

Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra informačních technologií

Studijní program: Aplikovaná informatika

Obor: Informační systémy a technologie

Vyhledávání jako služba

DIPLOMOVÁ PRÁCE

Student: Bc. Luděk Veselý

Vedoucí: Prof. Ing. Zdeněk Molnár, CSc.

2017

University of Economics in Prague
Faculty of Informatics and Statistics
Department of Information Technologies

Study programme: Applied Informatics
Branch: Information Technologies

Search as a Service

DIPLOMA THESIS

Student: Bc. Luděk Veselý
Supervisor: Prof. Ing. Zdeněk Molnár, CSc.

2017

Prohlášení

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze které jsem čerpal.

Datum:

Podpis:

Abstrakt

Tato diplomová práce popisuje návrh a tvorbu fulltextového vyhledávání poskytovaného jako služba.

Klíčová slova

Fulltext, Elasticsearch

Abstract

This diploma thesis describes creation of fulltext search service.

Key words

Fulltext, Elasticsearch

Poděkování

Rád bych poděkoval Ing. Ivanovi Jelínkovi za rady a pomoc při řešení.

Obsah

1	Úvod	9
1.1	Cíle práce	9
1.2	Cílová skupina	10
1.3	Použité metody	10
1.4	Struktura práce	10
2	Komentovaná rešerše použitých informačních zdrojů	12
3	Analýza byznys požadavků na aplikaci	13
3.1	Vysvětlení základních pojmů	13
3.2	Vysvětlení problému, který je řešen	14
3.3	Specifika elektronického obchodování	16
3.4	Definice požadavků na vyhledávání	16
3.5	Technické požadavky na aplikaci	17
3.6	Popis oborů, kterých se práce dotýká	18
3.7	Stávající možnosti implementace vyhledávání	18
3.7.1	Využití relační databáze	18
3.7.2	Elasticsearch, Apache Solr, Sphinx Search	18
3.7.3	Algolia	19
3.7.4	Swifttype Site Search	19
3.7.5	AWS CloudSearch	19
3.7.6	Google Custom Search Engine	19
3.8	Odůvodnění vytváření nového nástroje	20
4	Plnotextové vyhledávání	21
4.1	Analýza dat, v kterých bude vyhledáváno	22
4.2	Automatické indexování textů	23
4.3	Specifika českého jazyka	23
4.4	Problematika tvarosloví	24
4.4.1	Základní pojmy tvarosloví	24
4.4.2	Operátor pravostranného rozšíření	25
4.4.3	Derivátor slovních tvarů	26
4.4.4	Stematizace	26
4.4.5	Lematizace	26
4.5	Problematika významnosti	27

4.5.1	Negativní slovník	27
4.5.2	Významnost výrazů	28
4.6	Tezaurus	28
4.7	Přibližné vyhledávání	29
4.7.1	Editační vzdálenost	29
4.7.2	N-gramová podobnost	30
4.8	Relevance	31
5	Návrh řešení	33
5.1	Definice případů užití	33
5.1.1	Registrace do aplikace	34
5.1.2	Napojení XML souboru s produkty	34
5.1.3	Napojení vyhledávání na elektronický obchod	34
5.1.4	Vyhledávání produktů	35
5.2	Návrh architektury aplikace	35
5.3	Datový model	36
5.4	Návrh API	37
5.4.1	Datové struktury	38
5.4.2	Koncové body	40
6	Implementace	41
6.1	Výběr nástroje pro vyhledávání	41
6.1.1	PostgreSQL	41
6.1.2	Elasticsearch	42
6.1.3	Apache Solr	42
6.1.4	Sphinx	42
6.1.5	Porovnání nástrojů pro vyhledávání	42
6.2	Nastavení a nasazení Elasticsearch	43
6.2.1	Vytvoření Docker obrazu	44
6.2.2	Nastavení indexace v Elasticsearch	46
6.2.3	Vyhledávání v Elasticsearch	50
6.3	Výběr nástroje pro backend a frontend	53
6.4	Backend	53
6.5	Frontend	56
6.6	Deployment	57
7	Testování, ověření	60
8	Závěr	61
8.1	Dosažení vytčených dílů	61
8.2	Diskuze možného budoucího rozšiřování	61
A	Obsah přiloženého CD	62
B	Terminologický slovník	63

1 Úvod

V této diplomové práci se zabývám návrhem a implementací nástroje, který umožňuje provozovatelům elektronických obchodů snadnou a rychlou implementaci plnotextového vyhledávání. Při jeho implementaci je totiž třeba mít určitou úroveň znalosti principů samotného vyhledávání včetně souvisejících oborů a technologií. Implementace kvalitního vyhledávání tak může být pro provozovatele elektronického obchodu náročná jak časově, tak finančně. Tento nástroj je poskytován jako služba se všemi výhodami i omezeními s tímto principem spojenými. Konkrétní cíle diplomové práce popisují v následujících odstavcích.

1.1 Cíle práce

Cílem práce je vytvořit nástroj, který je nabízen jako služba a umožňuje provozovatelům elektronických obchodů snadnou implementaci plnotextového vyhledávání do elektronického obchodu, které K dosažení tohoto cíle je třeba naplnit cíle dílčí.

Prvním dílčím cílem je provedení analýzy problému a to jak z pohledu byznysového, tak z pohledu samotné problematiky vyhledávání. Co se týče pohledu byznysového, tak je třeba analyzovat potřeby zákazníků, zjistit, jaká jsou specifika oblasti elektronického obchodování a definovat kritéria, jejichž naplnění je pro provozovatele elektronických obchodů klíčové. Z pohledu problematiky vyhledávání je pak třeba provést analýzu této disciplíny a utřídit tak znalosti potřebné k poskytnutí kvalitních výsledků plnotextového vyhledávání.

Dalším dílčím cílem je vytvořit návrh řešení problému jednak na základě znalosti potřeb zákazníků a také na základě znalosti problematiky plnotextového vyhledávání. Tento návrh musí být v dalším kroku implementovatelný.

Předposledním dílčím cílem je samotná implementace aplikace dle jejího návrhu. Obnáší to výběr vhodných nástrojů, jejich nasazení do produkčního prostředí, naprogramování jednotlivých služeb a konečně také ověření samotné implementace. Tu je třeba porovnat vůči požadavkům a ověřit tak jejich naplnění.

Posledním dílčím cílem je ověření míry, kterou jsou naplněny požadavky na aplikaci (funkční i nefunkční). Tento krok potvrdí nebo vyvrátí správnost implementace.

1.2 Cílová skupina

První cílovou skupinou je vývojář, řešící problém plnotextového vyhledávání produktů při implementaci elektronického obchodu. Takovému čtenáři by měla práce poskytnout dostatečné teoretické znalosti potřebné pro implementaci vyhledávání. Užitečná také může být konkrétní podoba implementace, která je v této práci popisována.

Další možnou cílovou skupinou je provozovatel elektronického obchodu, který přemýšlí, jakým způsobem zlepšit (případně zavést) plnotextové vyhledávání. V této práci získá přehled o složitosti samotné implementace, poskytne mu komentovaný soupis možných řešení a konečně také poskytne funkční službu, kterou může okamžitě na webový portál napojit.

Poslední cílovou skupinou budiž kdokoli, kdo se zajímá o problematiku plnotextového vyhledávání v českém jazyce. V této práci nalezne soupis problémů souvisejících s češtinou, které je třeba řešit. Dále zde nalezne konkrétní implementaci, kterou se může inspirovat při řešení obdobného problému.

1.3 Použité metody

V této části popisuji metody použité k naplnění jednotlivých cílů. Pro zkoumání problému oboru provedu analýzu trhu, nabízí se také možnost dotazování potenciálních uživatelů. Pro porozumění problematice vyhledávání provedu rešerši literatury. Vzhledem k množství dostupné literatury bude třeba provést syntézu těchto informací. Při vytváření návrhu řešení bude použito modelování, výstupem by tedy měl být model řešení.

1.4 Struktura práce

V první části práce popisuji problematiku vyhledávání v prostředí elektronického obchodování. Definuji zde jednak kontext, v kterém se pohybuji a dále také popisuji problémy, které v tomto prostředí existují. Snažím se identifikovat potenciálního uživatele a definovat jeho požadavky na plnotextové vyhledávání. Toto prostředí má svá specifika, která také popisuji a vysvětluji, proč jsem se na tuto oblast zaměřil. V závěru této části porovnávám existující nástroje umožňující implementaci vyhledávání a zjišťuji tak, proč má smysl vytvářet další službu, jaká je její přidaná hodnota.

