TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Aplikace pro dlouhodobé sledování a analýzu cenové hladiny produktů

Bakalářská práce

Studijní program: B2646 – Informační technologie
Studijní obor: 1802R007 – Informační technologie

Autor práce: **Mojmír Křížek**
Vedoucí práce: Ing. Jan Kraus Ph.D.

Liberec 2018

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Application for long term monitoring and analysis of product price level

Bachelor thesis

Study programme: B2646 – Information technology
Study branch: 1802R007 – Information technology

Author: **Mojmír Křížek**
Supervisor: Ing. Jan Kraus Ph.D.

Liberec 2018

Technická univerzita v Liberci
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2017/2018**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Mojmír Křížek**
Osobní číslo:	**M15000033**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Aplikace pro dlouhodobé sledovaní a analýzu cenove hladiny produktů**
Zadávající katedra:	**Ústav mechatroniky a technické informatiky**

Zásady pro vypracování:

1. Navrhněte systém pro automatické vyhledávání a sledování parametrů nabízených produktů online, zaměřte se na sledování ceny, dostupnosti, velikost skladových zásob apod.
2. Vyhledávání a zadávání produktů by mělo být snadné, systém by měl sám umět nabízet či sledovat alternativy každého výrobku a měl by také umožňovat rozšiřování seznamu sledovaných dodavatelů a portálů.
3. Získaná data agregujte, archivujte a v přehledné formě prezentujte uživatelům, pokuste se vyřešit také otázku sledování cen v různých geografických oblastech a v různých měnách včetně přepočtů dle aktuálních kurzů.
4. V závěru shrňte získané poznatky s ohledem zejména na spolehlivost vyvinuté aplikace a uveďte možnosti dalšího rozvoje tématu.

Rozsah grafických prací: **dle potřeby dokumentace**

Rozsah pracovní zprávy: **30–40 stran**

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] **Heureka Košík - API [online], [cit. 2017-10-10]. Dostupné z: https://sluzby.heureka.cz/napoveda/kosik-api/**

[2] **KURTZ, Jamie, 2013. ASP.NET MVC 4 and the Web API: building a REST service from start to finish. Berkeley, CA: Apress. Expert's voice in ASP.NET.**

Vedoucí bakalářské práce: **Ing. Jan Kraus, Ph.D.**
Ústav mechatroniky a technické informatiky

Datum zadání bakalářské práce: **10. října 2017**

Termín odevzdání bakalářské práce: **14. května 2018**

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

L.S.

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 10. října 2017

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 14. 5. 2018

Podpis: M. Jiřík

Abstrakt

Práce se zaměřuje na tvorbu aplikace pro sledování cen produktů online. Byly využity dva největší eshopy jako zdroje dat - Ebay a Amazon, protože mají API a mají největší sortiment. Součástí aplikace je i rozšiřování skupiny sledovaných produktů, což představovalo největší programátorský problém. Tento proces spoléhá výhradně na Ebay a produkty ze sekce „Ostatní také prohlíželi“ a snaží se vybrat jeden název z této sady názvů produktů, který by měl být právě ten jiný, alternativní produkt. Tento problém byl nejprve řešen vzdáleností řetězců, která nebyla schopna dobře odlišit různé verze produtků. Například názvy `Samsung Galaxy S6` a `Samsung Galaxy S7` představovaly pro vzdálenost řetězců relativně nízkou odlišnost. Použitou metodou se stala frekvenční analýza zohledňující pozici slova v názvu a její implementace zabrala značnou část úsilí. Celkově proces objevení nového produktu měl pravděpodobnost úspěchu přibližně 30%. Tato pravděpodobnost byla ovlivněna několika faktory. V metodách aplikace bylo použito několik parametrů, které by bylo vhodné lépe nastavit. Mimo to se odlišný produkt v sadě podobných produktů nemusí vůbec nacházet. Serverová část aplikace byla napsána v JavaScriptu pro Node.js. Při tvorbě byla kladen důraz na co největší využití syntaxe z nových standardů jazyka, zejména klíčová slova `async` a `await`, která pochází z ECMAScript 2017 standardu. Pro ukládání dat byla použita MySQL databáze, která byla nejprve navržena v softwaru MySQL Workbench. Uživatelská část aplikace byla vytvořena ve frameworku Polymer. Uživatelskou částí se tedy stala jednostránková webová aplikace zobrazující přehled všech produktů. Pro každý produkt jsou zobrazeny grafy minimální, maximální, průměrné ceny a graf dostupnosti produktu.

Klíčová slova: Ebay, Node.js, ECMAScript 2017, Polymer, vzdálenost řetězců, frekvenční analýza

Abstract

This work concentrates on creating the application for monitoring of product prices online. The picked sources of prices are Ebay and Amazon because of their APIs and amount of products listed. The biggest problem was encountered when programming the process of expanding the group of products being monitored. This process entirely relies on Ebay and sources its data from the "Others also viewed"section. The goal is to pick the most different product name from this set of suggested products. There was an attempt to use string distance methods for this picking, but it turned out to be unable to calculate a requested high distance between `Samsung Galaxy S6` and `Samsung Galaxy S7`. The method used after that was frequency analysis taking word position into account. Creating and debugging of this method took quite some effort. The overall probability of discovering new product turned of to be near 30%, additional tweaking of a few internal parameters could increase the probability. Apart from that a different product is not always in the set, which lowers the probability as well. The server side application was written in JavaScript for Node.js. There was a substantial effort to use as much new syntax as possible, especially `async` and `await` keywords, which come from ECMAScript 2017 specification. MySQL database was used to store the data, the database was designed in MySQL Workbench first. The user application was created as a single page web app using Polymer framework. The user application shows an overview of all the monitored products. Charts of lowest, highest and average prices as well as availability charts can be displayed for any product.

Keywords: Ebay, Node.js, ECMAScript 2017, Polymer, string distance, frequency analysis

Obsah

Seznam tabulek

Seznam obrázků

1 Úvod

Práce se zaměřuje na poskytování historie ceny produktů na internetu. Tato historie ceny může být využita například k výběru nejlepší doby pro nákup. Aplikace poslouží jako náhled do problematiky s tím spojené, ukáže způsoby možné implementace.

Historie cen produktů na internetu je informace, kterou někteří zákazníci shledávají užitečnou. Užitečná je hlavně tím, že se mohou dozvědět jaká byla například minimální cena produktu a podle toho produkt buď koupit nebo počkat na cenu lepší. Například portál Heureka u každého evidovaného produktu zobrazuje graf, který ukazuje průběh ceny minimální a průměrné. Pohyby na grafu jsou ovlivněny marketingovými akcemi jako je černý pátek nebo vánoční svátky. Tyto výkyvy by mělo být možné pozorovat i v této aplikaci.

Kontrolování cen produktů je v principu jednoduchá záležitost. Samočinné objevování ještě neevidovaných alternativ k zadaným produktům představovalo těžce řešitelnou překážku. Největší problém bylo zajistit smysluplnou spolehlivost algoritmu. Většina úsilí byla věnována právě metodám využitým v objevování nových produktů. Bylo vyzkoušeno také několik nevhodných algoritmů. V některých částech algoritmu byla vyvinuta metoda, která se snažila dohnat nedostatečnou úspěšnost metody předchozí. Při řešení objevování produktů byl vymyšlen vlastní algoritmus řešící vzdálenosti řetězců.

Celou práci jsem pojal jako příležitost k získání zkušeností s novými prostředími. V budoucnu bych chtěl hledat zaměstnání v oboru využívající tato prostředí. Zároveň jsem využil i svých stávajících znalostí z tvorby webových stránek, programování v jazyce JavaScript a navrhování MySQL databází.

2 Rešerše

Bylo zapotřebí se ujistit o třech základních procesech. Prvním z nich je získávání ceny produktů, což je hlavním smyslem aplikace. Druhým problémem je identifikace produktů mezi různými zdroji, což zaručí správné seskupování sbíraných cen. Třetím řešeným problémem je objevování podobných produktů ke kontrole, což by v ideálním případě měl zvládnout program bez zásahu uživatele.

2.1 Zdroje Cen

Jako zdroje dat byly zvoleny dva největší obchodní portály - Ebay a Amazon. Hlavní výhodou u velkých společností je dostupnost aplikačního programového rozhraní (dále jen API). U eshopů, které neposkytují API, by bylo nutno použít klasickou metodu zpracování HTML (HTML parsování). Tato metoda by extrahovala cenu produktu z webové adresy, kde je produkt zobrazen. Všechny ceny jsou sledovány v amerických dolarech (USD).

2.1.1 Ebay

Ebay poskytuje mnoho druhů API sloužících k různým účelům (například prodávání produktů, marketing, vyhledávání). Pro tuto aplikaci je nejvhodnější vyhledávací API. Vyhledávací API umožňuje mnoho dalších věcí mimo potřeby této aplikace. Použití Ebay API je omezeno na několik API volání za den. Tento limit je závislý především na úrovni přístupu k API (jimi schválené takzvané „Compatible Applications“ mají menší omezení) a také na druhu a účelu jejich API. Většinou mají v běžném přístupu limit 5000 API volání za den.

Dokumentace k Ebay API je rozsáhlá, avšak mnohdy nekompletní nebo zastaralá. Často je v dokumentaci odkazováno na proměnnou, která nebyla přejmenována na všech jejích místech výskytu. Například funkce `findItemsByProduct` umožňuje vyhledání podle UPC kódu, ale databáze, ze které data vrací, je velice omezená. O omezenosti této funkce se člověk dozví až na fóru [6], kde jeden z uživatelů API měl podobný problém.

Pro přístup do API je třeba se autorizovat přiděleným klíčem aplikace. Nejprve se uživatel musí registrovat, poté může na Ebay portálu vytvořit aplikaci. Této aplikaci volí jméno uživatel, Ebay této aplikaci přidělí přístupový klíč (AppId). Tento klíč nahrazuje přihlašování a je nutno ho použít při jakémkoliv volání API. Z tohoto důvodu je vhodné ho nesdílet veřejně. Klíč pro tuto aplikaci vygenerovaný pro jejich

„Production“ prostředí vypadá nějak takto:
`MojmrKek-mkBP2017-PRD-XXXXXXXXX-XXXXXXXX`. Příklad odpovědi z vyhledání UPC kódu na Ebay API je možno nalézt v příloze B.

