval správně, je třeba, aby se skládal z více stromů, které musí být odlišné. Jak lze této vlastnosti dosáhnout? Odlišnost a zároveň zachování přiměřené úspěšnosti každého ze stromů docílíme tak, že při sestavování stromu bereme v potaz pouze náhodně utvořenou podmnožinu z celkové množiny dat. Jelikož se náhodný les skládá z více odlišných stromů, je kladně redukována možnost přeučení. Naopak, kdyby náhodný les sestával pouze z jednoho stromu, přeučení by bylo nevyhnutelné. S větším počtem rozhodovacích stromů roste i čas trénování náhodného lesu. Tato práce se nebude zabývat teorií této metody strojového učení do hloubky, a tak pro získání více informací na toto téma lze využít knihu zabývající se touto problematikou [35].

3 Návrh

V této kapitole bude rozebrán návrh systému a budou jmenovány technologie, které budou v této práci využity.

Hlavní částí bude v této práci komponenta pro získávání dat vybraného trhu. Tato komponenta bude realizována formou rozšíření prohlížeče s využitím WebExtension API a DevTools API. Rozšíření se tedy bude snažit být univerzální pro všechny prohlížeče. V rozšíření bude konfigurace, pomocí které si uživatel může změnit URL adresu webového API a směrovat tak data na vlastní úložný prostor. Vlastní webové API a webová aplikace zde budou tedy sloužit jako určité doplnění celku práce.

3.1 Komponenta pro získávání dat časové řady

Komponenta se bude snažit být co nejvíce uživatelsky přívětivá a intuitivní. V komponentě bude zapotřebí zadat určitou konfiguraci, tedy jaká data budou sbírána, kam posílána a jak budou pojmenována. Tato konfigurace je vcelku složitá a uživatelská možnost volby elementu potřebných dat je obtížná. Poměrně jednoduchá se zdá možnost využít JSON konfiguraci, kde si zkušenější uživatelé napíší konfiguraci vlastní a uživatelé bez nutných znalostí využijí již konfigurace hotové. S cílem dosažení uživatelské komfortnosti bude komponenta vyvinuta jako rozšíření vývojářských možností prohlížeče. Komfortnější je v tomto případě rozšíření vývojářských možností proto, že se konfigurace píše ručně v zmiňovaném JSON formátu, a tak by při klasickém rozšíření musel uživatel zbytečně měnit okna vývojářských možností a prohlížeče. V práci bude vytvořen nový panel postranního panelu, který bude rozšiřovat funkci již zavedeného panelu „Elements“. Rozšíření bude využívat knihovny React pro tvorbu uživatelského rozhraní a CSS frameworku Bootstrap pro jednodušší stylizaci, který nabízí již zhotovenou stylizaci pro jednotlivé HTML třídy.

Odesílaná data také musí obsahovat identifikátor, který jednoznačně identifikuje danou instanci sbírání dat. Tento identifikátor bude v systému formou unikátního řetězce, který komponenta obdrží od webového API při spuštění. Jednoznačnost identifikátoru bude zajištěna rekurzivním voláním metody pro jeho generování v případě vytvoření duplikátu.

3.2 Back-end systému

Jak název vypovídá, back-endem systému je myšlena ta část, která není přístupná uživateli ze strany prohlížeče. Jedná se tedy o webové API, databázi a R skript pro analýzu dat.

Jako rozhraní pro přístup k datům a jejich ukládání do databáze bude sloužit webové API. Využívá jej jak komponenta pro ukládání dat, tak webová aplikace pro přístup k nim. Pro toto API bude zvolen Django REST framework, a to z důvodu jednoduchosti tvorby RESTful API. Aby komponenta pro získávání dat mohla do databáze posílat jakékoliv data s proměnným počtem sloupců, odlišnými datovými typy a názvy sloupců, museli bychom pro každou stránku, na které se data sbírají, vytvořit novou databázovou tabulku upravenou na sbíraná data. Tomuto

problému lze předejít využitím databázového systému PostgreSQL. PostgreSQL totiž podporuje datový typ `jsonb`, který nám dovoluje ukládat JSON objekt reprezentující celý řádek sbíraných dat. Na webovém serveru, který bude hostovat Webové API, bude zároveň skript pro analýzu a následnou vizualizaci dat. Tento skript bude napsán v jazyce R vzhledem k jeho účelu pro statistiku. V R skriptu se vzhledem k relativně jednoduché implementaci a dobrým výsledkům používá pro předpověď následující hodnoty dané proměnné náhodný regresní les, anglicky random forest regression. Lineární regrese bude využita pro vykreslení trendové čáry. Lineární regrese je snadná na implementaci a pro trendovou čáru dostačující. Výstupem tohoto skriptu bude pdf soubor.

V případě, že uživatel bude chtít data zasílat na vlastní server, tak tento celkový back-end systému nebude použit.

3.3 Webová aplikaci pro analýzu a vizualizaci

Samotná webová aplikace pak bude umožňovat uživateli zobrazit si určitý přehled dat, stáhnout si data ve formě csv souboru, vizualizovat si časovou řadu v podobě grafu a analyzovat data časové řady. Již zmíněný vygenerovaný kód pro danou instanci získávání dat se specifikuje v URL adrese. Adresu nebude muset zadávat uživatel ručně, ale v rozšíření bude na stránku odkazovat již zhotovený odkaz. Webová aplikace pak stáhne záznamy s daným kódem a umožní práci nad správnými daty. Pro analýzu se odešle na server požadavek. Server vrátí uživateli soubor pdf s predikcí další hodnoty a vykreslenou trendovou čáru dat časové řady. Tato webová aplikace bude taktéž využívat knihovnu Javascript knihovnu React pro tvorbu uživatelského rozhraní a CSS framework Bootstrap.

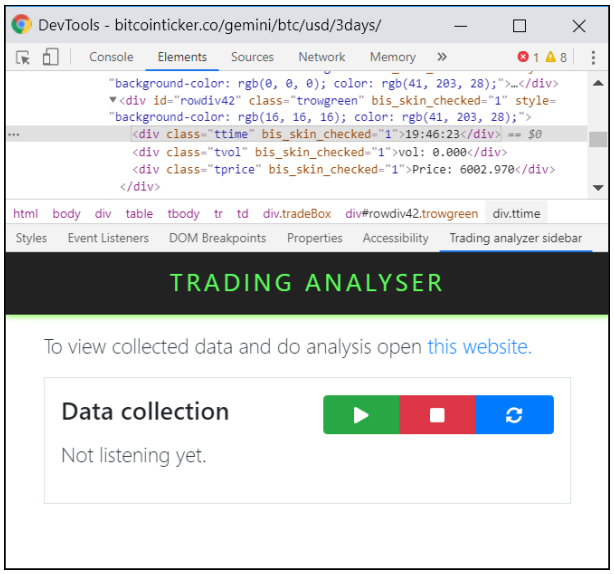
To, co bylo řečeno v předchozí podkapitole, platí i zde. Pokud tedy bude specifikován uživatelem vlastní server pro zasílání dat, nebude využita ani tato část pro analýzu a vizualizaci dat časové řady.

4 Implementace

V této kapitole se budeme snažit implementovat jednotlivé části systému podle zmíněného návrhu.

4.1 Implementace komponenty pro získávání dat časové řady

Komponenta je tedy realizována jako rozšíření prohlížeče. Jelikož rozšíření není veřejně publikováno a nelze tedy stáhnout z oficiálního internetového obchodu, musí se manuálně přidat soubory rozšíření do prohlížeče a rozšíření aktivovat. Postup závisí na použitém prohlížeči. K funkcím rozšíření pak lze přistupovat ve vývojářských nástrojích pod záložkou „Elements“. Jak rozšíření v prohlížeči vypadá je vyobrazeno níže.



Obrázek 4: Rozšíření v prohlížeči Chrome - úvodní obrazovka

Po kliknutí na zelené tlačítko - viz Obrázek 4, se zobrazí vyskakovací okno s možností přidání JSON konfigurace. Tato konfigurace obsahuje vše potřebné pro správný chod rozšíření a získávání požadovaných dat. Vzhled vyskakovacího okna je vyobrazen níže.