V druhé části se zabývám teorií plnotextového vyhledávání. Popisuji zde celý proces od analýzy vstupních dat, přes jejich indexaci, až po samotné vyhledávání. Tyto poznatky budou následně využity k naplnění požadavků na vyhledávání, k zajištění kvalitních výsledků vyhledávání.

V další části práce navrhuji samotnou aplikaci tak, aby vyhovovala požadavkům a zároveň byla následně implementovatelná. Porovnávám zde dostupné nástroje, definuji případný užitek aplikace a vytvářím model výsledné aplikace.

V poslední části diplomové práce popisují konkrétní implementaci v jazyce Go s pomocí úložiště Elasticsearch. Výstupem této části je otestovaný spustitelný program, který umožňuje provádět indexaci produktů a jejich následné vyhledávání.

2 Komentovaná rešerše použitých informačních zdrojů

Při práci na této literatuře vycházím nejvíce z knihy **Počítačové zpracování přirozeného jazyka** [2] od Petra Strossy, přičemž nejužitečnější pro tuto práci je první kapitola zabývající se automatizovaným indexováním textů. Knih zabývající se textovým vyhledáváním existuje celá řada – čerpám však především z knih **Searching in the 21st Century** [3] a **Mining Text Data** [4].

Dále je třeba mít povědomí o oblasti elektronického obchodování, což je přehledně shrnuto v knize **E-commerce** Petra Suchánka [1]. Z této knihy je pro práci nejpřínosnější její první polovina, která pojednává jak o obecných termínech, tak je důkladně vysvětluje.

Při návrhu API je třeba mít povědomí o tom jak jej navrhnout i jak jej v konkrétním programovacím jazyce implementovat. Obecně o návrhu API pojednává kniha **Build APIs You Won't Hate** [7], která je průvodcem návrhu dobře použitelného api od samého počátku, včetně ukázky na konkrétních příkladech. Pro seznámení s jazykem Go na komplexní úrovni je dostatek informací v knize **Go in Action** [8], která je další z řady knih nakladatelství Manning Publications pojednávající o řadě dalších nástrojích. Vzhledem k specifickému užití jazyka jsem čerpám z dalších dvou knih o tomto programovacím jazyce – **Mastering Concurrency in Go** [9] a **Build web applications with Golang** [10]. Co se samotného nasazování aplikace týče, velmi přínosná je kniha **The DevOps 2.0 Toolkit** [12], která by se dala chápat jako průvodce světem DevOps. Přináší spoustu informací pro pochopení tohoto přístupu, což je doloženo řadou příkladů. Speciálně pro důkladnější porozumění nástroji Docker je k dispozici kniha **Docker: Up & Running** [11], která s tímto nástrojem čtenáře seznámí víc než důkladně.

Vzhledem k použití nástroje Elasticsearch je třeba mít dostatek informací také o tomto nástroji. Jedna z nejpodrobnějších knih je **Elasticsearch: The Definitive Guide**, která popisuje veškerou funkčnost tohoto nástroje a popis doplňuje o teoretické informace nutné k pochopení dané problematiky. Elasticsearch se však velmi rychle vyvíjí a tak tato kniha nereflektuje přesně aktuální stav. Konkrétní funkčnost je tedy třeba vždy ověřovat vůči aktuální dokumentaci **Elasticsearch Reference** [6].

3 Analýza byznys požadavků na aplikaci

V první části diplomové práce popisují problematiku plnotextového vyhledávání v prostředí elektronických obchodů. Nejprve je čtenář uveden do problematiky elektronického obchodování a obeznámen s základními pojmy a principy. Následně je popsán problém, který je řešen a jsou definovány konkrétní podmínky, za kterých bude výsledný nástroj přínosný a použitelný. Jsou zde také diskutovány stávající možnosti implementace plnotextového vyhledávání a porovnány alternativní služby.

3.1 Vysvětlení základních pojmů

Pro usnadnění orientace v této práci nejprve vysvětlují dále používané pojmy. Jedná se vesměs o známé termíny, přesto považují za důležité uvést jejich význam na pravou míru.

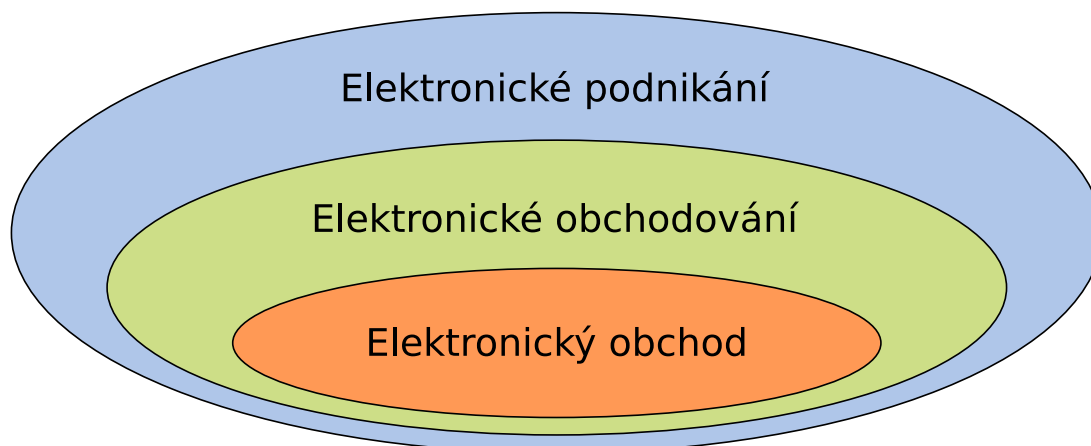
Elektronické obchodování

Termín elektronické obchodování (e-commerce) označuje veškerou činnost spojenou s obchodováním realizovaným prostřednictvím sítě internet [1, strana 11]. Spadá sem distribuce, nákup, prodej, marketing, servis produktů. Elektronické obchodování se dále podle zaměření na cílové skupiny, z nichž nejpodstatnější jsou B2C (zaměřeno na koncové zákazníky) a B2B (zaměřeno na obchodníky) [1, strana 17]. Zároveň je však elektronické obchodování podmnožinou elektronického podnikání (e-business), které označuje veškeré obchodní a výrobní aktivity, zatímco elektronické obchodování se týká samotného prodeje zboží a služeb.

Elektronický obchod

Elektronický či internetový obchod (e-shop) je webová aplikace, jejímž prostřednictvím provozovatel obchodu prodává zboží nebo služby zákazníkům [1, strana 16]. Jedná se o podmožinu elektronického obchodování, jde tedy o jeden z možných prodejních kanálů v prostředí internetu. Elektronický obchod je specifickou webovou aplikací, v níž se opakují určité vzorce. Konkrétně je to katalog produktů, kategorizace zboží, vkládání položek do košíku, platba objednávky, možnost kontaktu s zákaznickou podporou prostřednictvím on-line chatu přímo v aplikaci.

V elektronickém obchodě můžeme sledovat řadu podobností s obchodem kamenným, který zákazníci navštěvují osobně. Některé vlastnosti elektronického obchodu jsou zřejmě inspirovány obchody kamennými (procházení zboží podle kategorií, přidávání položek do košíku, uplatnění slevových poukázek při platbě). Často však elektronický obchod využívá možností, které webové prostředí nabízí (filtrace a řazení položek podle parametrů, vrácení se k naplněnému košíku, porovnávání zboží napříč obchody).



Obrázek 3.1: Vztah elektronického podnikání, obchodování a podnikání (Zdroj: [2])

Zcela zásadní výhodou elektronického obchodu oproti návštěvě obchodu kamenného je možnost vytvoření objednávky z pohodlí domova a její následné doručení zásilkovou službou. Tento přístup především šetří čas strávený cestou do obchodu a čas strávený v obchodě. Například v segmentu nákupu potravin může být výhodou to, že je takový nákup hygieničtější a zboží bude doručeno v lepším stavu díky uzpůsobeným vozům dopravců. Výhody ale vznikají i pro provozovatele – nemusí zřizovat prodejnu včetně jejího vybavení a zaměstnance, mohou efektivněji navrhnout sklad, nebo dokonce využít automatizace při kompletaci objednaného zboží.