2.1.2 Amazon

Amazon má svoji platformu Amazon Web Services (AWS), která zaštiťuje mimo jiné i API pro přístup do databáze produktů. AWS umožňuje využívat celou řadu vzdálených služeb. Tyto služby jsou zpravidla placené, proto při registraci do AWS je třeba zadat údaje k platební kartě. V případě API je částka stržena hned po překročení limitů volného užití. Vhledem k omezeným finančním prostředkům a k funkcím, které Ebay API umožňuje, nebude přístup do Amazon AWS nezbytný. Pokus o získání cen z portálu Amazon bude realizován pomocí procházení HTML, což může vést k brzkému zablokování přístupu této aplikace k celému portálu Amazon. Mimo jiné tato metoda může přestat fungovat, bude-li HTML procházených stránek změněno.

2.1.3 Ostatní Eshopy

Bylo by vhodné, aby aplikace nebyla omezena jen na tyto dva zdroje dat. Toto se dá realizovat takto. Každý evidovaný produkt bude mít URL adresu do eshopu. Každý eshop bude mít definovaný způsob vyhledání ceny v HTML. Tento postup využívající URL adresy a HTML identifikátory by měl fungovat na většině eshopů, které API nemají nebo ho nelze použít.

2.2 Identifikace Produktu

Jelikož jsou data čerpána z více zdrojů, je třeba jednotlivé produkty spolehlivě a jednotně identifikovat. K tomuto účelu se nabízí Univerzální kód výrobku (Universal Product Code - UPC) nebo Mezinárodní číslo obchodní položky (European Article Number - EAN). Z důvodů zmíněných v následující kapitole bylo zvoleno UPC. Této identifikace by se mělo využít všude tam, kde je to možné. Ebay API umožňuje vyhledávat pomocí UPC, Amazon API taktéž. Na ostatních stránkách, pro které budou produkty uloženy se svojí URL, nebude třeba vyhledávat tento identifikátor, což pravděpodobně ani většinou není možné.

2.2.1 Porovnání UPC a EAN

UPC je starší, více zažitý, identifikátor, který se skládá maximálně z dvanácti číslic. Oproti tomu EAN je v dnešní době více používaný pro identifikaci produktů a UPC zaštiťuje. EAN se obvykle skládá z třinácti číslic (EAN-13). Pokud se jedná o UPC kód převedený na EAN, pak je první číslice EAN kódu 0. Prostředky na vyhledání EAN kódu k názvu produktu jsou ovšem velice omezené. Z tohoto důvodu bylo použito UPC, na jehož vyhledání existuje prostředků dostatek.

2.2.2 Vlastní Identifkátory Ebay a Amazonu

Je naprosto běžné, že si eshopy zavádějí vlastní identifikátory. Tyto identifikátory jsou obvykle použity pro interní účely (objednávkový systém). Je tomu tak i v případě portálů Ebay a Amazon.

Ebay používá EbayId pro identifikaci jednotlivých vystavení. Tento identifikátor je nutno použít pro vyhledání podobných produktů. Produkt pro tento požadavek nelze specifikovat pomocí UPC. Musí být zvoleno konkrétní vystavení produktu.

Amazon používá ASIN pro identifikaci varianty produktu, je to v podstatě náhrada za UPC/EAN kód. Pro zobrazení vystavení produktu je třeba znát tento identifikátor. Tento identifikátor je zjištěn po každém vyhledání UPC kódu. Pro ušetření počtu přístupů na stránky portálu Amazon je ASIN ukládán vedle každého UPC kódu v databázi.

2.3 Objevování Produktů

V podstatě každý eshop uživateli nabízí další relevantní produkty k aktuálně zobrazovanému. Tohoto by se dalo využít pro samočinné objevování dalších produktů. Ebay tuto sekci u každého produktu zákazníkovi ukazuje. Funkci této sekce lze nalézt v Ebay API. Volání vrací seznam obvykle dvaceti vystavení podobných produktů. Za podobný produkt je považován i stejný model výrobku, takže mezi podobnými produkty je většina vystavení stejný produkt. V tomto seznamu je tedy nutno rozpoznat jiné produkty od původních. Přímo z odpovědi jsou k dispozici všechny atributy listingu (název, cena, ebayId). Pro vizualizaci objevení nových produktů je možno ukládat odkaz na původní produkt, ke kterému byly tyto podobné produkty zjištěny.

2.3.1 Dohledání UPC k názvu produktu

Z úvodu této kapitoly vyplývá, že bude třeba nějak zjistit UPC kódy pro nové produkty z jejich názvu. K tomuto lze využít online UPC databází. Tyto webové aplikace obvykle umožňují vyhledání názvu, pokud známe UPC. V tomto případě je třeba právě opačný směr vyhledávání, ten umožňuje už menší část z těchto online UPC databází. Pokud by byl jako identifikátor produktu zvolen EAN, vedlo by to k problémům s tímto typem dohledání.

Jednou z takovýchto databází je `upcitemdb.com`. Tato stránka poskytuje API přístup, umožňuje vyhledávat název produktu a její databáze je dostatečně obsáhlá. Počet volání API je opět omezen, tentokrát na 100 požadavků za 24 hodin. Bude tedy možné objevit maximálně 100 nových produktů za den, což je dostačující. Případně je možno zakoupit méně omezený přístup k API.

Odpověď z vyhledání názvu produktu v `upcitemdb.com` databázi je možno nalézt v příloze C.

3 Členění aplikace

Neodmyslitelnou částí aplikace je databáze. Do databáze bude vkládat data serverová část. Data z databáze bude zobrazovat uživatelská aplikace. Práce byla tedy rozdělena na tyto tři části:

- Serverová část „Crawler“ - kontrola cen produktů a ukládání do databáze
- Databáze - MySQL nebo MariaDB
- Uživatelská část „Browser“ - prohlížení uložených dat z databáze

3.1 Serverová část aplikace

Úkolem serveru by měla být především kontrola cen evidovaných produktů, jejich identifikace a ukládání těchto dat do databáze. Serverová část by dále měla být schopna sama (nebo s asistencí uživatele) zvětšovat množinu evidovaných produktů. Aplikace by měla ceny sledovat sama, například každý den. Objevení produktu bude vyvoláno správcem.

Tato část se skládá z několika JavaScript souborů, z nichž každý obsahuje funkce týkající se určitého procesu. Například v souboru `discovery.js` jsou obsaženy funkce týkající se objevování nových produktů. Sémantické významy těchto souborů jsou vyznačeny v následující tabulce.

Soubor	Počet řádků	Popis
discovery.js	341	funkce pro objevení produktu
database.js	281	třída databáze a SQL dotazy
checkers.js	151	funkce na kontrolu cen
app.js	121	hlavní konzolová aplikace (menu, kontrola cen)
apiServer.js	74	API koncový bod pro uživatelskou aplikaci
merger.js	26	funkce pro slučování produktů

Tabulka 1: Seznam souborů serverové části aplikace

3.2 Databáze

Úkolem databáze by přirozeně mělo být uchovávání dat o evidovaných produktech a informací sloužících k identifikaci produktu, vedení záznamů o akcích v serve-

rové části (log) a především uchovávání historií cen produktů. Z důvodu potenciální komplexnosti aplikace by se mělo jednat o SQL databázi, kde je možno využít databázových relací, cizích klíčů a pohledů.

3.3 Uživatelská část aplikace

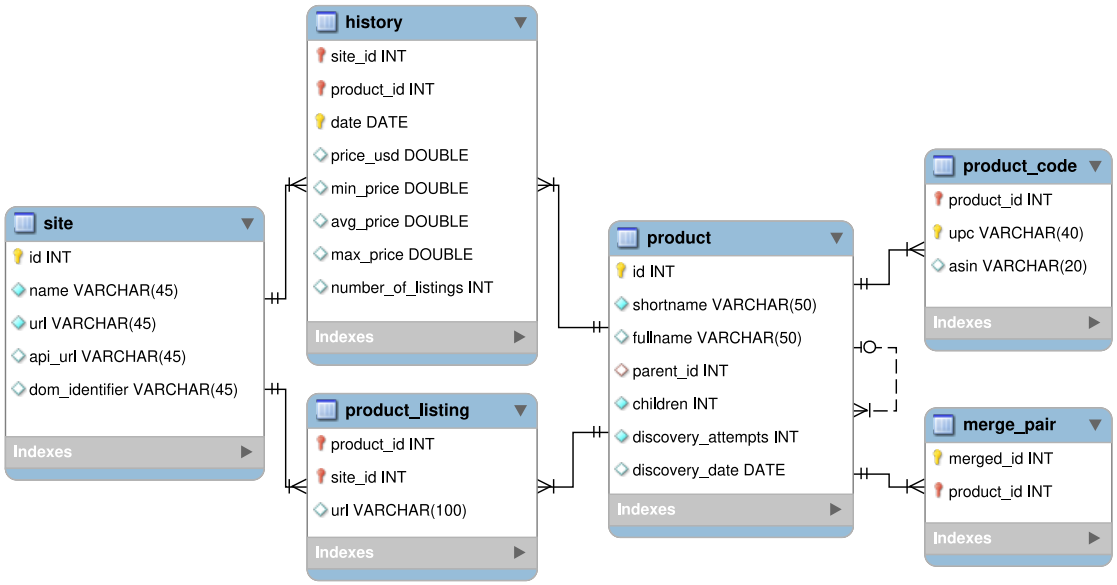
Uživatelská část by měla uživateli poskytovat přehled o všech produktech, zobrazovat historii ceny jako jeden nebo několik grafů, případně i umožnit vyhledání produktu podle jeho názvu.

4 MySQL databáze

Při ukládání dat je využití databázového systému samozřejmostí. Tím nejběžnějším je MySQL a jeho open source klon MariaDB. Pro MySQL existuje software MySQL Workbench, který umožňuje správu i návrh databázové struktury a následnou dopřednou konstrukci (forward engineering) celé databáze. Tento software je kompatibilní s použitým MariaDB klonem. Software tohoto typu byl použit především pro návrhu databáze.

4.1 Návrh

Struktura databáze byla navržena tak, aby bylo možné přidat zdroj cen, vytvořit strom objevených produktů nebo případné duplicitní produkty sloučit.