Obrázek 5: Rozšíření v prohlížeči Chrome - vyskakovací okno

JSON objekt, který se využívá jako konfigurace, má následující strukturu:

- `post_url` : URL webového API pro zpracování dat odesílaných rozšířením pomocí metody POST.
- `user_code_url` : URL webového API pro vygenerování unikátního klíče, který jednoznačně identifikuje dané získávání dat. Používá se v případě, že ukládáme do jedné databázové tabulky záznamy z několika různých zdrojů.
- `save_interval` : Jedná se o minimální interval uvedený v ms, ve kterém rozšíření posílá data na URL specifikovanou v `post_url`. Minimální proto, že data časové řady jsou mnohdy na webové stránce zveřejňována v nepravidelných intervalech.
- `main_selector` : Jednoznačný identifikátor elementu obsahujícího potřebná data. Rozšíření pak sleduje jakékoliv změny v rámci tohoto elementu.
- `record` : Objekt reprezentující jeden řádek v tabulce. Objekt obsahuje tyto atributy:
  - `name` : Název řádku.
  - `selector` : Jednoznačný identifikátor elementu reprezentujícího jeden řádek získávaných dat.
  - `fields` : Pole objektů reprezentující sloupce řádku. Každý objekt obsahuje tyto atributy:
    - `name` : Název sloupce.

    - `selector` : Jednoznačný identifikátor elementu reprezentujícího požadovanou hodnotu.
    - `regexp` : Regulární výraz v případě určité manipulace s hodnotou tohoto sloupce.

Po potvrzení zadané konfigurace se vytvoří pozorovatel (observer) MutationObserver. Návrhový vzor pozorovatele umožňuje odběratele zaregistrovat a přijímat oznámení od zprostředkovatele. Vždy, když dojde k splnění definované podmínky, události nebo změny stavu, zprostředkovatel automaticky upozorní všechny pozorovatele voláním jedné z jejich metod. [26] Tento observer sleduje změny, které jsou provedené na stromu DOM. Rozšíření používá metodu observe(), která konfiguruje MutationObserver tak, aby přijímal notifikace při změně DOM. MutationObserver poté spustí metodu, která data podle vložené konfigurace připraví pro odeslání metodou POST. Rozšíření odesílá nová data na server s minimálním intervalem, specifikovaným v konfiguraci, kde jsou data dále zpracovávána. Více ohledně tohoto observeru lze nalézt v oficiální dokumentaci [23].

4.2 Implementace back-endu systému

V předchozí podkapitole 4.1 byla vysvětlena implementace komponenty pro získávání dat časové řady. Tato podkapitola na ni naváže.

Pro data, která komponenta zasílá, je připraveno webové API. API sestává z několika metod, pomocí kterých můžeme pracovat s daty. Metody jsou níže popsány.

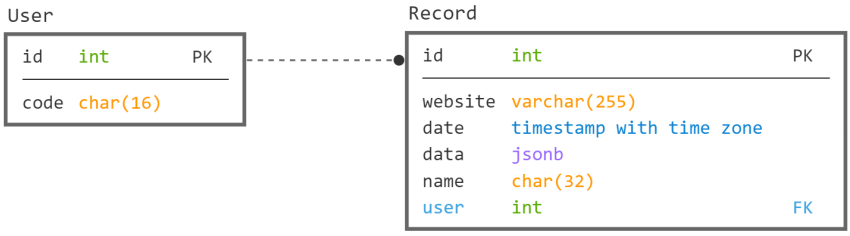
- `records` - **metoda**: GET, **url**: `/records/`, **odpověď serveru**: seznam všech záznamů v tabulce Records
- `user_records` - **metoda**: GET, **url**: `/records/user_records/?code=X`; kde X je unikátní kód uživatele generovaný metodou `register-user`, **odpověď serveru**: seznam všech záznamů v tabulce Records, kde kód uživatele je X
- `insert_records` - **metoda**: POST, **url**: `/records/insert/insert-records/`, **tělo, které metoda přijímá**:

```
{
   data: "pole nových elementů",
   user: "unikátní identifikátor uživatele",
   date: "aktuální datum",
   website: "stránka na které je rozšíření právě aktivní"
 }
```

- `analyse_records` - **metoda**: GET, **url**: `/records/analyse_records/?username=X&colnum=Y`; kde X je unikátní kód uživatele generovaný metodou `register-user` a Y je index sloupce, který chceme analyzovat, **odpověď serveru**: vytvoří se pdf soubor s vyhotovenou analýzou, přístupný pod `media/Z`; kde Z je kód uživatele

- `users` - **metoda**: GET, **url**: `/users/`, **odpověď serveru**: seznam všech uživatelských kódů
- `register-user` - **metoda**: GET, **url**: `/users/register-user/`, **odpověď serveru**: vygenerovaný unikátní kód

Databáze obsahuje 2 tabulky. Tabulka `User` obsahuje pouze vygenerované unikátní kódy metodou `register-user`. Tabulka `Record` slouží k ukládání dat, zaslaných metodou `insert_records`.



Obrázek 6: Schéma databáze

Pro ukládání získaných dat se tedy v databázi používá sloupec `data`, který je datového typu `jsonb`. Do tohoto sloupce se uloží JSON objekt obsahující pole všech sloupců a jejich hodnota na řádku. Práce s datovým typem `jsonb` je uvedena v dokumentaci PostgreSQL [28]

Na serveru se také nachází R skript, který je volán metodou `analyse_records`. Tento skript se připojí k databázi, získá požadované záznamy tabulky a nad těmito daty provede určité akce: předpověď další hodnoty časové řady, vykreslení trendové čáry, porovnání předpovědi se skutečnými hodnotami. Skript pro tuto celou analýzu vygeneruje, za pomocí balíčku pro LaTeX, pdf soubor, který na serveru uloží.

4.2.1 Trendová čára

Přímka lineární regrese slouží v aplikaci pro znázornění trendové čáry mezi uživatelem zvolenou proměnnou a časem. Diferenciální výpočet zmíněný v kapitole 2.1.1 se v jazyce R nemusí dělat ručně, ale je zde například funkce `lm()`, která umí vypočítat směrnici přímky a koeficient. Přímka se pak vykreslí pomocí funkce `abline()` do již existujícího grafu.

4.2.2 Předpověď následující hodnoty

Pro předpověď následující hodnoty je zvolen náhodný regresní les. Jazyk R nabízí mnoho knihoven strojového učení pro práci s regresními i klasifikačními lesy. V této práci je využita knihovna `randomForest` (viz [29]), která dovoluje využití stejnojmenné funkce.

Princip, jaký se používá v aplikaci pro odhadnutí následující hodnoty zvolené proměnné, je založen na metodě posuvného okna. Pokud tedy máme data časové řady ve formátu vektoru, tedy $(x_1, x_2, ..., x_n)$ a požadovaná hodnota je x_{n+1}, pak si tyto data musíme připravit tak, abychom z

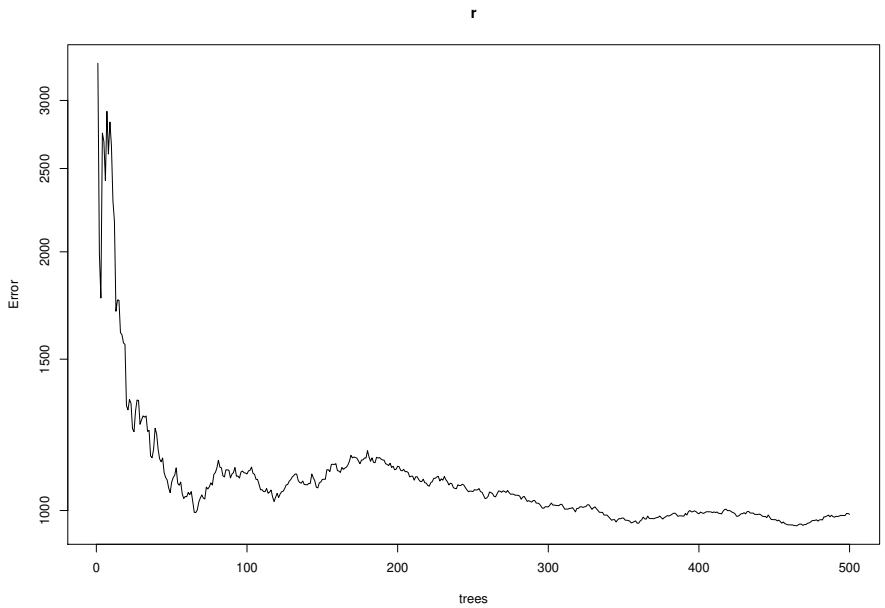
dat získali matici, kde každý řádek představuje vstupní hodnoty a vektor, který pro každý řádek matice má právě jednu hodnotu reprezentující výsledný výstup pro tyto vstupní data. Postup pro úpravu dat vypadá tedy takto:

$$\boldsymbol{d} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \Longrightarrow \quad \boldsymbol{X} = \begin{bmatrix} x_1 & x_2 & \cdots & x_{k-1} \\ x_{1+p} & x_{2+p} & \cdots & x_{k+p-1} \\ x_{1+2p} & x_{2+2p} & \cdots & x_{k+2p-1} \\ & & \vdots & \\ x_{n-k} & x_{n-k+1} & \cdots & x_{n-1} \end{bmatrix} \quad , \boldsymbol{y} = \begin{bmatrix} x_k \\ x_{k+p} \\ x_{k+2p} \\ \vdots \\ x_n \end{bmatrix} \tag{1}$$