Plnotextové vyhledávání

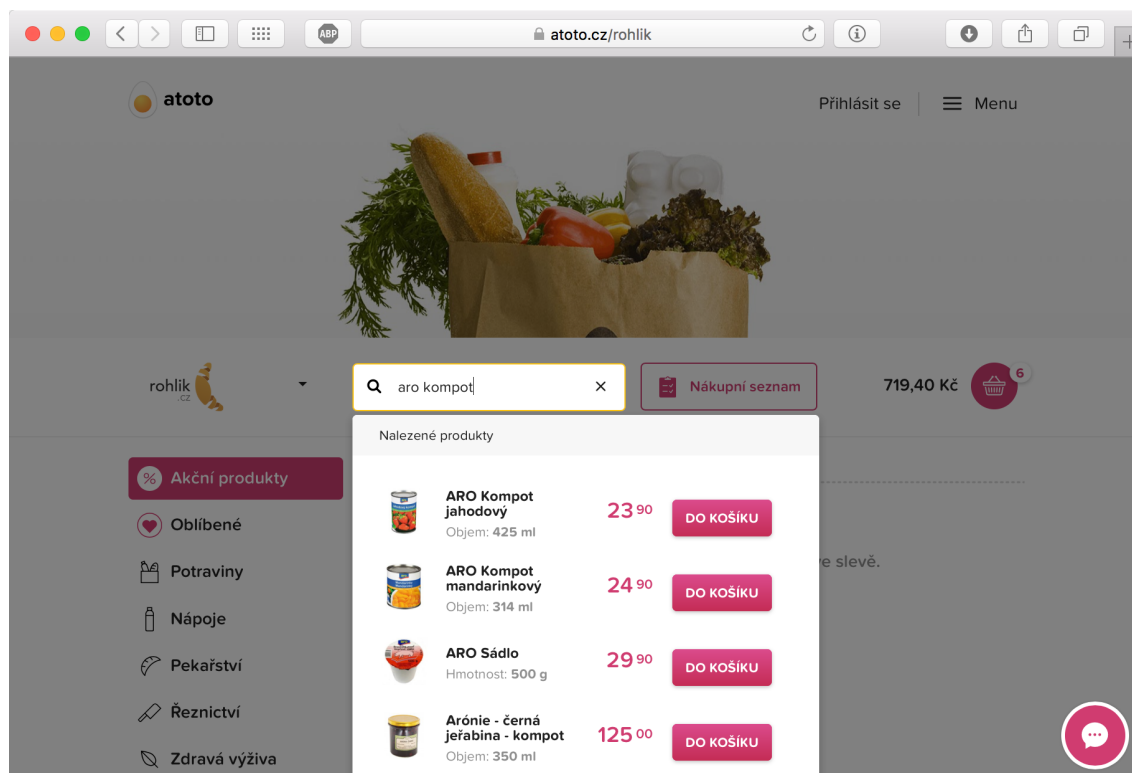
Plnotextové vyhledávání (full-text search), je způsob vyhledávání v textových datech, kdy se vyhledává uživatelem formulovaný dotaz v invertovaném souboru (někdy také indexovém souboru nebo invertovaném indexu), tedy v souboru obsahující výrazy, podle nichž je daný záznam vyhledatelný [2, strana 15]. Proces, kterým vzniká indexový soubor se nazývá indexace. Tato problematika je podrobněji popisována v druhé kapitole.

3.2 Vysvětlení problému, který je řešen

Plnotextové vyhledávání je rychlým prostředkem nalezení konkrétního produktu v elektronickém obchodě. Typicky je využito zákazníky, kteří jsou na webu s konkrétní

potřebou a jsou schopni formulovat výraz, podle kterého produkt hledají. Může jít o název typu produktu, jeho značku, variantu nebo kategorii. Pro takového uživatele je procházení webu procházením kategorií zdlouhavé a právě plnotextové vyhledávání mu může zásadně zrychlit cestu k nalezení hledaného produktu.

Důležitost plnotextového vyhledávání navíc roste s rostoucím počtem produktů a kategorií, v kterých je složité se orientovat. Uživatel tak může vyhledávání využít už pro nalezení kategorie, která se nachází v dlouhém a nepřehledném menu.



Obrázek 3.2: Ukázka plnotextového vyhledávání v elektronickém obchodu atoto.cz

Implementace plnotextového vyhledávání však není triviální záležitostí. Vstup uživatele je v přirozeném jazyce, takže je třeba jej odpovídajícím způsobem zpracovat, aby bylo vůbec vyhledávání na webu použitelné. Studium této problematiky může být zdlouhavé a v konečném důsledku drahé. Je třeba brát v potaz i případnou složitost dalších nástrojů, které bude třeba pro funkční vyhledávání spravovat, to s sebou nese více prostoru k potenciálním chybám.

Pokud bude k dispozici hotové řešení, které bude snadno nasaditelné na elektronický obchod a bude umožňovat jednoduché uzpůsobení potřebám konkrétního webu, bude to výrazná úspora jak času, tak peněz provozovatele obchodu.

Veškeré výše jmenované problémy mohou být značnou bariérou pro implementaci kvalitního vyhledávání, přes kterou se tak množství zejména menších obchodů nepřenese. Problémy lze shrnout do následujících bodů:

- Provozovatel elektronického obchodu chce nasadit textové vyhledávání na web, to se musí naprogramovat

- Aktuální podoba textového vyhledávání na webu je nekvalitní
- Vyhledávání na webu je pomalé, s přibývajícimi produkty stále hůře použitelné

Řešení vyhledávání svépomocí je navíc z teoretického hlediska obtížné z těchto důvodů:

- Je nutná znalost principů přirozeného jazyka
- Je nutná znalost nástrojů, které umožňují pokročilou práci s jazykem
- Je nutná znalost technologií, které poskytnou vyhledávání dostatečně rychle

3.3 Specifika elektronického obchodování

V prostředí elektronického obchodování existují jisté vzorce které se opakují a odlišují toto prostředí od ostatních. Vyhledávají se zpravidla produkty (případně služby), které jsou zařazeny do určitých kategorií a disponují atributy jako cena, název, kód, popis, url, obrázek, dostupnost a dalšími parametry, které bývají číselné (hmotnost, objem) nebo výčtové (barva, značka).

Z toho vyplývá co a podle čeho se bude vyhledávat. Produkty mívají jednoznačné identifikátory, které jsou navíc standardizovány (EAN, ISBN). Pomocí těchto kódů je možné vyhledat produkty napříč elektronickými obchody nebo je párovat v cenových srovnávacích. Pro vyhledávání je dále důležitý název produktu, případně jeho varianty, která upřesňuje konkrétní verzi produktu. Méně často je nutné vyhledávat v popisu produktu, kde bývá uveden jak slovní popis, tak výpis parametrů.

U produktů dále bývá evidována cena, která může být interně uložena v kombinaci s marží. Obchody operující se zlevněným zbožím pak mívají uvedenou cenu jak před slevou, tak po této slevě, aby zákazník viděl, kolikaprocentní sleva je na produktu. I samotná cena produktu (případně marže prodejce) může hrát svou roli ve vyhledávání.

3.4 Definice požadavků na vyhledávání

Požadavky na samotné vyhledávání jsou ovlivněny prostředím, ve kterém vyhledávání probíhá a uživateli, kteří vyhledávání využívají. Primárně by mělo být možné nalézt daný produkt podle jeho názvu, nebo části názvu. To samé platí pro kód produktu. Produkt by měl být ale vyhledatelný i podle dalších atributů, jako je název kategorie nebo název značky výrobce. Někdy dokonce uživatel nemusí vyhledávat konkrétní produkt, ale jen produkty dané značky nebo kategorie, což by měl systém také umět rozpoznat.

Podoba vyhledávaných výrazů bude zadávána zákazníky v přirozeném jazyce, s čímž by si mělo vyhledávání také poradit. Konkrétně jde o tvarosloví, kdy může uživatel zadat výraz například v jiném pádu, než je uvedeno v názvu produktu. Dále jde o vyrovnání se s chybami, které mohou vzniknout při formulaci vyhledávaného

výrazu – překlady nebo pravopisné chyby. V neposlední řadě jde také o vztah slova k jeho významu, kdy může jedno slovo mít více významů (homonymum) nebo naopak více slov může odpovídat jednomu výrazu (synonymum).