Obrázek 1: Schéma navržené databáze

4.1.1 Tabulka product

Tato tabulka obsahuje základní informace o produktu spolu s pomocnými informacemi k procesu objevení nových produktů. Například atribut `parent_id` je vazbou na jiný produkt, ze kterého byl tento produkt objeven. Atribut `children` představuje počet úspěšně objevených produktů z tohoto produktu. Je to jeden z mála

atributů, které není přímo nutné ukládat. Počet potomků produktu by bylo možné zjistit z tabulky díky již existujícímu atributu `parent_id`. Vzhledem k faktu, že atribut `children` je aktualizován pouze po úspěšném objevení nového produktu a čten pouze při výběru nejlepšího kandidáta na produkt pro spuštění objevení, byl pro zjednodušení a ušetření případného nahrazujícího dotazu ponechán v tabulce.

4.1.2 Relace product na product_code

Tato relace umožňuje přiřazení více UPC kódů k jednomu produktu. Aplikace se snaží varianty produktů označené pokaždé jiným UPC kódem sloučit do jednoho produktu.

4.1.3 Relace product na product

Při objevování nového produktu jsou zjištěny podobné produkty k již evidovanému produktu. Po úspěšném objevení nového produktu je tedy možno uchovat informaci o tom, ze kterého produktu byl tento nový produkt objeven. Tato vazba umožňuje vytvořit strom objevených produktů.

4.1.4 Tabulka site

Tato tabulka obsahuje seznam zdrojů ze kterých je možno zjistit cenu pro alespoň jeden z evidovaných produktů. Základní dva záznamy v tabulce jsou Ebay a Amazon. Pro ostatní portály, které nemají žádné API, je možno uložit obecnou adresu pro zobrazení produktu (atribut `url`) a spolu s tím i HTML identifikátor elementu, který bude cenu obsahovat. Tato část aplikace ovšem nebyla nijak realizována, úsilí bylo soustředěno pouze na Ebay a Amazon a především na automatické objevování nových produktů.

4.1.5 Tabulka history

Tato tabulka obsahuje klíčové informace celé aplikace. Každý záznam je identifikován identifikátorem zdroje (atribut `site_id`), identifikátorem produktu (atribut `product_id`) a datem zjištění ceny. O produktu na určitém portálu je pro každý den zaznamenávána minimální, průměrná a maximální cena a dostupnost (atribut `number_of_listings`). V případě, že jedná o produkt z konkrétní adresy mimo Ebay a Amazon, je možno konkrétní cenu uložit do atributu `price_usd`.

4.1.6 Tabulka product_listing

Tato tabulka plní spíše pomocný účel. Určuje, který produkt je evidován na kterém portálu, což je informace, která v databázi není nikde jinde uložena. Tato informace je využita při kontrole ceny. Při kontrole ceny určitého produktu je cena zjištěna ze všech portálů, na kterých je dle této tabulky daný produkt evidován. Obvykle

jsou v této tabulce jen dva záznamy pro každý produkt (jeden záznam pro Ebay a druhý pro Amazon).

4.1.7 Tabulka merge_pair

Tato tabulka není z hlediska funkce aplikace příliš důležitá. Její záznamy slouží ke zpětnému dohledání informací o produktech, které byly sloučeny do jiných, pravděpodobně stále existujících produktů. Tyto informace mohou mít svou podstatu při procházení záznamů o běhu aplikace. V těchto záznamech může být odkazováno na identifikátor produktu, který už nemusí v aktuální databázi existovat. V tomto případě je možno z této tabulky postupně vyčíst, se kterými produkty byl tento odkazovaný produkt ze záznamů sloučen. Tímto postupem je možno dostat například název produktu, který dle záznamů působil často problémy například při kontrole cen.

4.2 Pohledy

Pohled (anglicky view) je v databázových systémech zjednodušeně řečeno uložený dotaz. Definice pohledu je složena z několika klíčových slov určujících to, že se jedná o pohled a definujících jméno toho pohledu. Zbytek definice je samotný dotaz. K pohledu je poté možno přistupovat jako ke kterékoliv tabulce, ovšem pouze pro výpis (příkaz `SELECT`) dat. V databázi byly vytvořeny hned čtyři pohledy. Podrobně popsán bude jen ten nejzajímavější.

4.2.1 Výběr nejvhodnějšího produktu pro pokus o objevení

Tento pohled (pojmenován `best_product_view`) je použit hned na začátku algoritmu pro objevení produktu (krok 1.). Pohled není použít v případě, že funkci pro objevení nového produktu není předán identifikátor produktu jako argument. Cílem tohoto pohledu je vrátit seznam produktů uspořádaných podle jejich úspěšnosti v procesu objevení nového produktu. Pohled by měl upřednostňovat nově objevené produkty, pokud ještě tento produkt nebyl použit pro objevení. Zároveň by produkty s mnoha pokusy o objevení měly být až na konci seznamu. Upřednostňování produktů s těmito požadavky je možno dosáhnout výpočtem ohodnocení pro každý produkt a následným sestupným seřazením dle tohoto ohodnocení.

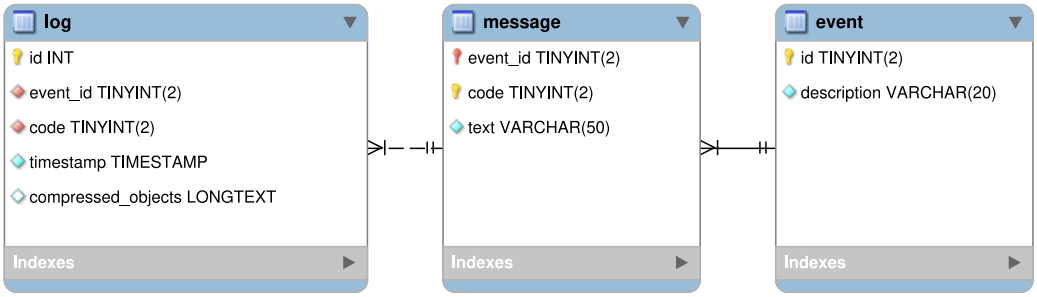
```
CREATE VIEW c_DB.`best_product_view` AS
    SELECT
        id,
        shortname,
        children,
        discovery_attempts,
        (children + 1) / (discovery_attempts + 1) AS success_rate
    FROM
        c_DB.product
    ORDER BY success_rate DESC;
```

Obrázek 2: Definice pohledu pro výběr nejlepšího produktu pro objevení

4.3 Záznamy o běhu aplikace

Jelikož proces objevení nového produktu může skončit neúspěchem v mnoha různých stavech, je vhodné si nějak zaznamenávat vše, co se stalo a který problém zabránil objevení nového produktu. Další motivací byl sběr trénovacích dat pro případnou neuronovou síť, kterou by bylo možno využít na rozpoznání odlišného produktu (viz sekci 7.1). Například pokud v kroku 10 nedojde ke kolizi UPC kódu, byl nový produkt z kroku 4 zvolen správně. V porovnání s tím, pokud v kroku 6 je vyhodnocen zvolený nový produkt z kroku 4 jako stejný, pak je možno považovat výsledek metody vybrání nového produktu za chybný. Situaci ovšem komplikuje fakt, že v Ebay nemusí opravdový jiný produkt do sady v kroku 3 vůbec zařadit.

Pro tento účel byla navržena další databáze, při jejímž návrhu byla předpokládána možnost zaznamenávání výsledků z kontroly cen a sloučení produktů. Obsah této databáze může potenciálně vzrůst na poněkud velké velikosti, ostatně jako každý záznam o běhu (log). Z tohoto důvodu byl kladen důraz na co nejmenší duplicitu dat a z toho vyplývající minimální velikost obsahu.



Obrázek 3: Schéma databáze pro záznamy o běhu

5 Serverový JavaScript a nová syntaxe

Jazyk se kterým mám největší zkušenosti a zároveň jsem v něm periodické kontrolování internetových stránek už vytvářel, je Javascript. Tento jazyk se k tomuto účelu hodí nejvíce, protože dokáže porozumět struktuře webové stránky a posílání HTTP požadavků (zpravidla na API jiné stránky) je taktéž běžné. I přesto, že tento skriptovací jazyk se používá hlavně na webových stránkách, je možné ho použít i na serveru, kde by se dříve běžně použil jazyk PHP.

Pro Javascript na serveru je jasnou volbou Node.js. Jedná se prostředí programovatelné v Javascriptu s rozšířenou funkcionalitou pro poskytování HTML stránek (odpovídání na HTTP požadavky). Tyto stránky nejsou automaticky považovány za dynamické, jako tomu je u PHP. Propojení s MySQL/MariaDB databázemi není problém. Node.js používá systém balíčků, typicky je správcem těchto balíčků Node Package Manager (npm).

5.1 Standardy jazyka

Jazyk JavaScript je neustále vyvíjen, k čemuž přispívá možnost použití jazyka JavaScript na serverech skrze Node.js. V podstatě každý rok vychází nový standard jazyka, tyto standardy jsou označovány například ECMAScript 2017, názvy standardů jsou často zkracovány na například ES2017. Také se někdy používá označení standardů z hlediska verze nebo čísla vydání, pro ES2017 je to 8 (ES8 nebo ECMAScript 8).

5.2 Balíčkovací systém npm

Node Package Manager je tím nejběžnějším a nejrozšířenějším balíčkovacím systémem pro Node.js. Je to obdoba balíčkovacích systémů na UNIX operačních systémech. Při zakládání Node.js aplikce je často doporučováno `npm init`, což vytvoří základní minimální strukturu aplikace. Jedním z důležitých souborů je `package.json`. Tento soubor obsahuje informace o balíku (každá aplikace je považována za balík) a také závislosti tohoto balíku. Seznam závislostí je seznam balíků, které jsme do aplikace nainstalovali, typicky jsou to knihovny.

5.2.1 Závislosti aplikace

Balíky se instalují příkazem `npm install nazevBaliku`. Balíky jsou instalovány do složky `node_modules`, která je mezi programátory pověstná svou mnohdy nezanedbatelnou velikostí nebo také počtem složek a souborů, které obsahuje. Při vývoji aplikace s využitím systému správy verzí (typicky Git) je nemyslitelné nahrávání složky `node_modules` na server systému správy verzí. Pokud si takto distribuovanou aplikaci chce spustit někdo cizí, musí si vedle stažení zdrojového kódu stáhnout také závislosti aplikace. Tyto závislosti je možno automaticky nainstalovat příkazem `npm install`, který si přečte seznam závislostí ze souboru `packager.json`. Vývojářem instalované balíky jsou do tohoto souboru zaznamenány spolu s jejich instalovanou verzí, což eliminuje závislost na dodržení zpětné kompatibility použitých knihoven.