Kde n je celkový počet dat vektoru d, p je posun posuvného okna a k je velikost posuvného okna. Výsledkem této úpravy je tedy matice X, která obsahuje vstupní data a y obsahující data výstupní. Využitím již zmiňované funkce `randomForest` můžeme nyní vytrénovat náhodný les. Ještě před úpravou si ale z dat odebereme část pro pozdější testování, konkrétně 80% pro trénování a 20% pro testování. Trénování náhodného lesu v R vypadá tedy takto:

```
model <- randomForest(x=X_train, y=y_train, keep.forest=T, ntree=500)
```

Po vytrénování náhodného lesu se můžeme podívat na to, jak se vyvíjí chyba v tomto lese společně s narůstajícím počtem rozhodovacích stromů.

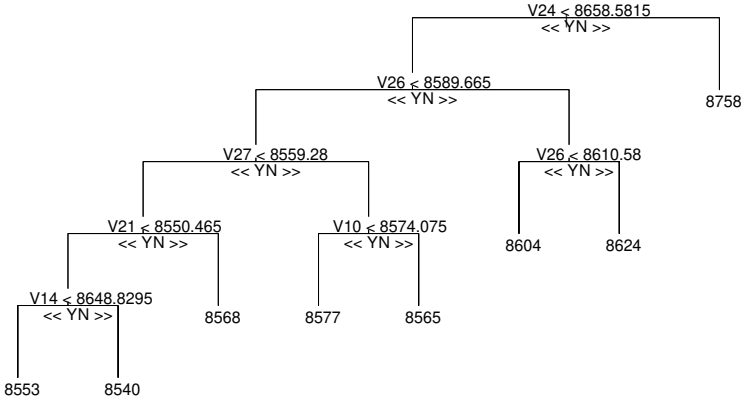


Obrázek 7: Chyba versus počet rozhodovacích stromů (OOB error)

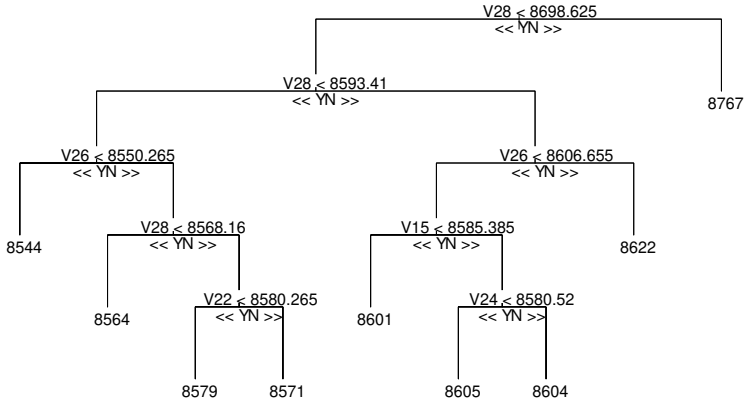
Z grafu lze vypozorovat, že náhodný les dosahuje nejlepších výsledků pro tyto data při přibližně 80 rozhodovacích stromech, a poté až při 350 a více stromech. S jinými zvolenými daty bude graf vypadat jinak, a tak nelze bezpečně určit počet stromů, při kterém bude mít náhodný

les nejlepší výsledky. V aplikaci je použit počet stromů 500, což je zároveň výchozí hodnota vstupního parametru funkce.

Pro lepší představu, jak tento náhodný les v aplikaci funguje, si můžeme vykreslit několik rozhodovacích stromů, které byly vytrénovány.



Obrázek 8: Rozhodovací strom 1



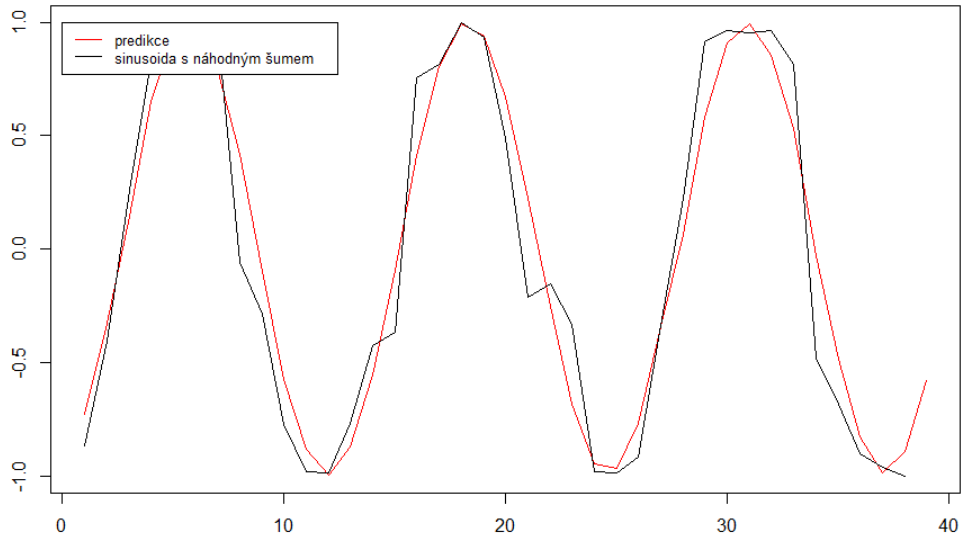
Obrázek 9: Rozhodovací strom 2

Z grafů lze vypozorovat, že při vytváření těchto rozhodovacích stromů byla skutečně využita jiná podmnožina dat, neboť například atribut dat V10 se v prvním grafu nachází, ale v 2. není.

Také lze vidět, že tyto sestavené stromy se opravdu skládají z jednoduchých podmínek typu porovnávání dvou čísel.

Pro požadovaný odhad následující hodnoty, tedy x_{n+1}, stačí využít R funkce `predict`, která jako vstupní parametr přijímá v tomto případě vektor $(x_{n-k+1}, x_{n-k+2}, \cdots, x_n)$, tedy posledních $k-1$ hodnot vektoru d.

20% dat, která byla z dat pro trénování odebrána, lze nyní využít pro otestování, zda je vytrénovaný náhodný les dostatečně přesný. Můžeme si například vykreslit graf, tak jako v aplikaci, který zobrazí předpověď pomocí vytrénovaného náhodného lesu společně se skutečnými hodnotami. Jako data je nyní zvolen sinusový signál s náhodným šumem.



Obrázek 10: Predikce a skutečné hodnoty sinusoidy s šumem

Sinusoida osciluje, i přes přidaný šum, rovnoměrně, a tak je předpověď další hodnoty přesná. Je třeba ale dodat, že model pro předpověď nevytvořil data pro tento graf jen za pomocí předchozího trénování, graf této predikce je tvořen tak, že jsou testovací data upravena stejným způsobem, jako data trénovací a model se pak snaží vyhodnotit vektor $\widetilde{y}$. Tento graf je tak pouze vykreslení vektorů y a $\widetilde{y}$. Graf znázorňující porovnání predikce a skutečné hodnoty je taktéž obsažen v analýze vygenerované uživatelem.