Další problematikou při zadávání hledaného výrazu uživatelem, je poskytování relevantních výsledků už ve chvíli, kdy uživatel formuluje dotaz, tedy jej teprve píše. V takovém případě je třeba odhadnout, který výraz chce napsat a tuto informaci využít při zobrazení odpovídajících výsledků.

Ve chvíli, kdy systém zná produkty, které odpovídají hledanému výrazu by měl výsledky poskytovat ve vhodném pořadí, měl by tedy pracovat s relevancí výsledků vzhledem k zadanému výrazu. Toto může být velmi obtížné, protože každý uživatel má zájem o jiné produkty a tak pro něj mohou být ro jeden výraz relevantní jiné produkty, než pro někoho jiného. Dále do tohoto pořadí můžou vstupovat požadavky provozovatele elektronického obchodu v případě, kdy má specifické požadavky na zboží, které chce nabízet přednostně. Důvodů pro takové chování může být více, může souviset s marží konkrétních produktů, nebo s filozofií samotného obchodu.

3.5 Technické požadavky na aplikaci

Na aplikaci je kladeno několik technických požadavků. Nejedná se o funkční požadavky, nýbrž o požadavky související s použitelností, výkonností, spolehlivostí a podporou. Naplnění těchto požadavků zajistí následný bezproblémový chod aplikace a uspokojení potřeb zákazníka.

Prvním technickým požadavkem na aplikaci je její rychlost. Ta je důležitá pro spokojenost koncového zákazníka, který provádí vyhledávání. Všechny části aplikace nejsou z hlediska rychlosti tak kritické, jedná se primárně o samotné vyhledávání. Akceptovatelná rychlost uživatelského rozhraní v tomto případě je 100 ms [19], v ideálním případě do 50 ms [20]. Pro naplnění tohoto požadavku je třeba, aby byla rychlost odezvy API co nejnižší. Rychlost by dále nemělo výrazně ovlivňovat množství současných požadavků. Aplikace by měla být navržena tak, aby ji bylo možné snadno škálovat při zvyšující se zátěži.

Dalším požadavkem je formát samotného API. Bude třeba použít takový formát, který bude co nejsnadněji implementovaný v používaných webových technologiích pokud možno bez instalace jakýchkoli rozšíření. To je důležité pro rychlé nasazení aplikace do provozu a odbourání možných vstupních bariér při rozhodování o využití nástroje. Pro načtení produktů z elektronických obchodů by bylo ideální využít stávajících exportů pro jiné systémy, kterými by obchody mohly implementovat. Pokud by nic takového neexistovalo, musí být vytvoření export produktů pro provozovatele elektronického obchodu co nejsnazší.

Posledním technickým požadavkem je dokumentace samotného API. Dokumentace musí být dobře pochopitelná, tedy psaná přehledně, ideálně s konkrétními ukázkami použití. Výhodou je také možnost vyzkoušet si práci s API vůči testovacímu prostředí, kde nebude riziko vzniku chyb na produkčních datech.

3.6 Popis oborů, kterých se práce dotýká

Samotné řešení problematiky vyhledávání je výrazně interdisciplinární obor. Je nutné mít značné povědomí o získávání informací, počítačovém zpracování přirozeného jazyka, což souvisí jak s počítačovou lingvistikou, tak úzce s teorií formálních jazyků a překladačů. Existuje také vztah mezi lingvistikou a logikou, jakožto vědě o myšlení, která se odehrává v kategoriích lidského jazyka. Podobný vztah lze nalézt také s umělou inteligencí, vzhledem k tomu, že používání přirozeného jazyka je inteligentní činností. Díky tomu, že probíhá ukládání dat, je důležitá znalost teorie datových modelů. Vzhledem k množství ukládaných dat se také můžeme dotýkat oboru Big Data. V neposlední řadě je třeba porozumět potřebám potenciálních uživatelů, tedy mít určitou znalost podnikání a obchodování.

3.7 Stávající možnosti implementace vyhledávání

Na trhu existuje několik služeb, které umožňují implementaci plnotextového vyhledávání. Ty se však liší funkčností, obtížností implementace i cenou. Níže popisují nejvýznamější z nich zejména vzhledem k požadavkům na aplikaci.

3.7.1 Využití relační databáze

První možností, jak vyhledávání implementovat je využití stávající databáze, kterou elektronický obchod disponuje. Ať už se jedná o open source (MySQL, Postgres) nebo komerční (SQL Server, Oracle) databázi, možnosti plnotextového vyhledávání jsou zde omezené. Výhodou je to, že jsou data stále v jednom úložišti, není třeba řešit správu (instalaci, konfiguraci nebo zálohování) dalšího nástroje. Ani vývojáři se nemusí učit nic nového, pouze využijí stávající databázi. Vyhledávání zde probíhá v nejjednodušší formě pomocí operátoru LIKE. Některé databázové systémy mají disponují pokročilejšími funkcemi, kterými lze vyhledávání zlepšit [21].

3.7.2 Elasticsearch, Apache Solr, Sphinx Search

Využitím nástrojů přímo určených pro implementaci plnotextového vyhledávání lze dosáhnout nejlepších výsledků, ať už se jedná o Elasticsearch či Apache Solr využívající Apache Lucene [22], nebo Sphinx. Jde o typický krok v okamžiku, kdy přestává funkčnost relační databáze pro potřeby vyhledávání dostačovat. Tyto nástroje umožňují pokročilé nastavení indexace a zároveň jsou dostatečně rychlé, aby byly schopny provádět vyhledávání v řádu milisekund. Zřejmou nevýhodou tohoto řešení je nutnost znalosti dalšího nástroje, jeho správa a řešení synchronizace se stávající databází.

3.7.3 Algolia

Algolia je pokročilý nástroj umožňující implementaci plnotextového vyhledávání [15]. Zaměřuje se na poskytování kvalitních výsledků v co nejkratším možném čase (v řádu milisekund). Snaží se usnadnit práci programátorům tím, že nabízí řadu připravených integrací pro konkrétní programovací jazyky a frameworky. Kromě výsledků vyhledávání umožňuje faceting, tedy dodání dat pro tvorbu filtrů, podporuje řadu jazyků nebo vyhledávání podle geografické lokality. V neposlední řadě je k dispozici dashboard zobrazující stav systému a statistiky vyhledávání.

Pokud si zákazník vystačí s 10 000 vyhledatelnými produkty, může nástroj Algolia používat zdarma, pouze je povinnen uvést ve výsledcích vyhledávání logo firmy. Placené verze začínají na 59 dolarech za měsíc za 100 000 produktů.

3.7.4 Swiftype Site Search

Swiftype Site Search je nástroj podobný nástroji Algolia, zaměřuje se však více na uživatelské rohraní a celkově na snadnost použití [16]. Mezi jeho hlavní funkce patří vyhledávání v okamžiku formulace hledaného výrazu, možnost filtrace výsledků, dodatečná úprava pořadí výsledků ručním zásahem na úrovni konkrétních výsledků, nebo na základě některého z atributů produktu. V neposlední řadě je také k dispozici přehledná analytika proběhlého vyhledávání. Cena za provoz služby začíná na 299 USD měsíčně.

3.7.5 AWS CloudSearch

Firma Amazon poskytuje v rámci svého cloudu službu CloudSearch [17]. Jejímí přednostmi jsou především vysoká výkonnost a škálovatelnost. Velkou vstupní bariérou je však její počáteční složitost. Uživatel musí nejprve proniknout do některých základních konceptů Amazon Web Services, až poté může začít pracovat se samotným vyhledáváním. Co se funkčnosti týče, nabízí služba podporu 34 jazyků, možnost nastavení váhy jednotlivých atributů, nebo doplňování textu během psaní. Cena záleží na využití výpočetní kapacity, její určení je tedy složitější.

3.7.6 Google Custom Search Engine

Google Custom Search Engine [18] se nejvíc odlišuje od ostatních služeb. Vyhledávání je plně řízeno algoritmem společnosti Google a také výsledky vyhledávání vypadají obdobně, jako výsledky vyhledávání na www.google.com. Odlišné je i samotné napojení na službu – výsledky vyhledávání jsou na web vloženy jako samostatná stránka, na které je vidět logo společnosti Google. Základní varianta je však zdarma, je to tedy levná cesta, jak rychle zprovoznit vyhledávání na webu. Pokročilejší varianta umožňující konfiguraci stojí 100 USD ročně.