5.3 Zjednodušení asynchronních volání

Programovací jazyk JavaScript vyniká způsobem řešení asynchronní volání. Asynchronní volání může být například zápis na disk, HTTP požadavek nebo provedení databázového dotazu. Pokud chceme vykonat určité řádky kódu po vykonání asynchronní operace, musí být předány asynchronnímu volání v takzvané callback funkci. Toto znamená pro mnoho nových programátorů překážku. Běžně by se dalo očekávat, že řádky za asynchronním voláním budou vykonány až po dokončení asynchronní operace, avšak tomu tak není. Tato nutnost použití callback funkcí měla za následek zbytečné zanoření kódu, což při komplexních algoritmech snižovalo přehlednost kódu. Pro porozumění těmto novým syntaxím popsaných v následujících kapitolách byl použit [3].

5.3.1 Promise

S příchodem ECMAScript 2015 (ES6), který přinesl takzvaný Promise, se situace začala zlepšovat. Programátoři už nebyli nuceni zanořovat kód závislý na výsledku asynchronního volání do callback funkcí. Avšak syntaxe pořád nebyla moc čitelná.

Promise je objekt, který se podobá funkci. Promise má tělo a může vrátit hodnotu nebo vyhodit výjimku. V těle je obvykle nějaká asynchronní operace. Na Promise, jako objekt, je možno počkat v místě volání a navázat na to další řádky kódu, které se vykonají po dokončení těla Promise (po vrácení hodnoty).

```
class DatabaseConnection {
  constructor () {
    ...
    this.connection = mysql.createConnection(options)
  }
  query (sql, args = []) {
    return new Promise((resolve, reject) => {
      this.connection.query(sql, args, (error, result) => {
        if (error) reject(error)
        resolve(result)
      })
    })
  }
}
```

Obrázek 4: Definice třídy pro databázové přípojení využívající `Promise` objekt

V této ukázce kódu aplikace je definována třída `DatabaseConnection`, jejíž metoda `query()` vrací Promise objekt. Tento Promise může vrátit hodnotu pomocí metody (zpravidla pojmenované) `resolve()` nebo může skončit neúspěchem zavoláním metody `reject()`.

```
function showProducts() {
  let db = new DatabaseConnection()
  let sql = 'SELECT ...'
  db.query(sql).then((result) => {
    return processResult(result)
  }).then((processedResult) => {
    showResult(processedResult)
  }).catch((error) => {
    logError(error)
  })
}
```

Obrázek 5: Ukázka navázaní kódu na asynchronní operaci bez `await`

Tento kousek kódu ukazuje možné navazování dalšího kódu na událost dokončení objektu Promise pomocí funkce `Promise.then()`. Funkce `Promise.catch()` slouží k zachycení výjimek.

5.3.2 Async/Await

Situaci elegantně vyřešila klíčová slova `async` a `await` obsažena ve standardu ECMAScript 2017. Díky těmto dvěma klíčovým slovům už celý je proces čekání na dokončení asynchronní události mnohem čitelnější. Pokud by aplikace byla psána o pouhý rok dříve, většina jejího kódu by vypadala poněkud jinak.

Objekt Promise je zde také nutno použít, avšak syntaxe pro čekání na dokončení využívá nových klíčových slov a jednoduší syntaxe. Funkce různých knihoven

obvykle existují už v „promisifikované“ („promisified“) variantě a je vhodné tyto varianty vyhledávat. Například balík `request` existuje i ve variantě využívající objekt Promise pod názvem `request-promise-native`.

Na objekt Promise je nově možno počkat pomocí klíčového slova `await`. Výjimky lze zachytit klasickým způsobem pomocí `try` a `catch`. Jakákoliv metoda používající `await` se stává asynchronní a toto je nutno označit klíčovým slovem `async` před `function`. V takto označených metodách jsou všechna vrácení hodnot pomocí `return` automaticky nahrazena za vrácení objektu Promise. Toto není v následující ukázce kódu využito.

```
async function showProducts() {
  let db = new DatabaseConnection()
  let sql = 'SELECT ...'
  let result
  try {
    result = await db.query(sql)
  } catch (error) {
    logError(error)
    return
  }
  let processedResult = processResult(result)
  showResult(processedResult)
}
```

Obrázek 6: Ukázka navázaní kódu na asynchronní operaci s využitím `await`

Tato ukázka je přepsání kódu 5 pomocí klíčových slov `async` a `await`. K zavolání smyšlené funkce `showResult()` dojde až po dokončení asynchronního volání `db.query()`.

5.4 Operátor rozvinutí

Tento operátor je syntakticky vyjádřen jako tři tečky předcházející názvu proměnné, dá se tudíž označit za „trojtečku“. Tento operátor se používá pro rozbalení prvků pole do argumentů funkce, ale není to jeho jediné využití.

```
let numbers = [-100, 1, 2, 3, 100]
let maximum = Math.max(...numbers)
```

Obrázek 7: Ukázka použití operátoru rozvinutí pro spočítání maxima

V tomto příkladu vidíme jednoduché použití pro spočítání maxima z pole čísel. Předpokladem pro použití je, že funkce `Math.max()` přijímá neurčitý počet argumentů, ze kterých vybere prvek nabývající maximální hodnoty a vrátí ho. Hodnota proměnné `maximum` tedy bude `100`.

5.5 Líné vyhodnocování

Tímto označením je myšlena jednoduchá logika kódu bez nutnosti podmínek. Tyto zkratky jsou obvykle použity pro definování implicitních hodnot proměnných, avšak je možno je použít i jako velice efektivní náhradu podmínek, jejichž jedna větev by vedla k ukončení vykonávaného bloku kódu. Oba tyto případy použití jsou demonstrovány v následujících příkladech.

5.5.1 Výchozí hodnota proměnné

```
function foo(argument1) {
  let n = argument1 || 10
}
```

Obrázek 8: Ukázka použití líného vyhodnocování pro výchozí hodnotu

Zde je použito líné vyhodnocování. Proměnné `n` je přiřazena v podstatě první hodnota, kterou je možno vyhodnotit jako `true`. Je-li alespoň jeden z operandů operátoru disjunkce vyhodnotitelný na `true`, pak je vyhodnocování ukončeno, výraz vyhodnocen a proměnné `n` přiřazena hodnota.

5.5.2 Náhrada jednoduchého větvení

```
function printSqrt(n) {
  console.log(Math.sqrt(n))
}

let n = 10
n >= 0 && printSqrt(n)
```

Obrázek 9: Ukázka použití líného vyhodnocování místo větvení

V této ukázce je demonstrováno podmíněné spuštění funkce. Předpokládejme funkci `printSqrt(n)`, která pro zjednodušení vypíše druhou odmocninu z daného čísla. Počítat druhou odmocninu má smysl jen pro nezáporná čísla, proto před volání funkce dáme podmínku, která toto ověří. Líné vyhodnocování v tomto případě skončí s vyhodnocováním výrazu, pokud první podmínka nebude vyhodnotitelná na `true`. V případě splnění první podmínky vyhodnocování pokračuje na pravou stranu výrazu, kde zavolá funkci s argumentem `n` nabývající nezáporné hodnoty.

5.6 Ladění kódu

Snad žádný programátor si nedokáže představit vyvíjení komplexní aplikace bez možnosti ladění. Programátoři často nahrazují funkci ladění výpisy do konzole, což je ve většině případů neefektivní postup. Zvláště pak, pokud se aplikace záhadně přeruší někde uprostřed a programátor začne do kódu vkládat mnoho výpisů čísel, jen aby zjistil, která část kódu se provede a která už ne. Node.js umožňuje dva způsoby ladění aplikace.

5.6.1 Ladění v terminálu

Ladění v terminálu může být pro mnoho programátorů poněkud nepředstavitelné. V dnešní době se pojmem ladění typicky myslí ladění ve vývojovém prostředí (v IDE), kde uživatel vidí několik oken obsahující informace o pozici v kódu, seznam aktuálně definovaných proměnných a mimo jiné i tlačítka pro ovládání běhu programu. Avšak v dobách, kdy ještě vývojová prostředí neexistovala, bylo potřeba nějak ladit program. Už v roce 1986 napsal Richard Stallman GNU Debugger (gdb), který nebyl první svého typu, ale stal se na UNIX systémech standardem. Většina konzolových ladících programů (Node.js Debugger, The Python Debugger pdb) sdílí ovládání právě s GNU Debugger. Ladění aplikace je možno spustit příkazem `node inspect script.js`. Dokumentace k Node.js Debugger je dostupna na [5].

Uživateli se při každém kroku programu v terminálu vypisuje aktuální řádek a jeho blízké okolí. Uživatel ovládá běh programu příkazy v konzoli, ladící program někdy umožňuje i přidání proměnných do takzvaného watchlistu, proměnné jsou při každém kroku programu vypsány.

```
debug> n
break in discovery.js:271
Watchers:
  0: parentId = 24

 269   debugger
 270   parentId = parentId || await database.getBestProductForDiscovery()
>271   await database.discoveryStarted(parentId)
 272   let productCodes = await database.getProductCodes(parentId)
 273   shuffle(productCodes)
debug>
```

Obrázek 10: Ukázka ladění Node.js aplikace v terminálu

Tabulka 2: Základní ovládání ladění v terminálu

Příkaz	Poznámka	Význam
c	continue	pokračovat do dalšího breakpointu
n	step over	přejít na další řádek v souboru
s	step into	přejít na další řádek nebo vstoupit do funkce
o	step out	vystoupit z funkce
r	restart	spustit program znovu
repl		otevřít JavaScript konzoli na aktuálním řádku
list()		vypsat blízké okolí kódu
sb()	set breakpoint	zapnout breakpoint na tomto (nebo daném) řádku

5.6.2 Ladění skrze Chrome Developer Tools

Druhý a pohodlnější způsob ladění Node.js programů je použití Chrome DevTools. Příkazem `node --inspect-brk script.js` je možné spustit ladící program v režimu naslouchání. Na tuto instanci ladícího programu je třeba se připojit, k tomu je třeba webový prohlížeč založený na projektu Chrome (nebo Chromium). Po otevření `chrome://inspect` adresy je nutno vybrat lokální instanci ladícího programu. Poté se otevře okno Chrome DevTools, kde programátor vidí svůj kód zastavený na prvním řádku. Samotné ladění programu v Chrome DevTools už patří ovšem do jiné kapitoly.