4.3 Implementace webové aplikace pro analýzu a vizualizaci dat časové řady

Pro zobrazování dat, a provádění operací nad nimi, se využívá webová aplikace. Každý uživatel, který začne data sbírat, obdrží od API unikátní klíč. Tento klíč se používá právě v této webové aplikaci, ve které klíč uvedeme v URL. Webová aplikace pak zobrazuje a pracuje s daty daného uživatele. Z tohoto přístupu k uživatelským datům vyplývá, že data jsou veřejně k dispozici bez nutnosti jakéhokoliv přihlášení. Data jsou nicméně chráněna právě šestnáctimístným klíčem

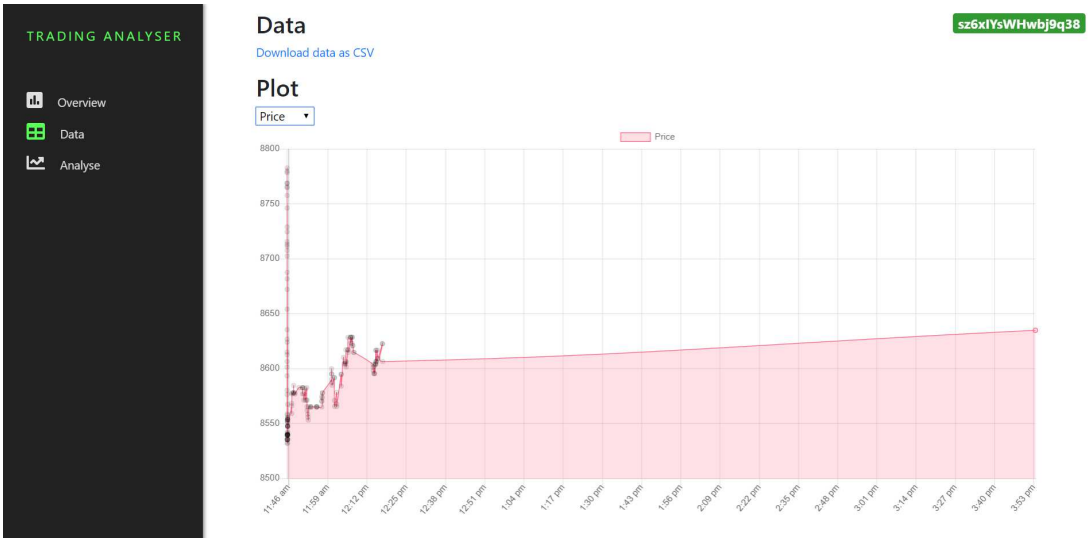
složeným z čísel a písmen abecedy s možností velkých a malých písmen uvedeným v URL. Toto zabezpečení zajistí to, že se uživatel nedostane nechtěně k datům jiných uživatelů.

Webová aplikace má 3 podstránky. První se nazývá „Overview“, a jak z názvu vypovídá, nachází se zde přehled sbíraných dat. Uživatel má možnost podívat se na počet získaných dat, průměrný počet získávání za minutu, webovou stránku ze které data získává a další položky týkající se obecných informací o datech.

Další podstránka má název „Data“. Zde si uživatel může zobrazit graf časové řady tvořenou libovolným sloupcem uchovávajícím číselné údaje dat. Pro tyto grafy je využita React knihovna `react-chartjs-2`, která dovoluje využívat již hotové komponenty představující daný graf. Webová aplikace využívá z této knihovny komponentu `Line`, která vykresluje graf hranový. Více informací ohledně této knihovny pro vykreslování grafů lze nalézt v dokumentaci [31]. Také je zde možnost stáhnout si data v podobě csv souboru.

Poslední podstránkou je podstránka „Analyse“. Zde si uživatel zvolí sloupec s číselnou hodnotou, pro který chce provést analýzu. Webová aplikace se pak odkáže na API, konkrétně na metodu `analyse_records`. Výsledný soubor vygenerovaný touto metodou je pak k dispozici pro zobrazení na stejné podstránce webové aplikace. Vygenerovaný soubor je v této práci k dispozici k nahlédnutí v sekci přílohy.

Níže je zobrazen vzhled této webové aplikace. V levé části aplikace se nachází navigace. Pomocí odkazů v této navigaci se pak vykresluje obsah v pravé části aplikace. V horním pravém rohu lze vidět unikátní kód přidělený uživateli. Taktéž si lze povšimnout, že kód je na zeleném pozadí. Zelené pozadí indikuje, že data byla stažena. Naopak oranžové pozadí indikuje, že data zatím nebyla stažena ze serveru.



Obrázek 11: Webová aplikace, záložka Data

5 Experimenty a výkonnostní testy

Celkový software, který je v této práci vytvořen, tedy webová komponenta pro získávání dat časové řady, back-end systému a webová aplikace pro analýzu a vizualizaci dat časové řady, by měl být pro získání informací ohledně kvality softwaru, a jeho následnou optimalizací, testován. V této kapitole budou probrány možnosti testování webových aplikací a budou vytvořeny experimenty se specifickým nastavením, ve kterých bude ukázáno, jak aplikace funguje v praxi. Tyto experimenty nám také poskytnou podklad k výkonnostnímu testování a z toho důvodu bude mít každý experiment jiné podmínky spuštění aplikace. Snahou je tedy nalézt optimální podmínky pro chod aplikace.

Testování může být provedeno buďto dle určitého scénáře manuálně, nebo se může zautomatizovat s cílem testovat opakující se scénáře. V této práci budeme testovat aplikaci dle scénáře v podobě určitých experimentů s přesně daným nastavením.

5.1 Testování webových aplikací

Zde vyvinutý software sestává z třech výše uvedených částí. Každá tato část více méně odpovídá vlastnostem webových aplikací, tedy jedná se o software přístupný webovým prohlížečem. Proto si nejprve osvětlíme teorii testování webových aplikací. Webové aplikace se vyznačují zvláštnostmi, které je odlišují od jiných softwarových aplikací. Tyto odlišnosti ovlivňují testování v mnoha směrech a to může mít za následek složitější testování než u jiných aplikací. Pro efektivní testování webových aplikací je zapotřebí definovat a použít vhodné metody a techniky. Testování se v literatuře rozděluje na více druhů, jejichž pojmenování není jednoznačné.[7]

Výkonnostní testování je obvykle spuštěno simulací různého množství přístupů uživatelů k systému v daném časovém intervalu. Je tedy nutné zpracovat dané množství požadavků. Testujeme, zda se při limitních hodnotách systém navrátí z havarijního stavu a podobně. Informace o přístupu jsou nahrávány a analyzovány. Výkonnostní testování může také sloužit ke kontrole serverové části.[7, 14]

Testování kompatibility slouží k ověření, zda aplikace běží očekávaně na různé kombinaci softwaru, hardwaru a middlewaru. Díky různorodosti internetových prohlížečů je testování kompatibility obzvláště důležité. Je tedy důležité aplikaci otestovat na alespoň těch nejpoužívanějších systémech.[7, 14]

Testování použitelnosti se používá k zjištění, do jaké míry je aplikace snadná k používání. Použitelnost obvykle ovlivňuje design a implementace uživatelského rozhraní. Testování je tedy primárně zaměřeno okolo testování uživatelského rozhraní.[7, 14]

Funkční testování je postup, ve kterém se ověřuje, zda se aplikace chová dle definovaného chování. Do této kategorie testování patří například testování správné funkčnosti JavaScriptu a AJAX dotazů. Součástí tohoto testování je také již zmíněné testování kompatibility.[7, 14]

Bezpečnostní testování se snaží o verifikaci efektivity celkové schopnosti webové aplikace bránit se proti různým neautorizovaným činnostem. Webové aplikace jsou vystaveny většímu nebezpečí než jiné aplikace díky možnosti přistupovat k nim odkudkoli a kýmkoli.[7, 14]

V této práci se zaměříme na výkonnostní testování. Každý jednotlivý experiment se bude snažit testovat následující části systému:

1. Komponentu pro sbírání dat časové řady, tedy rozšíření prohlížeče, budeme testovat v prohlížeči Chrome pomocí nástrojů pro vývojáře. Konkrétně tedy budeme využívat funkce záložek „Performance“ a „Network“.