3.8 Odůvodnění vytváření nového nástroje

Výš uvedené nástroje sice umožňují implementaci textového vyhledávání, lze s nimi dosáhnout dobrých výsledků. Nicméně pokud provozovatel obchodu není ochoten investovat velké částky do pokročilých řešení, nastává problém. Tento problém je ještě výraznější, pokud obchod nedisponuje vlastním IT oddělením a pro samotnou implementaci vyhledávání je vhodné využít minimum času najímaných IT specialistů.

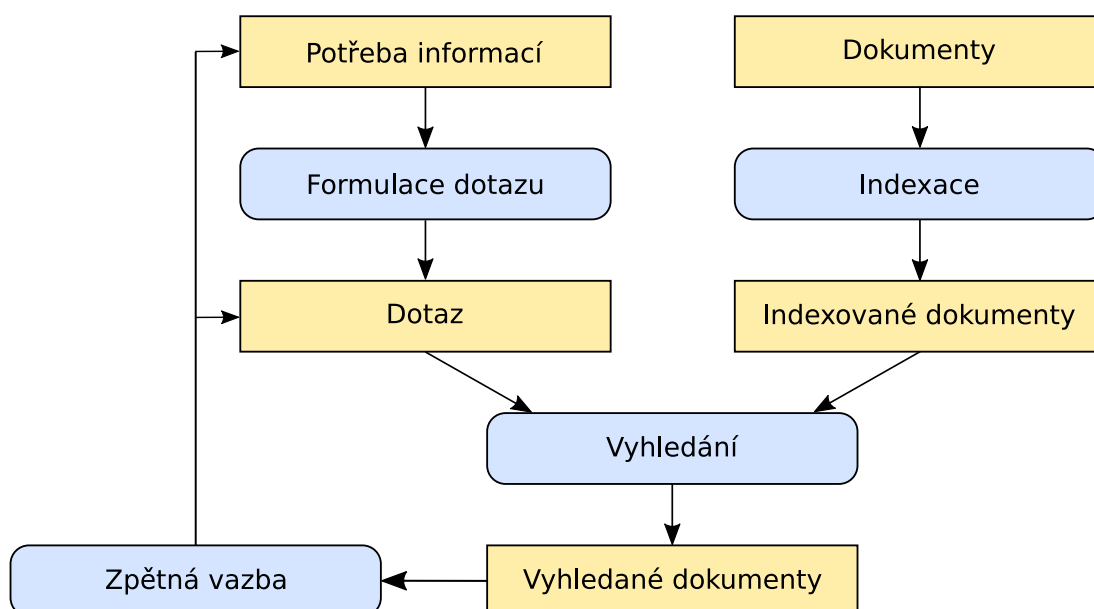
Vzhledem k zjištěným problémům při implementaci textového vyhledávání je očekávaným přínosem této práce, respektive vzniklé aplikace:

- Na webu obchodu bude dříve kvalitnější a rychlejší vyhledávání
- Rychlejší implementace vyhledávání (pokud ještě na webu žádné není)
- Vyhledávání může být outsourcováno (úspora nákladů na IT i provoz)

4 Plnotextové vyhledávání

Tato kapitola popisuje problematiku textového vyhledávání a poukazuje tak na konkrétní problémy, které je při vyhledávání nutno řešit. Problematika je rozebírána v kontextu elektronického obchodování, jsou tedy uváděny pouze principy aplikovatelné v tomto oboru.

Vyhledávání je proces sestávající jednak z ukládání dokumentů určených k vyhledávání a dále z provedení vyhledávání uživatelem. Obecné schéma vyhledávání vypadá následovně:



Obrázek 4.1: Proces vyhledávání (Zdroj: [3])

Ukládání dokumentů probíhá tak, že se upraví pro potřeby vyhledávání a uloží do indexu, přičemž celý tento proces se označuje jako indexace. Ve chvíli, kdy jsou dokumenty připraveny k vyhledávání, může uživatel mající potřebu v těchto dokumentech vyhledávat (získávat z nich informace) formulovat dotaz, kterým chce získat odpovídající dokumenty. Z něj je vytvořen dotaz, podle něž bude vyhledávání provedeno. Následně jsou uživateli dodány dokumenty vyhovující dotazu, ten pak může dotaz na základě získaných dokumentů a pochopení způsobu vyhledávání dotaz upravit.

4.1 Analýza dat, v kterých bude vyhledáváno

Nejprve je třeba prozkoumat data, ve kterých bude vyhledávání prováděno. Vycházím ze skutečnosti, že naprostá většina elektronických obchodů své zboží nabízí také prostřednictvím srovnávačů zboží, přičemž v počtu návštěv a tedy celkovém provozu v rámci českého internetu jsou nejdůležitější Heureka.cz a Zboží.cz [23].

Data pro tyto srovnávače musí být dodávána- prostřednictvím XML feedu, který odpovídá specifikaci jednotlivých systémů. Při porovnání obou specifikací lze nalézt jistou podobnost v požadovaných údajích. Samotný XML feed obsahuje produkty obchodu, přičemž u každého produktu je možné uvést několik atributů, z nichž některé jsou povinné. Konkrétně v případě Zboží.cz musí mít každý produkt uveden název, popis, URL, cenu a dostupnost. Dále lze využít volitelných parametrů jako název kategorie, název značky a výrobce, EAN, ISBN, produktový kód výrobce, případně další doplňkové informace a parametry [25]. V případě Heureka.cz je podstatný název obsahující případně výrobce a produktové číslo či kód, název kategorie, dostupnost a cena dopravy. Informace o produktu mohou dále obsahovat popis, EAN, kód produktu nebo soupis parametrů [24]. Minimální XML soubor s jedním produktem ve formátu Heureka by vypadal následovně:

Výpis 4.1: Ukázka XML souboru ve formátu Heureka.cz

```
<?xml version="1.0" encoding="UTF-8"?>
<SHOP>
  <SHOPITEM>
    <ITEM_ID>40272131</ITEM_ID>
    <PRODUCT>
      Matrace MYRBACKA - Latexová matrace, střední tvrdost, bílá
    </PRODUCT>
    <PRODUCTNAME>Matrace MYRBACKA</PRODUCTNAME>
    <DESCRIPTION>
      <![CDATA[Latex vám umožní lépe odpočívat a to tak, že sleduje
      kontury vašeho těla, uvolňuje tlak a poskytuje podporu.
      - Potah: 64% polyester, 36% bavlna
      - Vnitřní látka: Netkaný polypropylen, 100% jehněčí vlna
      - Komfortní materiál: syntetický latex, pěna polyuretan]]>
    </DESCRIPTION>
    <URL>http://ikea.com/cz/cs/catalog/products/40272</URL>
    <IMGURL>http://ikea.com/cz/cs/images/products/40272.JPG</IMGURL>
    <PRICE>9087</PRICE>
    <PRICE_VAT>10990</PRICE_VAT>
    <CATEGORYTEXT>Dům a nábytek | Nábytek | Matrace</CATEGORYTEXT>
    <MANUFACTURER>IKEA</MANUFACTURER>
  </SHOPITEM>
</SHOP>
```

Na základě znalosti struktury feedů lze zjistit, že při využití stávajících XML exportů internetových obchodů bude každý produkt obsahovat název, popis, URL, cenu, dostupnost a název kategorie, přičemž pravděpodobně bude k dispozici řada dalších atributů, dle použitého formátu feedu a dle pečlivosti provozovatele obchodu.