6 Kontrola cen

Kontrola cen představovala ten menší problém, nebyla předem známa žádná překážka. Kontrola ceny by měla proběhnout vždy jednou za den pro daný produkt. Proces nejprve musí zjistit z tabulky `product_listing`, na kterých zdrojích cen je produkt evidován. Tyto záznamy budou zpravidla dva (Ebay a Amazon), tato tabulka spolu s tímto postupem ovšem předpokládá i možný výskyt produktu na jiném zdroji s danou přímou adresou. Nezávisle na tom je třeba zjistit všechny evidované UPC kódy tohoto produktu z tabulky `product_codes`. Nyní je třeba pro všechny (Ebay a Amazon) zdroje tohoto produktu, které umožňují vyhledání UPC kódů, provést vyhledání těchto UPC kódů na daném zdroji/portálu s cílem získat seznam vystavení produktu. Z každého vystavení je možno přečíst jeho cenu. Vždy lze vyhledat právě jeden UPC kód na daném portálu. Výsledkem tohoto je obecně několik (v tomto případě dva) seznamů cen, ze kterých se spočítá minimum (viz 7), maximum a průměr. Dostupnost je vzata jako počet prvků těchto seznamů. Tyto skupiny hodnot, každá pro jeden portál, jsou uloženy do tabulky `product_history`. Tento postup je opakován pro každý kontrolovaný produkt (zpravidla všechny).

Pro vyhledání UPC kódu na portálu Ebay, byl využit požadavek `findItemsAdvanced` (viz dokumentaci [4]).

6.1 Ukázka kódu pro Ebay

Nevýhodou využívání API více portálů je, že každé API se ovládá a chová jinak. Proto musely být napsány funkce pro Ebay a Amazon. V ukázce 11 je zobrazena funkce pro portál Ebay. Funkce pro API konkrétního portálu vždy přijímá pole UPC kódů (a referenci na databázové připojení `database`) a vrací pole zjištěných cen.

```
async function ebay (productCodes, database) {
  let prices = []
  for (let row of productCodes) {
    let pricesOfOneUPC = []
    let UPC = row.UPC
    let appId = 'MojmrKek-mkBP2017-PRD-xxxxxxxxx-xxxxxxxx'
    let url = 'http://svcs.ebay.com/services/search/FindingService/v1?OPER...'
    let result = await request.get({
      url,
      json: true,
      headers: {
        'User-Agent': 'request'
      }
    })
    let items = result.findItemsAdvancedResponse[0].searchResult[0].item
    if (!items) return []
    for (let item of items) {
      let price = item.sellingStatus[0].currentPrice[0].__value__
      pricesOfOneUPC.push(parseFloat(price))
    }
    prices.push(...pricesOfOneUPC)
  }
  return prices
}
```

Obrázek 11: Funkce zjišťující ceny z portálu Ebay

Seznam vystavení je v proměnné `result` dostupný na indexu `.findItemsAdvanced Response[0].searchResult[0].item`. Cena je v každém z vystavení dostupná na indexu `.sellingStatus[0].currentPrice[0].__value__`. Proměnná `appId` je jakýsi klíč pro přístup do API (viz sekci 2.1.1). Proměnná `url` obsahuje API adresu, která bude volána. V obsahu této proměnné je využito několik proměnných a nastaveno několik filtrů vyhledávání.

```
'http://svcs.ebay.com/services/search/FindingService/v1?OPERATION-NAME=
    ↪ findItemsAdvanced&SERVICE-VERSION=1.13.0&GLOBAL-ID=EBAY-US&SECURITY-
    ↪ APPNAME=' + appId + '&RESPONSE-DATA-FORMAT=JSON&itemFilter(0).name=
    ↪ Condition&itemFilter(0).value(0)=1000&itemFilter(1).name=ListingType&
    ↪ itemFilter(1).value(0)=FixedPrice&keywords=' + UPC
```

Obrázek 12: Celá URL adresa sestavená v proměnné `url`

- `findItemsAdvanced` určuje konkrétní funkci z Ebay API.
- `Condition` = `1000` omezuje výsledky jen na nové produkty (`New` na portálu Ebay).
- `ListingType` = `FixedPrice` dále vybírá jen vystavení s možností okamžitého nakoupení.
- Proměnná `UPC` obsahuje jeden konkrétní UPC kód.

7 Algoritmus objevení produktu

Postup pro objevení nového produktu představoval největší překážku v celém projektu, jelikož měl v ideálním případě fungovat bez zásahu uživatele. Tento proces je plně závislý na portálu Ebay a mimo jiné zcela závisí i na názvech vystavení produktů. Z toho plyne komplikace způsobená samotnými prodejci produktů, kteří do názvů přidávali nic neříkající slova (například „very“, „now“ nebo „super“), používali nestandardní oddělovače (pomlčky, znak &) nebo oddělovač v některých případech úplně vynechali. Algoritmus byl navrhován průběžně při implementaci. Postup je možné shrnout do několika kroků:

1. vyber nejlepší produkt z databáze a vrať všechny jeho UPC kódy,
2. pro náhodný UPC kód vyber na Ebay portálu náhodné EbayId,
3. vrať odpověď z getSimilarItems z Ebay API pro tento produkt,
4. **vyber ten nejodlišnější (nový) produkt**,
5. vrať seznam podobných produktů pro nový produkt,
6. **porovnej podobnosti názvů, pokud jsou nepodobné, pokračuj**,
7. vyber nejlepší název pro nový produkt,
8. vyhledej UPC kódy pro tento nejlepší název v `upcitemdb.com` databázi,
9. vyber ty UPC kódy, jejichž název je dost podobný vyhledávanému,
10. detekuj kolize UPC kódů, pokud nenastane, pokračuj,
11. ulož produkt a seznam jeho UPC kódů do databáze.

Podrobně budou popsány jen ty nejzajímavější kroky algoritmu - krok 4 a krok 6. Celou funkci `discoverNewProduct()` je možno nalézt v příloze A.

7.1 Krok 4 - Výběr nejodlišnějšího názvu produktu

Tento krok představoval největší problém v celé aplikaci. Jedná se o problém, kdy je třeba ze sady názvů vybrat ten, který se nejvíce odlišuje. Ten nejvíce odlišující se název je, pokud vybráno správně, název jiného produktu. Cílem je získat EbayId

tohoto jiného produktu. Ze sady 13 je tedy třeba vybrat název označený číslem 1 a vrátit jeho EbayId, které je známo ze seznamu vystavení podobných produktů.

```
0: 73.83: grand theft auto v~gta 5 pc full access change
1:  3.65: the sims 3 into future expansion pack pc origin
2: 71.26: grand theft auto 5 pc steam account online
3: 72.59: grand theft auto v~gta new pc rockstar digital
4: 73.94: grand theft auto v~gta 5 pc online full access
5: 73.81: grand theft auto v~gta 5 pc social club account
6: 73.56: grand theft auto v~gta online pc social club
7: 72.70: grand theft auto v~gta 5 online playstation 4
```

Obrázek 13: Názvy ohodnocené použitou metodou k určení nejvíce se odlišujícího

7.1.1 Vzdálenost řetězců

Nejprve bylo zkoumáno použití některé z metod na určení vzdálenosti dvou textových řetězců. Byl by vybrán ten název produktu, který by byl od všech ostatních co nejvzdálenější. Na zjištění, která metoda se nejvíce hodí pro toto konkrétní použití, bylo využito stránky `https://asecuritysite.com/forensics/simstring`. Tato stránka spočítá většinu známých metod vzdáleností pro dva zadané textově řetězce. Po několika experimentech na zmíněné stránce a krátké úvaze se začala objevovat potřeba dávat větší váhu shodám na začátku řetězců. Této potřebě ovšem nebyla schopna dostatečně vyhovět žádná z dostupných metod. Použití metod vzdálenosti řetězců dále znemožňoval fakt, že například název `Samsung Galaxy S7` musí mít dostatečně velkou vzdálenost od názvu `Samsung Galaxy S6`, čehož je s těmito metodami obtížné dosáhnout.

7.1.2 Frekvenční analýza s ohledem na pozici

Po úvodním neúspěchu, kdy byla počítána vzdálenost vždy pouze mezi dvěma řetězci, bylo nahlíženo na názvy jako na sadu. Použitou metodou se stala frekvenční analýza zohledňující pozici slova v řetězci. Pro každé slovo byl spočítán počet jeho výskytů v celé sadě názvů a jeho průměrná pozice v řetězcích, kde se slovo vyskytuje. Výsledkem tohoto je takováto tabulka.

Tabulka 3: Výskyty a pozice slov z sady obrázku 13

Slovo	Počet výskytů	Průměrná pozice
grand	7	1
theft	7	2
auto	7	3
v	6	4
gta	6	5.66
...	...	...
the	1	1
sims	1	2
3	1	3
into	1	4
future	1	5

Každý název pak může být na základě této tabulky ohodnocen. Vybrán je název produktu s nejmenším ohodnocením. Na výpočet ohodnocení je použit následující vzorec

$$\sum_{i=0}^{i=n-1} \frac{f^2}{p * (i+1)}$$

kde `n` je počet slov, `f` je počet výskytů slova a `p` je průměrná pozice slova v názvech. Výsledkem jsou ohodnocení názvu podobná ohodnocením v ukázce 13. Vybráním názvu s minimálním ohodnocením je dosaženo cíle tohoto kroku.

7.1.3 Ukázka kódu

```
function computeScores (names, wordFrequency) {
  let scores = []
  names.forEach((words, index, array) => {
    let score = 0
    for (let i = 0; i < words.length; i++) {
      let word = words[i]
      let nOfOccurrences = (wordFrequency[word] || [0])[0]
      let meanPosition = (wordFrequency[word] || [1, 1])[1]
      score += nOfOccurrences ** 2 / (meanPosition * (i + 1))
    }
    scores[index] = score
  })
  return scores
}
```

Obrázek 14: Metoda pro výpočet ohodnocení názvů

Funkce `computeScores()` dostane sadu názvů, kde každý název je pole slov toho názvu. Argument `wordFrequency` je slovník, kde ke každému slovu v sadě je

pole o délce 2. Na prvním indexu je počet výskytů, na druhém indexu je průměrná pozice slova. Řádek 7 a 8 využívá líného vyhodnocení (viz sekci 5.5) pro specifikování výchozích hodnot. Tyto dva řádky řeší situaci, kdy slovo neexistuje jako klíč ve slovníku `wordFrequency` (toto není nedostatek kódu, viz sekci 7.2). Pokud tedy `wordFrequency[word]` neexistuje, je na pravé straně líného vyhodnocení vytvořeno pole obsahující číslo 0. Přiřazen je vždy prvek na nulté pozici v poli, a to buď ze slovníku `wordFrequency` nebo z nově vytvořeného pole. Toto efektivně a s minimálním větvením použije číslo 0 jako výchozí hodnotu. Řádek 8 využívá stejné logiky, avšak bylo třeba definovat výchozí hodnotu pro pozici na indexu 1. Element na indexu 0 v náhradním poli `[1, 1]` je libovolně zvolen a nepoužit.