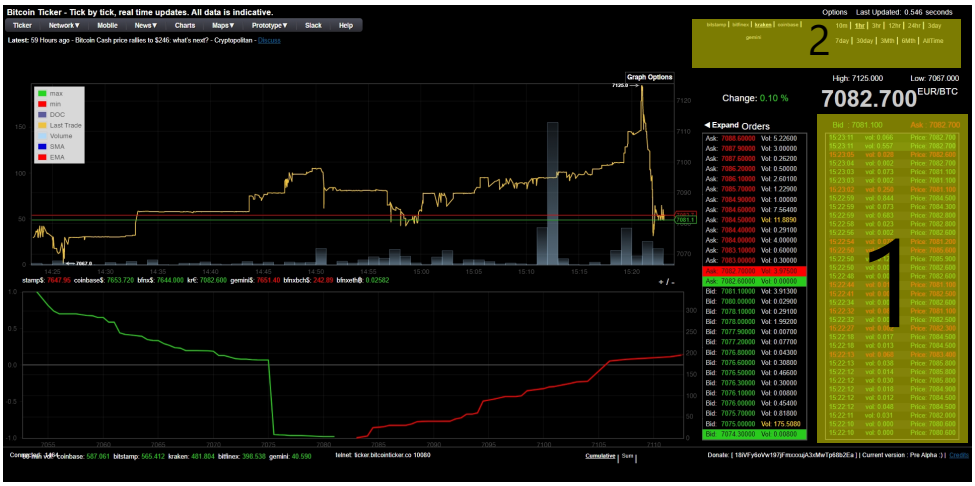
2. Databáze a webové API bude otestováno spuštěním vysokého počtu dotazů. Tento provoz bude na serveru monitorován a následně vyhodnocen.

3. Webová aplikace pro analýzu a vizualizaci dat časové řady bude testována na straně serveru. Frontend testování webové aplikace není náplní této práce.

5.2 Experiment 1

Tento experiment se zaměří na získávání dat časové řady aktuálně provedených transakcí Bitcoinu (BTC). Data budou sbírána ze stránky `https://bitcointicker.co/`. Konkrétně budeme sbírat data burzy „Kraken“.

CoinMate.io je webová stránka, která zobrazuje data různých burz nabízejících směňování kryptoměn, konkrétně Bitcoinu, v reálném čase.



Obrázek 12: Náhled na webovou stránku bitcointicker.co s vyznačenými oblastmi

V Obrázku 12 vidíme vzhled webové stránky `bitcointicker.co`. V levé části se nachází graf vykreslující vývoj ceny a graf nabídek a poptávek. Oba tyto grafy vykreslují data z tabulek v pravé části. Nad těmito tabulkami jsou vyznačeny oblasti 1 a 2. Oblast 2 slouží k zvolení požadované burzy a nastavení časového měřítka dat. Grafy a tabulky se přizpůsobují zvolenému

nastavení. Oblast 1 představuje naše požadovaná data. Jedná se tedy o transakce Bitcoinu, burzy Kraken, s časovým měřítkem 1 hodina.

Pro získávání požadovaných dat z této stránky použijeme pro rozšíření následující JSON konfiguraci:

```
{
    "post_url": "http://127.0.0.1:8000/records/1/insert-records/",
    "user_code_url": "http://127.0.0.1:8000/users/register-user",
    "save_interval": 5000,
    "main_selector": "div.tradeBox",
    "record": {
        "name": "Trade",
        "selector": "div.trowred, div.trowgreen",
        "fields": [
            {
                "name": "Time",
                "selector": "div.ttime",
                "regexp": "none"
            },
            {
                "name": "Volume",
                "selector": "div.tvol",
                "regexp": "/[0-9]*[.]?[0-9]*$/"
            },
            {
                "name": "Price",
                "selector": "div.tprice",
                "regexp": "/[0-9]*[.]?[0-9]*$/"
            }
        ]
    }
}
```

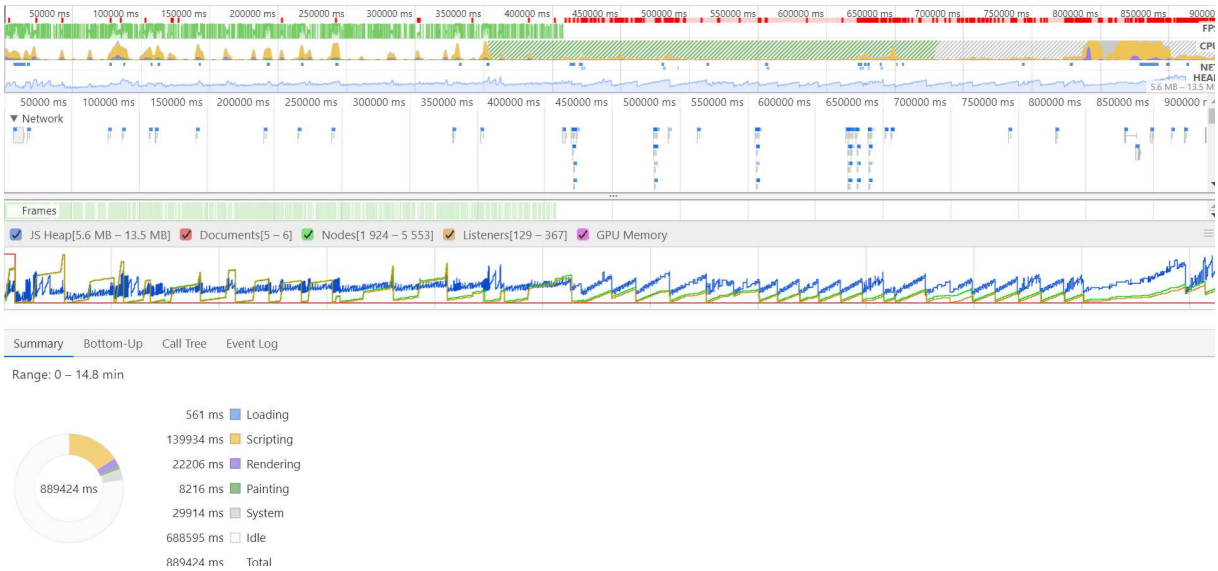
Webový server pro API a R skript je spuštěn na stejném počítači jako rozšíření. Tento počítač má následující parametry:

- Operační systém: Windows 10, 64bit
- Procesor: AMD Ryzen 5 2500U, 2.0 Ghz
- RAM: 16 GB DDR4

PostgreSQL 11.6 databáze běží na sdíleném serveru společnosti ElephantSQL (Ubuntu 7.4.0, 64-bit).

V JSON konfiguračním souboru nejprve nastavíme `save_interval` na 5000, to znamená, že každých 5 sekund se všechny sesbírané data odešlou na URL adresu specifikovanou v `post_url`.

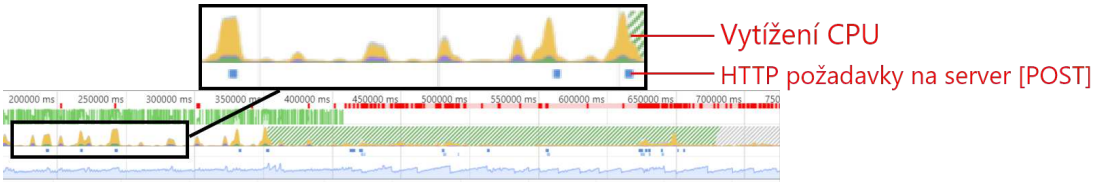
Celou činnost rozšíření si nahrajeme pomocí vývojářského nástroje pod záložkou „Performance“. Po asi 15 minutách získávání dat zastavíme nahrávání ve vývojářských nástrojích, ale rozšíření prohlížeče necháme dále pracovat. Nejprve si tedy zobrazíme výsledek tohoto testování, který lze vidět v níže uvedeném obrázku.



Obrázek 13: Výsledek výkonnostního testování s intervalem odesílání 5 sekund v prohlížeči Chrome

Údajů je opravdu hodně a celý report je velice obsáhlý. Pro lepší pochopení tohoto výkonnostního testování je dobré si přečíst oficiální dokumentaci [15].

Pro nás je zajímavá například první část, ve které lze vidět, jak jednotlivé zasílání požadavků na server ovlivňuje zatížení CPU. Je ale jasné, že hlavní podíl na tomto zatížení má samotná webová stránka, která pro nás data zprostředkovává.