4.2 Automatické indexování textů

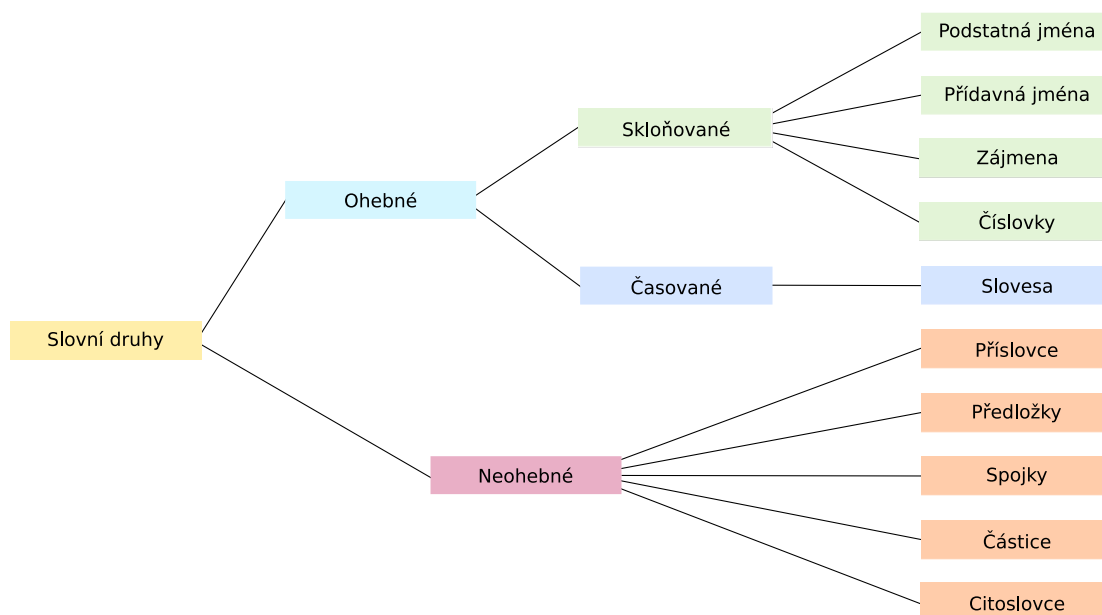
Indexace je proces, kdy jsou ukládány textové dokumenty určené k vyhledání. Ukládány jsou pouze výrazy, podle nichž by měl být ukládaný dokument vyhledatelný, místo kam jsou tyto výrazy ukládány se nazývá index. Jedná se vlastně o klíče, podle kterých bude možné rychle hledaný záznam nalézt. To je důležité z hlediska rychlosti vyhledávání – není možné všechny uložené záznamy procházet a analyzovat až ve chvíli vyhledávání.

Při indexování je třeba analyzovat každý výraz, který by měl být uložen do indexu. V první řadě je třeba rozhodnout, zda je třeba jej do indexu ukládat, je tedy třeba řešit významnost jednotlivých výrazů v textu. Pokud daný výraz není pro vyhledávání užitečný, nemá smysl jej ukládat. Dalším problémem je tvarosloví, tedy to, že slova mohou měnit tvar (jedná se například o skloňování podstatných jmen), stále se přitom jedná o jedno slovo. V neposlední řadě je také třeba zohlednit význam slov v daném oboru, kdy mohou stejná slova vyjadřovat jiné věci (homonymie – například myš ve smyslu počítačového příslušenství a myš jako zvíře), nebo může být jeden význam vyjádřen různými slovy (synonymie – lednička i chladnička označuje totéž). Konečně je také třeba brát v úvahu další vztahy mezi slovy, jako nadřazenost a podřazenost. Například banán a ovoce mohou označovat totéž, přičemž ovoce to činí v širším slova smyslu, toto označení lze použít i pro jiné ovoce, například jablko.

4.3 Specifika českého jazyka

Vzhledem k tomu, že je vyhledávání vytvářeno pro české internetové obchody, bude veškeré vyhledávání probíhat v českém jazyce, a proto nyní popíši jeho specifika. Český jazyk pracuje se slovy odlišně než jiné jazyky. Dochází i k drobným odchylkám i v rámci českého jazyka na základě geografické oblasti (nářečí) nebo na základě zaměření dané skupiny lidí (hantýrka). Dále se můžeme zabývat grafickou podobou jazyka (morfologií) nebo jeho zvukovou podobou (fonetikou). Vzhledem k tomu, že vyhledávání probíhá na textové bázi, zabývám se dále právě grafickou podobou jazyka.

Slova jsou v českém jazyce řazena do slovních druhů, kterých je celkem deset. Slovní druh pomáhá určit chování slova – slova stejného slovního druhu se ohýbají a vyskytují v textu podle podobných pravidel. Slovní druhy lze dělit podle toho, zda jsou ohebné (tedy zda se jejich tvar může měnit) a neohebné. Neohebné slovní druhy jsou zpravidla používány ve větě společně s ohebnými, protože samy o sobě nenesou informační hodnotu. Naopak ohebné slovní druhy samy o sobě nesou informační hodnotu, jsou proto vhodnými kandidáty pro to, aby se podle nich vyhledávalo. Výjimkou jsou zde zájmena, která nesou informaci nepřímo.



Obrázek 4.2: Dělení slovních druhů dle jejich ohebnosti

V rámci ohebnosti lze rozlišovat jakým způsobem jsou ohýbána. Slovesa mohou být časována, příslovce mohou být stupňována, podstatná jména, přídavná jména, zájmena a číslovky mohou být skloňována. Dále v češtině rozlišujeme tři jmenné rody (mužský, ženský a střední) a dvě čísla (jednotné a množné). U sloves navíc rozlišujeme jejich vid.

4.4 Problematika tvarosloví

První problém související s zpracováním přirozeného jazyka je vypořádání se s tvaroslovím (morfologií). Jednotlivá slova se v českých textech objevují v různých slovních tvarech, málokdy jsou všechna slova v základním tvaru. Například jeden produkt může mít název **Alex Čistič na laminát s pomerančovým olejem** i **Alex čistič na laminát pomerančový olej**. V obou případech je uveden **pomeranč**, jen v různém tvaru. Při vyhledávání však chceme docílit toho, aby byla obě slova vyhledatelná shodně, ať už bude slovo pomeranč zadáno v kterémkoli pádě. Typické řešení je vřazení modulu do procesu indexace dokumentů nebo dotazů, který sjednocuje podobu zadaných slov.

4.4.1 Základní pojmy tvarosloví

Nejprve vysvětlím základní pojmy rozebíraného oboru, jejichž znalost je nutná k pochopení následujícího textu.

Morfém

Morfém je minimální funkční jednotka s vlastním významem. Jde o základní jednotku vědy, která se zabývá tvorbou slov, tedy morfologie.

Lexém

Lexém je základní stavební jednotka slovníku, znaková jednotka vyjadřující pojem nesoucí význam. Je to základní jednotka vědy zabývající se slovní zásobou – lexikologie. Ta je stejně jako morfologie odvětvím lingvistiky. Každý lexém je pak možné dále dělit na koncovku, kmen, kořen, prefix a sufix.

Foném

Foném je hláska umožňující rozlišit význam. Obory, pod které spadá, se zabývají zvukovou stránkou jazyka a nazývají se fonetika a fonologie.

Flexe

Flexe neboli ohýbání popisuje změnu slov vzniklou při skloňování, časování nebo ohýbání. Jde o grafickou změnu lexému pro vyjádření odpovídajícího pádu, rodu a dalších gramtických kategorií.

4.4.2 Operátor pravostranného rozšíření

Nejjednodušším řešením problematiky ohýbání slov je operátor pravostranného rozšíření. Mějme slovo **svíčka**, které se při skloňování v jednotném čísle mění na svíčky, svíčky, svíčku, svíčko, svíčky, svíčkou. V tomto příkladu lze vypožorovat, že se mění pouze konec slova, zatímco levá část **svíč-** zůstává stejná. Tento přístup bude sice poměrně úspěšný, nicméně se nevyhneme případům, kdy bude mít více slov stejný základní tvar. Mohli bychom tak při vyhledání výrazu **svíčk** obdržet i dokumenty obsahující slovo **svíčková**.

Vzhledem k jednoduchosti a zároveň poměrně vysoké úspěšnosti bývá tento přístup často využíván jako jedna z prvních implementací plnotextového vyhledávání přímo v relační databázi pomocí operátoru **LIKE**. Konkrétně vyhledávání v jazyce SQL výše diskutovaného výrazu by bylo provedeno jako **LIKE 'pomoranč%'**.

Kromě problému s možným shodným základem pro různá slova je další komplikací při použití této metody skutečnost, že při ohýbání slov nedochází pouze ke změně koncovek, ale často se mění také některá písmena základu slova. Uvažujme slovo **kůň**, které má v druhém pádu podobu **koně**. Pro tento případ by bylo možné rozšířit možnost pravostranného rozšíření o nahrazení pouze jednoho znaku. Pokud bychom tak měli pro nahrazení více písmen využít znak ***** a pro nahrazení právě jednoho písmene znak **?**, byl by základní tvar tohoto slova **k?ň***. To je ale problém, protože takový výraz odpovídá například i slovu **kaňka**.