7.1.4 Ukázka obtížnější sady názvů

```
00: 413.55: New Samsung Galaxy S7 32GB SM-G930A AT&T TMobile Metro PCs Smartphone
05: 422.54: Samsung Galaxy S7 EDGE Duos SM-G935FD (FACTORY UNLOCKED) Black Gold
06: 297.35: NEW UNLOCKED Samsung Galaxy S7 EDGE SM-G935 32GB WHITE AT&T/T-Mobile
07: 247.71: Unlocked Verizon Samsung Galaxy S7 EDGE SM-G935 BLUE, Smartphone A+
08: 422.01: Samsung Galaxy S7 EDGE Duos SM-G935FD GOLD - BLACK SPOT ON LCD
18: 418.51: New Samsung Galaxy S7 & S7 Edge 32GB AT&T TMobile PCs Smartphone
19: 370.42: USED-Samsung Galaxy S7EDGE 128GB SM-G935FD (FACTORY UNLOCKED)Dual Sim
```

Obrázek 15: Ukázka obtížnějších ohodnocených názvů (nepodstatné názvy skryty)

Na této ukázce názvů produktů je vidět několik problematických názvů (obzvláště číslo 19) spolu s jejich neobvyklými aspekty:

- `USED-Samsung` klade požadavky na zahrnutí pomlčky do oddělovačů.
- `AT&T/T-Mobile` vyžaduje dělení na slova s využitím dopředných lomítek, avšak bude zbytečně rozděleno, pokud znak & bude také oddělovač.
- `UNLOCKED)Dual` opět vyžaduje, aby do oddělovačů patřily i závorky.
- `Unlocked Verizon ...` bude vybráno jako odlišný produkt jen díky svému neobvyklému uspořádání slov. Tento problém se snaží řešit krok 6 (viz sekci 7.2).
- `S7EDGE` nelze vyřešit běžnými způsoby dělení na slova.

Většina těchto problémů jsou zvláštní oddělovače, toto je řešeno inverzní logikou. Oddělovačem není znak v definované množině znaků. Tato množina je zastoupena regulárním výrazem `[A-Za-z0-9]`. Předpokladem je, že názvy budou složeny jen z názvů v angličtině. Názvy se znaky například z češtiny by nebyly rozděleny správně. Problémy s přebytečnými bezvýznamnými slovy jsou z části řešeny seznamem ignorovaných slov. Tento seznam je ovšem pevně deklarován v kódu, což není vhodné dlouhodobé řešení.

```
1  function preprocessNames (names) {
2    let ignoredWords = ['new', 'brand', 'used', 'good', 'super', 'now', 'great']
3    names.forEach((name, index, array) => {
4      let newName = name.trim().toLowerCase()
5      newName = newName.replace(/\s+/g, ' ').split(/[^A-Za-z0-9]/g)
6      newName = Array.from(new Set(newName))
7      for (let i = 0; i < newName.length; i++) {
8        if (ignoredWords.includes(newName[i])) {
9          newName.splice(i, 1)
10       }
11     }
12     array[index] = newName
13   })
14   return names
15 }
```

Obrázek 16: Metoda pro předzpracování názvů

Seznam ignorovaných slov byl v této ukázce zkrácen.

Řádek číslo 4 vytvoří nový název z původního oříznutím mezer (`.trim()`), převedením na malé znaky, nahrazením duplicitních bílých znaků (`.replace(/\s+/g, ' ')`) a následným rozdělením popsaným výše. Na řádku 5 dochází k odstranění duplikátů převedením na `Set`, který vždy obsahuje pouze unikátní hodnoty a následným převedením zpět na pole. Funkce `.splice()` na řádku 8 slouží k odstraňování elementů z pole. První argument je počáteční pozice, druhý argument je počet elementů ke smazání.

7.2 Krok 6 - Porovnání názvů původního a nového produktu

Tento krok byl přidán do algoritmu objevení za účelem zmenšení počtu duplicitně objevených produktů. Určitou chybovost kroku 4, kdy je vybrán jiný produkt špatně nebo do sady názvu není ani portálem Ebay jiný produkt zařazen, je možno tímto krokem do určité míry kompenzovat.

Vstupem do tohoto kroku jsou dvě sady názvů a slovník `wordFrequency` vytvořený z jedné z nich. Cílem je rozhodnout, zda se dle daných názvů jedná o stejné nebo odlišné produkty. Výsledkem může být i neurčitý stav, kdy nelze potvrdit ani stejnost ani odlišnost.

Navržený algoritmus ohodnotí názvy obou sad pomocí jednoho slovníku frekvence slov. Tento slovník je generován z jedné z těchto dvou sad (obvykle z první). Pokud se slovo ve slovníku nenachází, pak je hodnota slova rovna nule. Při tomto ohodnocování je uchovávána informace o původu názvu. Sada první je označena číslem 0, sada druhá je označena číslem 1. Tyto ohodnocené názvy jsou poté sloučeny do jednoho pole. Toto pole je poté seřazeno podle ohodnocení (na směru seřazení nezáleží). Podstatná věc na tomto seřazení je pole původů těchto seřazených názvů. Toto pole původů názvů je pole nul a jedniček. Na tomto poli detekujeme hrany

a počet těchto hran už představuje konkrétní číslo vypovídající o stejnosti nebo odlišnosti názvů. Pokud bude číslo blízké nule, pak se zcela jistě jedná o odlišný produkt (produkty z obou sad budou pohromadě v seřazené sadě). Pokud je počet hran poněkud vysoký (například 15), jsou produkty zcela jistě stejné. Pokud nebylo doteď rozhodnuto, pak nelze rozhodnout.

Z algoritmu vyplývá, že jeho úspěšnost a přísnost hodnocení jsou závislé na dvou pevně zadaných hranicích. Tyto dvě čísla je nutno určit v závislosti na úspěšnosti objevení. Aplikace byla navržena tak, aby těchto parametrů bylo co nejméně.

Varianta tohoto algoritmu je použita i v kroku 9. V kroku 7 je využit slovník `wordFrequency`, který je vedlejším výstupem z tohoto kroku 6, k nalezení nejvhodnějšího názvu. V kroku 7 je využito podobného postupu, jako v kroku 4.

```
0: New Lenovo Thinkpad X1 Carbon 5th Gen, i7-7600U, 16GB RAM, 512GB SSD ...
0: Lenovo ThinkPad X1 Carbon 5th Gen 20HR000DUS 14" FHD Ultrabook, Intel ...
1: Lenovo ThinkPad X1 Carbon 3rd Gen i7 5600U 2.6GHz 8GB 256GB SSD ...
1: Lenovo Thinkpad x1 Carbon 2.6GHz gen 5 Core i7 8GB 250GB SSD Win 10 ...
0: Lenovo ThinkPad X1 Carbon 14in. (256GB, Intel i7 5th Gen., 2.6GHz, ...
0: Lenovo ThinkPad X1 Carbon 14" Laptop PC i7-3667U 256GB SSD 8GB RAM ...
1: Lenovo Thinkpad X1 Carbon 3rd Gen i7-5600U 8Gb 256Gb SSD WQHD Win10 ...
0: Lenovo ThinkPad X1 Carbon laptop Core i7 3.3Ghz WQHD IPS 2560x1440 ...
0: Lenovo X1 Carbon 14" Laptop, Intel Core i7 2.0GHz 8GB RAM 256GB SSD ...
1: Lenovo ThinkPad X1 Carbon laptop Core i7 3.3Ghz Samsung 256GB SSD ...
0: Lenovo ThinkPad X1 Carbon 2nd Gen 14" 2.10GHz i7-4600U 8gb 256gb ...
1: Lenovo Ultrabook i7 X1 Carbon Thinkpad NEEDS LOVE
```

Obrázek 17: Seznam uspořádaných názvů s číslem jejich původní sady

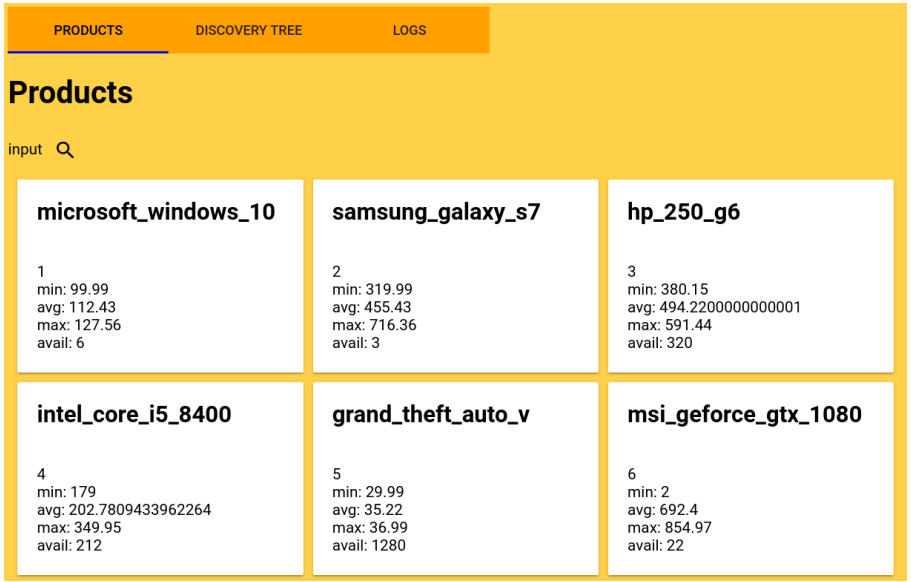
8 Webová aplikace s využitím frameworku Polymer

Tuto část aplikace uvidí případný uživatel. Jedním z dnešních trendů jsou jednostránkové aplikace (SPA), kde prakticky nedochází k opětovnému načtení stránky při změně URL adresy. Toto je u jednostránkových aplikací vyřešeno množstvím JavaScriptu, který se v případě potřeby dotáže HTTP požadavkem serveru, který aplikaci vrátí (obvykle z databáze) požadovaná data.