Obrázek 14: Detail obrázku 13

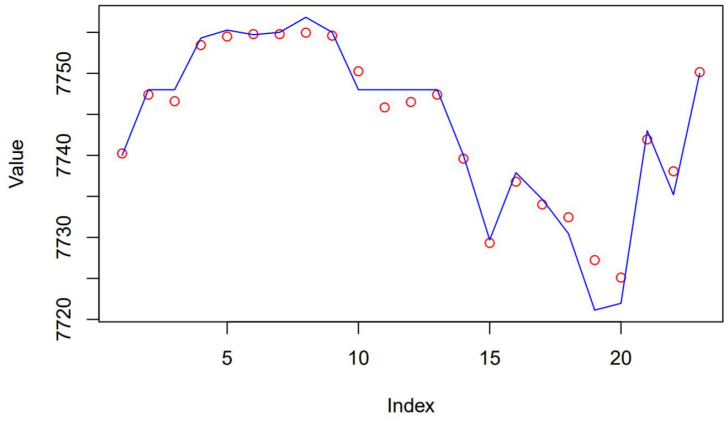
Webový server při obsluze 5 klientů, kteří sbírají data, využívá jen okolo 3% výkonu procesoru a 80 MB paměti RAM. Databázový server stíhá obsluhovat dotazy bez problému. Vytížení systému v tomto experimentu je relativně zanedbatelné a další testování by bylo vcelku zbytečné.

Pro to, abychom opravdu systém zatížili, musíme využít mnohonásobně frekventovanější data. V dalším experimentu tedy upustíme od sbírání dat kryptoměn a využijeme uměle vygenerovaných dat.

Nyní se podíváme na vizualizaci a analýzu dat pomocí webové aplikace, na kterou rozšíření odkazuje. Můžeme vidět, že byly získány údaje od celkem 601 transakcí s průměrnou rychlostí zápisu do databáze asi 5 transakcí za minutu. Tento experiment je zaměřen na vývoj ceny Bitcoinu, důležitým údajem transakce je tedy cena. Zobrazíme si ve webové aplikaci časovou řadu vývoje ceny.

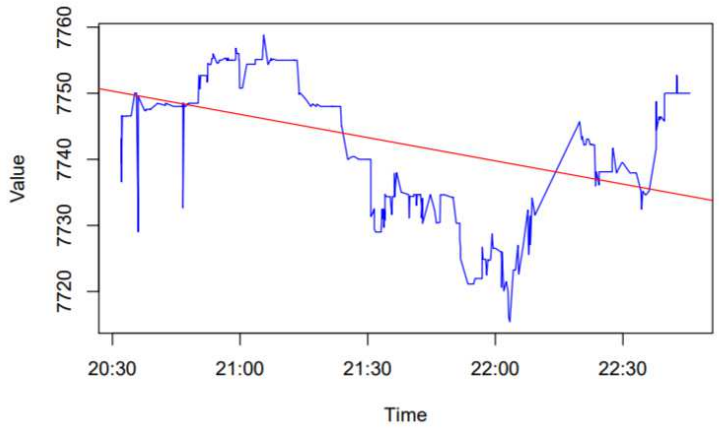
Obrázek 15: Časová řada vývoje ceny Bitcoinu

Dále je možno spustit R skript, který nám tuto časovou řadu vývoje ceny analyzuje. Po několika sekundách trénování modelu pro regresi, a následného vyhodnocení, lze stáhnout celý report analýzy. V reportu lze vidět další předpovídaná hodnota 7750.235, což by mohlo být relevantní například při velice krátkodobém obchodování s kryptoměnami. Dále pak lze vidět graf ukazující úspěšnost modelu nad množinou testovacích dat, který ukazuje, že úspěšnost modelu nad daty transakcí Bitcoinu je poměrně vysoká.



Obrázek 16: Graf testovacích vzorků (červeně - skutečná cena Bitcoinu, modře - předpovězená cena Bitcoinu)

Report dále obsahuje trendovou čáru vyobrazenou na obrázku níže. Trendová čára ukazuje klesání ceny Bitcoinu. To se ve skutečnosti nesplnilo a cena Bitcoinu v následujících hodinách začala stoupat.



Obrázek 17: Trendová čára vývoje ceny Bitcoinu

Tento experiment ukázal celkovou práci všech komponent systému a část také otestoval. Jak již bylo zmíněno výše, pro větší zátěž systému se v dalším experimentu pokusíme o získávání umělých, rychle obnovovaných dat.

5.3 Experiment 2

Jak již bylo výše zmíněno, v tomto experimentu se zaměříme na získávání rychle obnovovaných, uměle generovaných dat. Připravíme si velice jednoduchou webovou stránku pro tento účel:

```
<!DOCTYPE html>
<head>
```

```
    <meta charset="utf-8">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.
        js"></script>
    <title></title>
</head>
<body>
    <div class="container">
        <div class="container-2"><div class="randomval">0</div></div>
        <div class="container-2"><div class="randomval">0</div></div>
        <div class="container-2"><div class="randomval">0</div></div>
        <div class="container-2"><div class="randomval">0</div></div>
        <div class="container-2"><div class="randomval">0</div></div>
        <div class="container-2"><div class="randomval">0</div></div>
        <div class="container-2"><div class="randomval">0</div></div>
        <div class="container-2"><div class="randomval">0</div></div>
        <div class="container-2"><div class="randomval">0</div></div>
        <div class="container-2"><div class="randomval">0</div></div>
    </div>
    <script type="text/javascript">
        setInterval(function () {
            $(".container-2").each(function (i) {
                $(this).children().text(Math.random() * 100);
            });
        }, 200)
    </script>
</body>
</html>
```

Na stránce je umístěno 10 elementů, které pomocí JavaScriptu periodicky mění svou hodnotu na náhodné číslo od 0 do 99. Interval je prozatím nastaven na 200 milisekund. To znamená, že za 1 sekundu běhu rozšíření, se získá 50 náhodných čísel. Za minutu pak 3000. Změnou intervalu pak můžeme kontrolovat, jak rychle chceme data na stránce generovat a tedy zatěžovat více celý systém.

JSON konfigurační soubor, který je využit pro tuto webovou stránku, vypadá následovně:

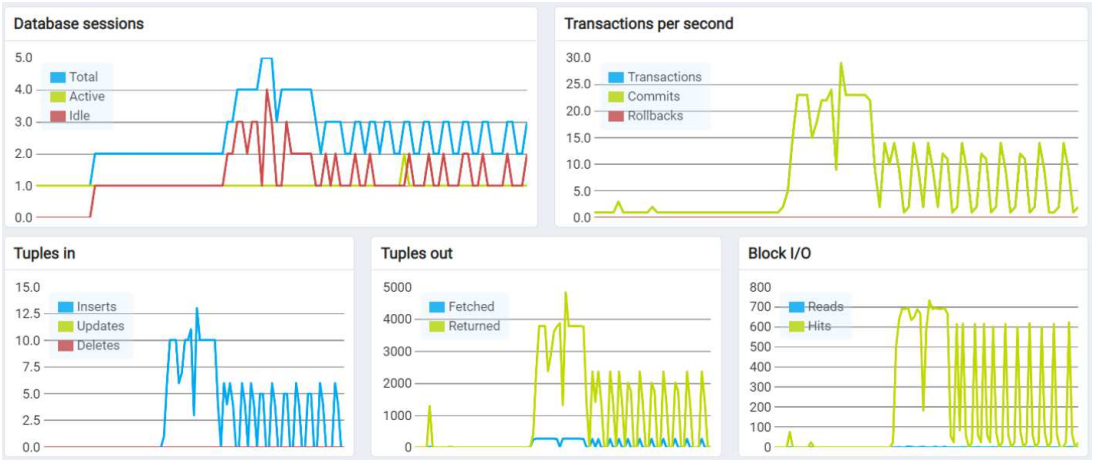
```
{
    "post_url": "http://127.0.0.1:8000/records/1/insert-records/",
    "user_code_url": "http://127.0.0.1:8000/users/register-user",
    "save_interval": 1000,
    "main_selector": "div.container",
    "record": {
```

```
        "name": "RandomNumber",
        "selector": ".container-2",
        "fields": [
            {
                "name": "RandomNumber",
                "selector": ".randomval",
                "regexp": "none"
            }
        ]
    }
}
```