4.4.3 Derivátor slovních tvarů

Derivátor slovních tvarů je nástroj, který namísto daného slova vygeneruje všechny jeho gramatické tvary, případně jeho použitelné odvozeniny. Ty jsou následně použity jako jejich logická disjunkce. Například místo slova **svíčka** by se použilo:

```
(svíčky OR svíčce OR svíčku OR svíčko OR svíčkou  
OR svíček OR svíčkám OR svíčkami OR svíčkách)
```

Takový nástroj musí obsahovat co nejpřesnější sadu pravidel, jak vytvářet veškeré tvary zadaného slova. Dále je třeba k expandovanému slovu zadat jeho veškeré informace, aby mohlo být vybráno správné pravidlo.

4.4.4 Stematizace

Stematizace je proces, kdy je slovo převáděno na jeho kmen (stem). Toho je docíleno odstraněním předpon, přípon a koncovek. Její použití je vhodné pro jazyky, které mění slova jen tímto způsobem (např. Angličtina). Pro češtinu je však stematizace hůře aplikovatelná, protože se slova ohýbají podle složitějších pravidel, ne jen změnou předpon, přípon nebo koncovek. Stematizaci tedy lze využít při hledání základního tvaru slova i v českém jazyce, je ale nutné ji rozšířit o další procesy, které jsou popisovány dále.

4.4.5 Lematizace

Lematizátor je nástroj, který převádí slovo na jeho základní gramatický tvar, tzv. lemma. Základním gramatickým tvarem se rozumí první pád jednotného čísla u podstatných jmen nebo infinitiv u sloves. Celý proces se nazývá lematizace a kromě převodu na základní tvar může lematizátor disponovat doplňkovou funkcí, kdy pro daný tvar poskytne i jeho vlastnosti, například slovní druh, pád nebo číslo. Lematizátor pracuje v podstatě opačně ve srovnání s derivátorem. Lematizace zároveň není totéž co stematizace, které předpokládá, že dochází jen k přidávání předpon a přípon. Lematizátor jej vlastně rozšiřuje, odstranění předpon nebo přípon však může být jednou jeho z činností. Především ale počítá s tím, že se koncovky nebo kmeny slov mohou měnit, což je důležité pro češtinu. Dalším rozdílem je výsledek daného procesu – po stematizaci může vzniknout díky ořezání předpon a přípon neexistující slovo, výstupem lematizace je však vždy slovo existující.

Algoritmů pro implementaci lematizátoru je více, triviálním řešením může být použití slovníku, který obsahuje veškerou slovní zásobu daného jazyka a všechny tvary těchto slov. V něm je pak nalezeno slovo v daném tvaru a k němu odpovídající základní tvar. Výhodou takového přístupu je přesnost lematizace, pokud je již daný tvar slova znám, je triviální najít přesný základní tvar. Jediný problém může způsobit duplicita daného slova, kdy se musí lematizátor rozhodnout, na který základní tvar

jej upraví. Nevýhodou je poté nutnost vytvoření takového slovníku, kdy je třeba popsat veškerou slovní zásobu.

Další možností je algoritmická lematizace, kdy jsou definována pravidla, podle kterých jsou slova ohýbána a ta jsou používána pro hledání základního tvaru. Využití takového přístupu znamená daleko menší soubor pravidel, podle kterých je lematizace prováděna ve srovnání s kompletním slovníkem slovní zásoby. Je tedy rychlejší takový lematizátor vybudovat od počátku a ve srovnání s použitím slovníku je velká šance, že bude správně zpracováno neznámé slovo. Pokud chybí ve slovníku, nelze rozhodnout, jak lema vytvořit. Pokud však budeme mít definovanu jen sadu pravidel, spíše bude lematizace provedena úspěšně. Zřejmou nevýhodou ve srovnání s využitím slovníku je kvalita takové lematizace. Čeština je dosti nepravidelná a existuje mnoho výjimek, které se při ohýbání slov vyskytují.

Konečně lze také pro lematizaci využít stochastických metod, respektive metod strojového učení. Ručně se vytvoří trénovací data, která se následně použijí pro vytvoření modelu. Na základě tohoto modelu lze provádět samotnou lematizaci, přičemž použitelných metod strojového učení je celá řada.

4.5 Problematika významnosti

Další problém, který je třeba řešit je významnost výrazů, podle nichž je vyhledáváno. Ne všechna slova jsou pro vyhledávání využitelná, protože nenesou žádnou užitečnou hodnotu, nebo jsou tak častá, že je jejich využití degradováno.

Zároveň však ne všechna slova charakterizují daný dokument stejnou mírou. Nelze se tedy jen rozhodovat, zda dané slovo ukládat do invertovaného indexu, je také třeba pamatovat na rozdílnou informační hodnotu slov vůči dokumentům – ať už se jedná o indexovaný dokument, nebo o celou jejich sadu.

4.5.1 Negativní slovník

Negativní slovník, někdy označovaný také jako stop-slovník nebo slovník stop-slov, je datová struktura obsahující slova, která jsou v daném případě považována za bezvýznamná z pohledu vyhledávání. Bývají to slova určitých slovních druhů, nebo obecně slova, která nenesou žádný význam. Obvykle jde o předložky, spojky nebo zájmena. Někdy také může jít o podstatná jména, která se v dané oblasti vyskytují tak často, že jsou pro vyhledávání nepoužitelná.

Kromě výše uvedených slovních druhů by také bylo možné použít některá nejčastěji používaná slova v českém jazyce [26]. Pokud se podíváme na prvních 5 nejčastěji používaných přídavných jmen (jiný, určitý, další, nový, velký), zjistíme, že právě tato jsou slova, která pravděpodobně nepůjdou pro vyhledávání dobře použít. Pro vytvoření negativního slovníku tak bude potřeba existující slovník obsahující slova potřebných slovních druhů a některá tato nejpoužívanější slova.

Další možností, která může jednoduše vyloučit spojky a předložky, je odfiltrování krátkých slov. Vzhledem k tomu, že většina takových slov má délku jeden až dva, nejvýše tři znaky, je možné to v filtru zohlednit. Hlavní výhodou tohoto přístupu je

rychlost oproti vyhledávání slova v slovníku. Je tedy optimální oba přístupy kombinovat, nejprve odfiltrovat krátká slova a poté využít slovníku stop-slov, která se tím výrazně zmenší (je možné krátká slova vypustit) a celá indexace se tím zrychlí.

4.5.2 Významnost výrazů

Důležitá veličina při indexaci výrazu je jeho významnost, neboli míra reprezentace indexovaného dokumentu. Některé výrazy totiž vystihují daný dokument přesněji než jiné. Uvažujme produkt s názvem **Jablko Idared červené**. Slovo **Jablko** tento produkt vyjadřuje poměrně dobře, ale stále ještě příliš obecně. Pokud uživatel vyhledává podle tohoto výrazu, stále ještě nelze rozhodnout, že je tento produkt ten, který hledal. Slovo **Idared** již reprezentuje konkrétní produkt daleko přesněji, žádný jiný produkt se takto pravděpodobně nejmenuje. Opakem je pak slovo **červené**, které vystihuje produkt nejméně, červené mohou být i jiné potraviny (rajčata) nebo dokonce úplně jiné produkty (například petrklíč červený).

Z uvedeného příkladu lze vypořádat, že významnost výrazu souvisí s jeho výskytem v celém souboru indexovaných dokumentů. Přesněji jde o vztah mezi frekvencí výrazu (TF – term frequency) a frekvencí v invertovaných dokumentech (IDF – inverse document frequency) [2, strana 21]. Ty lze vypočítat pomocí vzorců:

$$\text{tf}(t, d) = 0.5 + 0.5 \cdot \frac{f_{t,d}}{\max\{f_{t',d} : t' \in d\}}$$

$$\text{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

Samotná váha daného výrazu (TF-IDF) je definována jako součin těchto hodnot:

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D)$$

Pro zpřesnění tohoto výpočtu je možné dále pracovat s váhou a to jak u frekvence výrazu, tak u frekvence v invertovaném indexu. Hodnoty bývají nejčastěji normalizovány nebo logaritmovány.