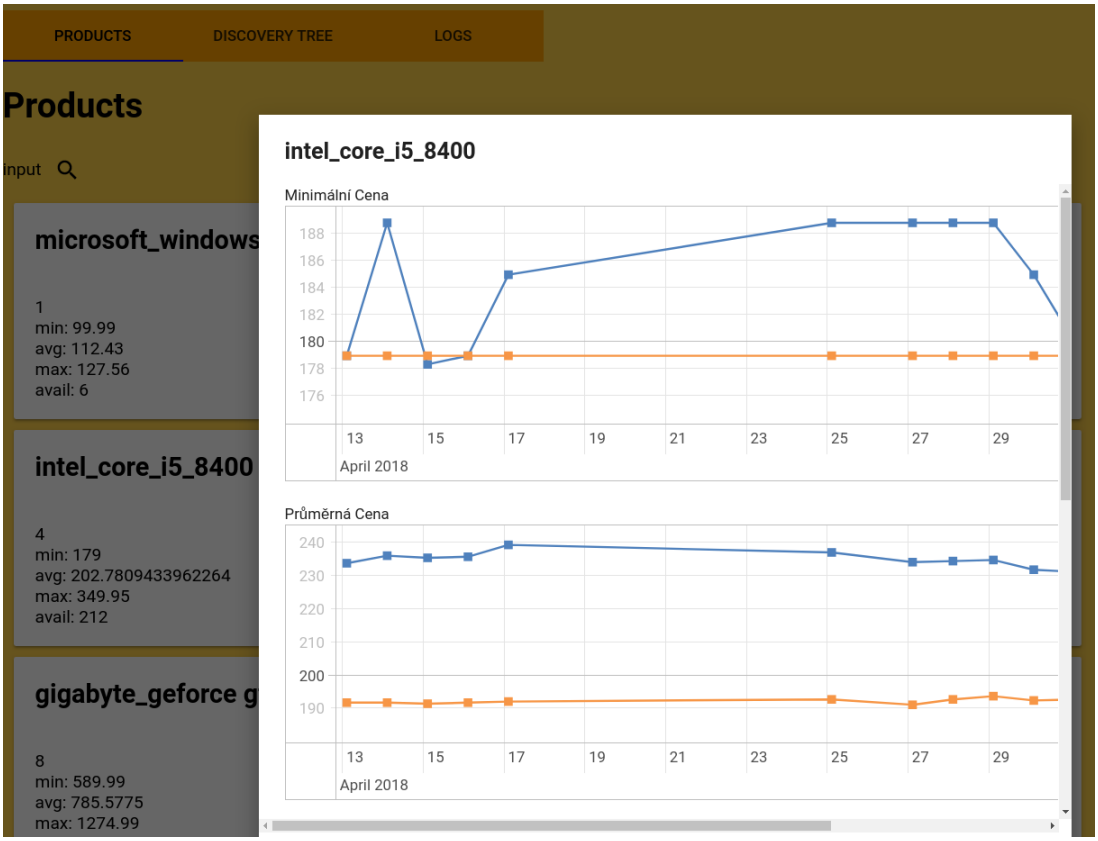
Uživatelské části aplikace nebylo věnováno příliš času a úsilí. Důvodem k tomuto je, že zpracování uživatelské části nepředstavuje mnoho programátorských problémů.

8.1 Přehled evidovaných produktů

Uživateli je prezentován seznam evidovaných produktů. Tyto produkty jsou uspořádány v přehledné mřížce. Uživatelsky přívětivým názvům nebyla věnována pozornost, je to z hlediska funkčnosti aplikace nepodstatná věc. Historii ceny produktu si uživatel může zobrazit kliknutím na dlaždici produktu. Zobrazí se vyskakovací okno s grafy pro každý sledovaný atribut daného produktu (viz obrázek 19). Každá barevně odlišená série hodnot v grafu představuje data z jednoho zdroje.



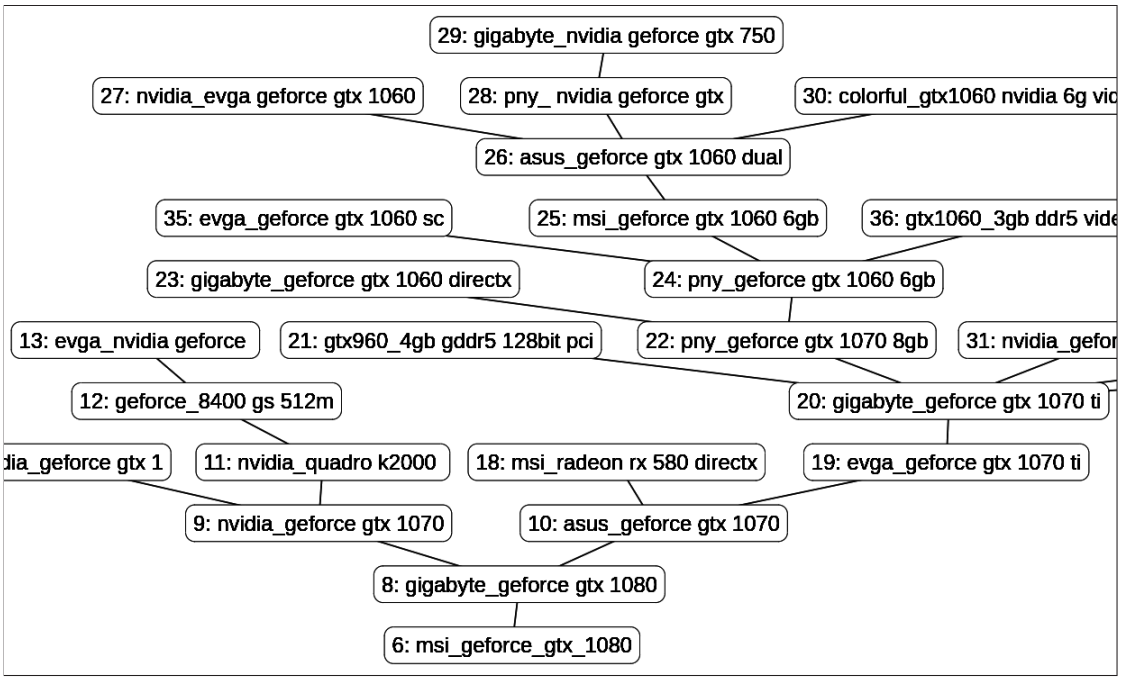
Obrázek 18: Přehled produktů v uživatelské aplikaci



Obrázek 19: Ukázka grafů ceny v uživatelské aplikaci

8.2 Strom objevených produktů

Tento strom není pro uživatele moc zajímavý, je zobrazen hlavně pro účely vizualizace během vývoje aplikace. Na tomto stromu (viz obrázek 20) je názorně vidět, jak objevování produktů postupovalo. V uzlu je vždy napsán identifikátor produktu z databáze a jeho název. Nejprve byl ručně zadán produkt číslo 6, z něj došlo k objevení čísla 8. Z čísla 8 byly objeveny produkty číslo 9 a 10. Tímto způsobem lze postupovat dále.



Obrázek 20: Výřez stromu objevených produktů v uživatelské aplikaci

8.3 Polymer

Pro tyto potřeby lze s výhodami využít některé JavaScript frameworky. Mezi ty neznámější patří Angular a React. Další alternativou je například Polymer. Jedná se o (stejně jako Angular) projekt společnosti Google. Polymer jako projekt vznikl v roce 2015 a od té doby je neustále vyvíjen. Společnost Google tento svůj projekt využila například pro nový vzhled YouTube, Google Earth nebo Google Play Music.

Stejně jako většina ostatních frameworků Polymer umožňuje obousměrnou synchronizaci dat mezi HTML a JavaScriptem, šablony na opakování elementů a samotné vlastní HTML elementy. Definice vlastního elementu je k nahlédnutí k příloze D.

8.4 Nasazení

Při vyvíjení Polymer aplikace je používán příkaz `polymer serve`. Po zavolání tohoto příkazu je na počítači spuštěn webový server, kde je možnost si aplikaci vyzkoušet. Pro připravení na nasazení aplikace slouží příkaz `polymer build`, který aplikaci sestaví. Výstupem z tohoto procesu jsou klasické HTML stránky, které je možno kdekoliv ve statické formě nasadit.

9 Možný Hosting

Při výběru hostingu pro aplikaci jsou dvě základní omezení. První z nich je použití ECMAScript 2017 syntaxe a druhé je použití MySQL databáze.

Použití syntaxe z nového standardu ES2017 mělo negativní dopad při výběru možného hostingu pro Node.js aplikaci a MySQL databázi. Minimální Node.js verze (určena standardem) byla 7.10.1. Toto značně omezilo výběr použitelných hostingů. V případě nutnosti by bylo možné využít kompilátor Babel. Tento kompilátor umožňuje použití syntaxe a objektů z nových standardů JavaScriptu při zachování zpětné kompatibility. Tyto prvky jsou kompilátorem přepsány do kódu, který nevyužívá prvků zvolených nových standardů jazyka (nahrazeny takzvaným polyfillem). Tento kód je však pomalejší ve srovnání s původním.

Dalším omezením byla MySQL databáze, což dále omezilo výběr, protože kombinace Node.js a MySQL databáze není příliš běžná.

9.1 Heroku

Heroku je platforma pro hostování mnoha prostředí pro webové aplikace. Podpora pro Node.js aplikace je velice dobrá, podporovány jsou i ty nejnovější verze (viz [2]). Aplikace na tomto portále je ovládána z příkazové řádky. Nicméně podpora pro MySQL není podle [1] nativní. MySQL podpora je řešena skrze ClearDB službu třetí strany, která je dobře intergrována do systému Heroku.

10 Výsledky

Byla provedena dvě měření. Skriptu byl vždy zadán požadavek na objevení n produktů. Byl sledován počet provedených pokusů o objevení (označen *ppp*). Z toho lze vyjádřit pravděpodobnost objevení nového produktu v tomto měření, jako $p = \frac{n}{ppp}$. Tato pravděpodobnost je ovlivňována samotným portálem Ebay tím, že jiný produkt nemusí do sady podobných produktů zařadit . Dále je pravděpodobnost snížena zahozením produktu v případě kolize UPC kódu.