Testování bylo provedeno na třech různých intervalech ukládání při intervalu generování dat 200 milisekund a ve dvou různých intervalech ukládání při intervalu generování dat 10 milisekund. Při nastavení konfiguračního souboru a webové stránky tak, jak je uvedeno výše v konfiguraci, je vytížení klientského počítače nepatrné. Počet požadavků na server je samozřejmě znatelně vyšší, než u předchozího experimentu:

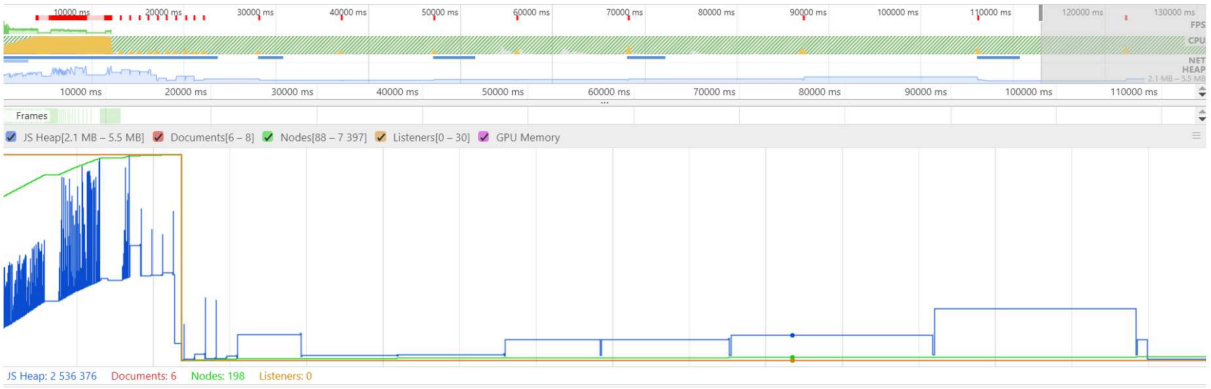
Obrázek 18: Výkonnostní testování - ukázka provozu na síti (interval ukládání 1000ms, interval generování dat 200ms)

Při intervalu ukládání 60000ms jsou kladeny vyšší nároky na paměť, ale jinak jsou rozdíly oproti předchozímu intervalu nepatrné. Problém ale nastává při intervalu 10ms, kdy databázový server odmítl některé požadavky z důvodu zahlcení velkým počtem požadavků. Tento problém nastal při tomto scénáři jen ojediněle a server data jinak bez problému uložil. Pro správu PostgreSQL databáze lze využít například aplikaci PgAdmin, která disponuje také velmi užitečným přehledem. Na tomto přehledu lze nalézt například počet transakcí za sekundu, počet vkládaných záznamů, počet aktualizovaných záznamů, atd. Více ohledně tohoto přehledu, a co přesně znamenají jednotlivé statistiky, lze nalézt v dokumentaci [27]. Tyto statistiky byly podobné u všech možností nastavení, záleželo pouze na tom, kdy klient odesílal požadavky na server. To znamená, zda se jednalo o stálé odesílání dat a rozprostření zátěže, nebo o nárazové vlny požadavků.



Obrázek 19: Přehled statistik databáze v programu PgAdmin 4

Při intervalu generování dat 10 milisekund už ale nastávají vážnější potíže. Webová stránka na které bylo rozšíření spuštěno měla velký pokles FPS a nezávisle na intervalu ukládání dat do databáze, se díky velkému vytížení, přerušilo odesílání požadavků na server a veškeré požadavky tak byly označeny jako „Pending“. Server požadavky přestal zpracovávat a přestal reagovat. Níže vyobrazený obrázek ukazuje rychlý nástup požadavků na server společně s vytížením CPU a paměti (JS Heap). Po asi 15 sekundách běhu lze vidět celkové zhroucení systému a značné omezení odesílání požadavků na server.



Obrázek 20: Výkonnostní test při intervalu generování i intervalu ukládání do databáze 10ms

I přes toto selhání systému si je třeba uvědomit, že při sbírání jakýchkoli dat trhu se nesetkáme s tak frekventovanými daty, jako byla v tomto scénáři, tedy 1000 řádků za sekundu.

Analýza časové řady tvořené náhodnými daty nedává smysl, a tak v tomto experimentu nebude obsažena. Nicméně vzhledem k počtu sesbíraných dat můžeme i tak otestovat čas trénování náhodného lesu, který byl při trénování nad množinou dat o velikosti 2000, asi 6.

6 Závěr

Cílem práce bylo navrhnout a implementovat komponentu a webovou aplikaci, umožňující sběr, analýzu a vizualizaci dat vybraného trhu. Práce se zabývá architekturou systému, použitými technologiemi včetně architektury rozšíření prohlížečů, použitým postupem pro analýzu dat a vizualizaci dat časové řady a ke konci i výkonnostním testováním systému.

Nejprve bylo vyvíjeno rozšíření prohlížeče a ukládání sbíraných dat. V první variantě rozšíření bylo vytvořeno rozhraní, se snahou o uživatelskou přívětivost, ve kterém bylo uživateli umožněno si volit elementy pomocí GUI. Tato možnost se ale ukázala jako neefektivní a značně omezená co se týče vybírání dat. Rozšíření tedy bylo upraveno tak, aby se data vybírala pomocí JSON konfigurace, což se ukázalo jako lepší řešení, ale pozdější práce s rozšířením ukázala určité nedostatky s jednoduššími webovými stránkami, u kterých je někdy složité vybrat správně selektor atributů. Jako způsob ukládání bylo nejprve zvoleno lokální úložiště prohlížeče, ale později byl z důvodu možnosti přístupu k datům odkudkoli zvolen způsob ukládání do databáze. Při tvorbě webové aplikace byl původně pro analýzu využit python, ale nakonec byl použit, díky svému jasnému zaměření na statistiku, jazyk R. Celkově tedy byl cíl práce splněn, ale je zde určitě mnoho možností pro budoucí rozšíření a vylepšení.

Budoucí možné úpravy a rozšíření celku jsou možné spíše u analýzy a vizualizace, než u samotného získávání. Bylo by lepší využít namísto náhodného lesu některou pokročilejší metodu strojového učení, například neuronové sítě. Také by bylo vhodnější vyhodnocovat předpověď rovnou do webové stránky, případně předpovídat hodnotu v reálném čase. V budoucnu by pak bylo dobré, kdyby se uživatel přihlašoval pomocí svého účtu, namísto generování náhodného kódu při každém spuštění rozšíření.

Odkazy

[1] *Angular vs React: Which One to Choose for Your App*. https://www.freecodecamp.org/news/angular-vs-react-what-to-choose-for-your-app-2/.

[2] Søren Bisgaard a Murat Kulahci. *Time series analysis and forecasting by example*. John Wiley & Sons, 2011.

[3] Leo Breiman et al. *Classification and regression trees*. CRC press, 1984.

[4] Apigee Corp. *Web API Design: The Missing Link*. https://docs-apis.apigee.io/files/Web-design-the-missing-link-ebook-2016-11.pdf.

[5] *CSS, Česká terminologická databáze knihovnictví a informační vědy*. https://aleph.nkp.cz/F/MQ4685FAIXS5IKG4TN8PPLSYF3Y9PYMVM1EVDE272299AH6QRT-07187?func=full-set-set&set_number=158481&set_entry=000002&format=999.

[6] Nilanjan Dey et al. *Machine Learning in Bio-Signal Analysis and Diagnostic Imaging*. Academic Press, 2018.

[7] Giuseppe Di Lucca a Anna Fasolino. „Web Application Testing“. In: břez. 2006, s. 219–260. DOI: 10.1007/3-540-28218-1_7.

[8] *Differences between API implementations*. https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Differences_between_API_implementations.

[9] *Django architecture*. https://pythonhosted.org/MyTARDIS/architecture.html.

[10] *Django documentation*. https://docs.djangoproject.com/en/3.0/.

[11] *Django REST framework*. https://docs.djangoproject.com/en/3.0/.

[12] Roy T Fielding a Richard N Taylor. *Architectural styles and the design of network-based software architectures*. Sv. 7. University of California, Irvine Irvine, 2000.