Měření byla nekonzistentní a bylo jich provedeno příliš málo. Algoritmus vybírající produkt pro objevení nebyl ideální, měl sklony k rozšiřování stromu do hloubky. Toto mělo za následek nacházení mnoha duplicitních produktů a docházelo tak často ke kolizím UPC kódů. Tyto kolize způsobovaly umělé snižování pravděpodobnosti úspěchu.

Měření	n	ppp	Pravděpodobnost objevení (p)
Měření 1	15	127	0,118
Měření 2	15	39	0,385

Tabulka 4: Tabulka výsledků měření pravděpodobnosti objevení produktu

11 Závěr

Bylo dosaženo většiny cílů práce. Kontrola cen byla splněna a historie cen je přehledně prezentována uživateli. Avšak ceny byly sbírány z příliš krátkého období na to, aby se projevily vlivy různých jevů na ceny produktů. Vyhledávání nových produktů (alternativ) bylo také splněno, aplikace je schopna vyřešit všechny problémy bez zásahu uživatele. Pokud by uživatel chtěl zasáhnout do průběhu objevení (specifikovat název produktu), je možno za krátký čas upravit funkci objevení (přeskočit několik volání funkcí) a vytvořit možnost zadání žádané veličiny. Aplikace je dobře připravena na rozšíření svých zdrojů, při návrhu databáze bylo myšleno na případnou potřebu tohoto rozšíření.

Aplikace by mohla být vylepšena hlavně v oblasti objevování, kde je dosaženo poněkud nízké pravděpodobnosti objevení. Tento proces je, mimo jiné, závislý na několika parametrech, které by bylo vhodné s ohledem na úspěšnost upravit. Konkrétně oba počty hran mohou značně ovlivnit celkovou úspěšnost algoritmu. Data o úspěšnosti lze kdykoliv vyčíst ze záznamů o běhu. Úprava algoritmů nebo rovnou jejich nahrazení je také možné. Na určení odlišného produktu stojí za vyzkoušení aplikace neuronové sítě. Trénovací data jsou v určité podobě opět v záznamech o běhu. Dále by mohlo být nějak vyřešeno samočinné slučování duplicitně nalezených produktů, kterých je stále mnoho i přes přísné nastavení hran. Toto slučování by nemuselo fungovat naprosto samočinně, mohlo by například předkládat návrhy uživateli k potvrzení sloučení.

Další vylepšení aplikace s ještě vyšším potenciálem může být AWS, které by mohlo nabízet jiný produkt přímo. Toto by eliminovalo potřebu frekvenční analýzy a detekci hran, čímž by se proces objevení stal velice úspěšný.

Vedlejším produktem práce byl značný osobní přínos. Tento přínos z části pochází ze seznámení se s frameworkem Polymer, avšak konkrétně tento framework příště už znovu nezvolím. Polymer představoval u takto malé aplikace spíše překážku, než usnadnění. Vývoj uživatelské aplikace by byl několikrát rychlejší s použitím klasické kombinace HTML, CSS a JavaScript. Většinu osobního přínosu tvoří zkušenosti s novými standardy jazyka JavaScript a se samotným Node.js. Tyto zkušenosti budou jistě velkým přínosem při hledání práce v tomto oboru.

Literatura

[1] ClearDB MySQL. Heroku Dev Center [online]. [cit. 2018-05-11].
Dostupné z: *https://devcenter.heroku.com/articles/cleardb*

[2] Heroku Node.js Support. Heroku Dev Center [online]. [cit. 2018-05-11].
Dostupné z: *https://devcenter.heroku.com/articles/nodejs-support#node-js-runtimes*

[3] 6 Reasons Why JavaScript's Async/Await Blows Promises Away. [online].
Mostafa Gaafar, 2017-03-25 [cit. 2018-05-11].
Dostupné z: *https://hackernoon.com/6-reasons-why-javascripts-async-await-blows-promises-away-tutorial-c7ec10518dd9*

[4] findItemsAdvanced. API Reference - Finding API [online].
eBay Inc. [cit. 2018-05-12].
Dostupné z: *https://developer.ebay.com/devzone/finding/callref/finditemsadvanced.html*

[5] Debugger. Node.js v10.1.0 Documentation [online].
eBay Inc. [cit. 2018-05-12].
Dostupné z: *https://nodejs.org/api/debugger.html*

[6] findItemsByProduct (UPC) fails. eBay Developer Forums [online].
eBay Inc. [cit. 2018-05-13].
Dostupné z: *https://forums.developer.ebay.com/questions/11215/finditemsbyproduct-upc-fails.html*

A Funkce pro objevení nového produktu discoverNewProduct

```
async function discoverNewProduct (database, parentId) {
  parentId = parentId || await database.getBestProductForDiscovery()
  await database.discoveryStarted(parentId)
  let productCodes = await database.getProductCodes(parentId)
  shuffle(productCodes)
  let ebayId
  for (let code of productCodes) {
    try {
      ebayId = await getBestEbayId(code.UPC)
    } catch (e) {
      continue
    }
    if (ebayId) break
  }
  if (!ebayId) throw {c: 1, o: {parentId}}
  let [names, ebayIds] = await getSimilarProducts(ebayId)
  let theDifferent = pickTheDifferentProduct(names)
  let newProduct = theDifferent.pick
  let wordFrequency = theDifferent.wordFrequency
  let [newNames, newEbayIds] = await getSimilarProducts(ebayIds[newProduct])
  if (isTheSameProduct(names, newNames, wordFrequency) === 1) {
    throw {c: 2, o: {parentId, names, newProduct}}
  }
  let [newName, bestListingName] = pickGeneralProductNames(newNames)
  newName = bestListingName.slice(0, 5).join(' ')
  let newUPCs
  try {
    newUPCs = await getUPCsFromName(newName)
  } catch (e) {
    if (e.message === 0) throw {c: 3, o: {}}
    if (e.message === 1) throw {c: 4, o: {parentId, newNames, newName}}
    if (e.message === 2 || e.message === 3)
        throw {c: 8, o: {parentId, newNames, newName}}
    throw e
  }

  let newWordFrequency = evaluateNames(newNames, false)
  let filteredUPCs = []
  for (let [UPC, name] of newUPCs) {
    let UPCEbayId
    try {
      UPCEbayId = await getBestEbayId(UPC)
    } catch (e) {
      console.log(e.message)
      continue
    }
    let [similarNames, similarEbayIds] = await getSimilarProducts(UPCEbayId)

```

```
49      let isSimilar = isTheSameProduct(similarNames,
50          newNames, newWordFrequency.wordFrequency, true, false) === 1
51          if (!isSimilar) {
52            continue
53          }
54          filteredUPCs.push(UPC)
55        }
56        if (!filteredUPCs.length) throw {c: 5, o: {parentId, newUPCs}}
57        if (await database.isAnyUPCInDatabase(filteredUPCs)) throw {c: 6, o: {}}
58        let result = await database.saveDiscoveredProduct(parentId,
59        newName, filteredUPCs, database)
60        if (result === false) throw {c: 7, o: {parentId, newName, filteredUPCs}}
61        return {c: 0, o: {parentId, names, newProduct, newProductId: result}}
62      }
```

B Odpověď findItemsAdvanced z Ebay API


```
{ "findItemsAdvancedResponse": [ {
"ack": [1 item],
"version": [1 item],
"timestamp": [1 item],
"searchResult": [ {
    "@count": "100",
    "item": [
      {
        "itemId": [ "162530276160" ],
        "title": [ "New Samsung Galaxy S7 SM-G930A - 32GB AT&T Unlocked ..." ],
        "globalId": [1 item],
        "subtitle": [1 item],
        "primaryCategory": [1 item],
        "galleryURL": [1 item],
        "viewItemURL": [1 item],
        "paymentMethod": [1 item],
        "autoPay": [1 item],
        "postalCode": [1 item],
        "location": [1 item],
        "country": [1 item],
        "shippingInfo": [1 item],
        "sellingStatus": [
          {
            "currentPrice": [ { "@currencyId": "USD", "__value__": "284.99" } ],
            "convertedCurrentPrice": [1 item],
            "sellingState": [1 item],
            "timeLeft": [1 item]
          }
        ],
        "listingInfo": [1 item],
        "returnsAccepted": [1 item],
        "condition": [1 item],
        "isMultiVariationListing": [1 item],
        "topRatedListing": [1 item]
    },
    {19 items},
    ...
    {17 items}
    ]
  }
],
"paginationOutput": [1 item],
"itemSearchURL": [1 item]
} ] }
```

C Příklad odpovědi z upcitemdb.com


```
{
  "code": "OK",
  "total": 861,
  "offset": 6,
  "items": [
    {
    "ean": "0737989980220",
    "title": "Samsung Galaxy S6 Edge G925 64GB Factory Unlocked ...",
    "description": "Find cell phones at Target.com! Samsung ...",
    "upc": "737989980220",
    "brand": "Samsung",
    "model": "G925 64GB GOLD",
    "color": "",
    "size": "",
    "dimension": "",
    "weight": "",
    "currency": "",
    "lowest_recorded_price": 155.59,
    "highest_recorded_price": 645,
    "images": [7 items],
    "offers": [7 items],
    "elid": "253555721745"
    },
    {17 items},
    {15 items},
    {14 items},
    {15 items}
  ]
}
```

D Definice vlastního elementu ve frameworku Polymer

```
<link rel="import" href="../../bower_components/polymer/polymer-element.html">
<link rel="import" href="../../bower_components/paper-card/paper-card.html">

<dom-module id="product-el">
  <template>
    <style>
      paper-card {
        width: 100%;
        height: 200px;
        background-color: white;
        padding-left: 20px;
      }
    </style>
    <paper-card class="product">
      <h2>[[shortName]]</h2><br />
      [[idProduct]]<br />
      min: [[minPrice]]<br />
      avg: [[avgPrice]]<br />
      max: [[maxPrice]]<br />
      avail: [[availability]]<br />
    </paper-card>
  </template>
  <script charset="utf-8">
    class Product extends Polymer.Element {
      static get is() {
        return 'product-el'
      }
      static get properties() {
        return {
          idProduct: Number,
          shortName: String,
          fullName: String,
          minPrice: Number,
          avgPrice: Number,
          maxPrice: Number,
          availability: Number
        }
      }
      constructor() {
        super()
      }
    }
    customElements.define(Product.is, Product)
  </script>
</dom-module>
```