[13] *Firefox browser addons*. https://addons.mozilla.org/cs/firefox/.

[14] *Funkční testování webových aplikací*. https://is.muni.cz/th/h7jvs/bmazoch-bp.pdf.

[15] *Get Started With Analyzing Runtime Performance*. https://developers.google.com/web/tools/chrome-devtools/evaluate-performance.

[16] *Google Chrome docs*. https://developer.chrome.com/extensions/getstarted.

[17] Marijn Haverbeke. *Eloquent javascript: A modern introduction to programming*. No Starch Press, 2014.

[18] *HTML, Česká terminologická databáze knihovnictví a informační vědy*. https://aleph.nkp.cz/F/P3B6BYCESECL5KBK7KIYP3IGIT11T6DS8SHCG4IRVBPL83V9GA-44484?func=find-acc&acc_sequence=000017252.

[19] *Chrome developer devtools*. https://developer.chrome.com/extensions/devtools.

[20] *Chrome developer overview*. https://developer.chrome.com/extensions/overview.

[21] *Internetový obchod chrome*. https://chrome.google.com/webstore/category/extensions?hl=cs.

[22] *MDN Web docs*. https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions.

[23] *MutationObserver*. https://developer.mozilla.org/en-US/docs/Web/API/MutationObserver.

[24] *Návrh webového rozhraní API*. https://docs.microsoft.com/cs-cz/azure/architecture/best-practices/api-design.

[25] V. NOVOTNÝ. *Úvod do počítačových sítí*. Kuřim: Střední odborná škola Kuřim, 2010.

[26] *Observer Design Pattern*. https://docs.microsoft.com/cs-cz/dotnet/standard/events/observer-design-pattern.

[27] *pgAdmin Tabbed Browser*. https://www.pgadmin.org/docs/pgadmin4/3.6/pgadmin_tabbed_browser.html.

[28] *PostgreSQL documentation - json functions and operators*. https://www.postgresql.org/docs/9.5/functions-json.html.

[29] *R documentation - randomForest*. https://www.rdocumentation.org/packages/randomForest/versions/4.6-14/topics/randomForest.

[30] *React documentation*. https://reactjs.org/docs/getting-started.html.

[31] *react-chart-js documentation*. https://openbase.io/js/react-chartjs-2.

[32] *Rozhodovací stromy a lesy*. https://www.iba.muni.cz/res/file/ucebnice/komprdova-rozhodovaci-stromy-lesy.pdf.

[33] George AF Seber a Alan J Lee. *Linear regression analysis*. Sv. 329. John Wiley & Sons, 2012.

[34] Robert H Shumway a David S Stoffer. *Time series analysis and its applications: with R examples*. Springer, 2017.

[35] Chris Smith. *Decision trees and random forests: a visual introduction for beginners*. Blue Windmill Media, 2017.

[36] William S Vincent. *Django for Beginners: Build websites with Python and Django*. Still River Press, 2020.

[37] Robin Wieruch. *The Road to learn React*. CreateSpace Independent Publishing Platform, 2018.

A Ukázka pdf souboru analýzy dat

Data analysis report

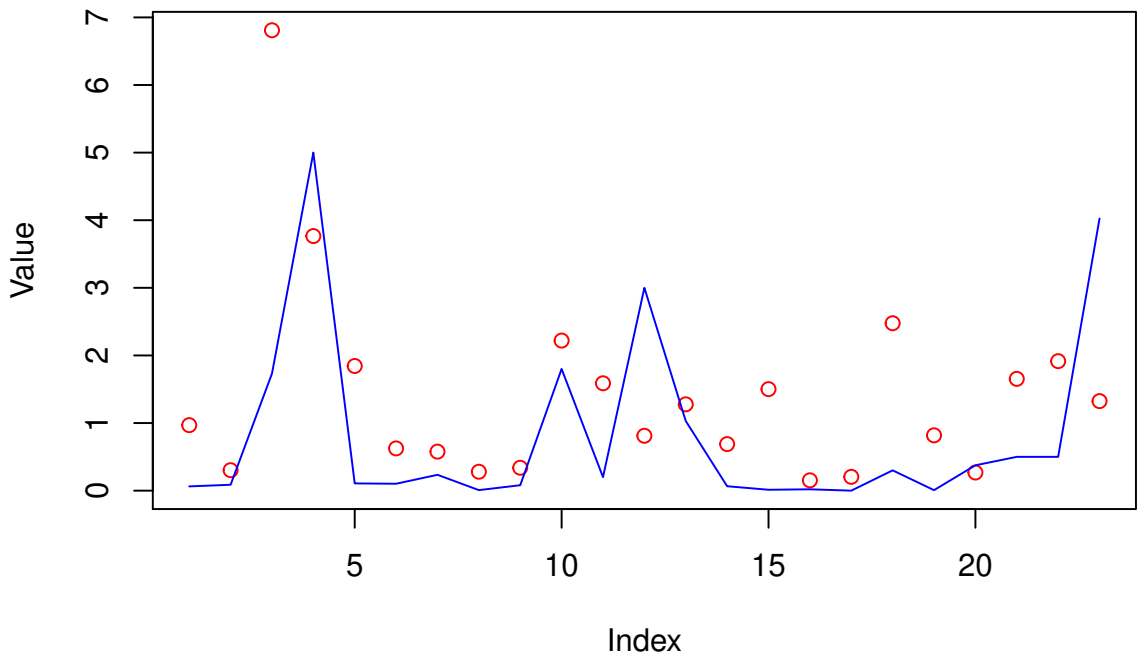
Prediction

The next value predicted by random forest regression model is going to be:

```
## [1] 4.430645
```

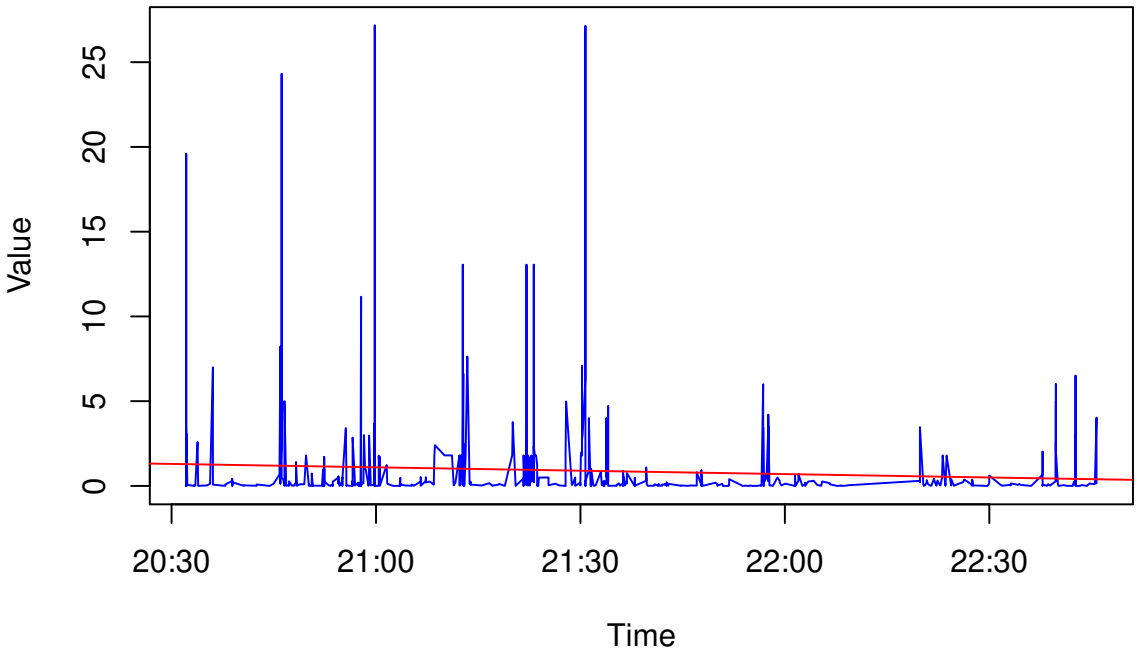
Plot of test samples

The graph below shows real values of test data (red dots) and values predicted by model (blue line).



Trend line

Trend line between values and time.


Value
0
5
10
15
20
25
20:30
21:00
21:30
22:00
22:30
Time