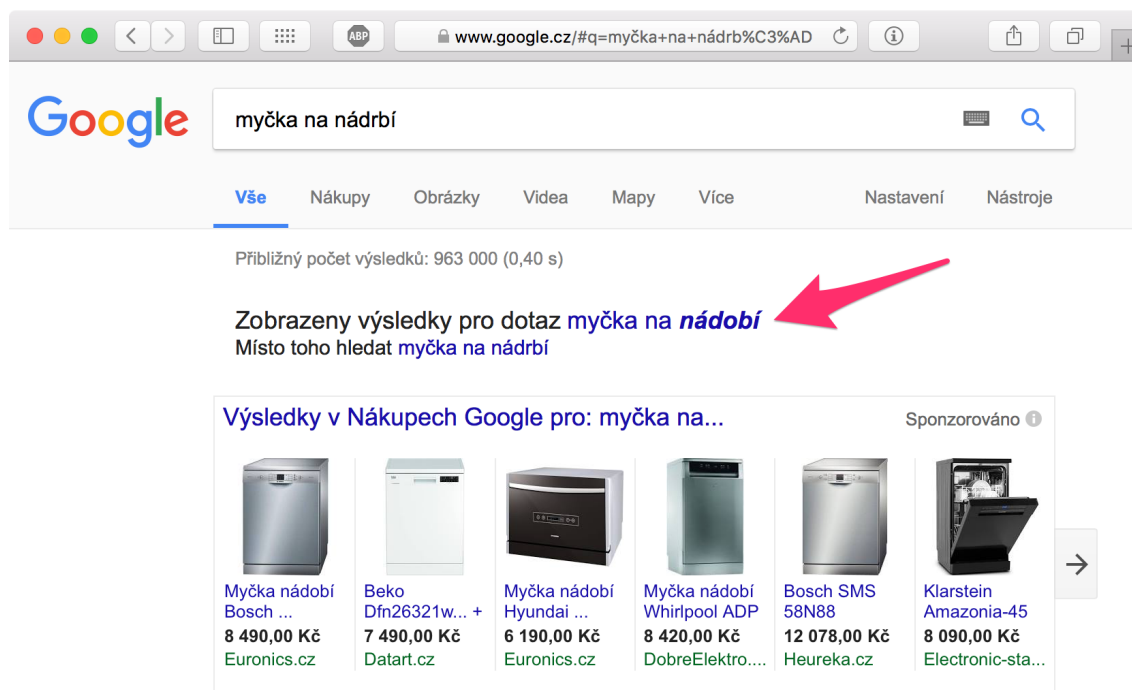
4.6 Tezaurus

Tezaurus označuje slovník, který charakterizuje vztahy mezi slovy. Zachycuje podobnost, nadřazenost nebo podřazenost mezi slovy. Lze jej využít při indexaci nebo formulaci dotazu a s jeho pomocí je možné daná slova rozšířit o slova s obdobným významem, přičemž význam může být obecnější nebo naopak přesnější. Tezaurus je díky množství vztahů které popisuje propracovanější než jen seznam synonym. Lze jej využít i pro neobvyklá slova, jako je žargón nebo další expresivní výrazy.

Vytvoření takového slovníku je závislé na oblasti, pro kterou je vytvářen. Například slovo **oko** bude nahraditelné jinými slovy v kontextu gastronomie a jinými slovy v oboru biologie. Nelze tedy bezmyšlenkovitě používat jeden takový slovník globálně pro veškeré vyhledávání, je třeba zohlednit obor, ve kterém bude využíván.

4.7 Přibližné vyhledávání

Dalším komplexním problémem, kterým je třeba se zabývat je přibližné vyhledávání, tedy vyrovnání se s překlepy nebo schopnost vyhledat dokumenty už v okamžiku, kdy je formulován vyhledávaný dotaz. Tato funkčnost je označována jako search-as-you-type nebo také incremental search.



Obrázek 4.3: Odhalení překlepu ve vyhledávání na webu google.com

U přibližného vyhledávání je třeba se vypořádat s měrou, jak přibližný daný výraz je vůči výryzu hledanému, což lze vyjádřit pomocí editační vzdálenosti. Přistoupit k této problematice lze i jednoduším přístupem – generováním n-gramů.

4.7.1 Editační vzdálenost

Editační vzdálenost je způsob, jak vyjádřit podobnost dvou textových řetězců. Je vyjádřena jako celé číslo, které udává počet operací nutný pro transformaci z jednoho textového řetězce na druhý. Editačních vzdáleností pro textové vyhledávání je více, podle toho, jaké operace na textovém řetězci umožňují.

V roce 1965 Vladimir Levenshtein definoval **Levenshteinovu vzdálenost** jako počet změn znaků vedoucí k transformaci z jednoho textu na druhý [27]:

- Nahrazení jednoho znaku jiným: **jxblko** → **jablko**
- Vložení jednoho znaku: **jaxblko** → **jablko**
- Odstranění jednoho znaku: **jblko** → **jablko**

Frederick Damerau ji později rozšířil o další transformaci, která má shodnou váhu:

- Prohození dvou znaků: **jbalko** → **jablko**

Tato transformace vlastně nahrazuje více dílčích operací definovaných výše. Vzdálenost s touto transformací o hodnotě 1 je označována jako **Damerau–Levenshteinova vzdálenost**. Počtem transformací se zvyšuje editační vzdálenost a zároveň s ní se snižuje pravděpodobnost, že daný textový řetězec je překlepem druhého řetězce. Damerau vyzníval, že 80% překlepů má editační vzdálenost rovnou jedné [28]. Je tedy výrazně nižší pravděpodobnost, že v textu bude tolik chyb, aby byla vzdálenost vyšší, což lze zohlednit při konstrukci vyhledávacího algoritmu a bude mít zřejmě pozitivní vliv na rychlost vyhledávání.

Další vzdáleností je **Hammingova vzdálenost**, která počítá pouze s náhradou znaku v textovém řetězci. Z tohoto omezení je však patrné, že je použitelná pouze pro řetězce shodné délky. V praxi textového vyhledávání tedy budeme pracovat pravděpodobně s vzdálenostmi definovanými výše.

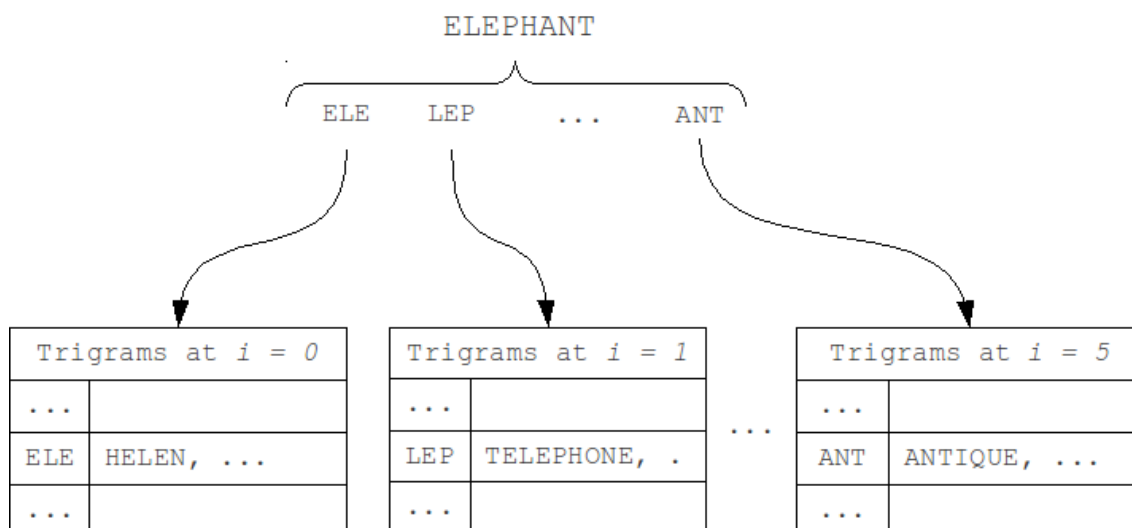
Na mírně odlišném principu funguje **vzdálenost nejdelšího společného podřetězce** (nebo také posloupnosti – LCS distance). Jejím úkolem je nalézt nejvyšší možný za sebou jdoucí počet znaků, který je společný oběma řetězcům. Tento přístup je ale méně chodný pro vyrovnání se s chybami uprostřed slov, navíc je výpočetně náročnější.

Další z možných vzdáleností je **Jarova vzdálenost**, která kromě počtu změn na znacích počítá i počet shodných vzdáleností. Jejím rozšířením je **Jaro-Winklerova vzdálenost**, která navíc počítá s tím, že shodnost na začátku řetězce má vyšší váhu, než na jeho konci – vychází totiž z pozorování, že na začátku textu se vyskytuje méně chyb [30]. Tento přístup také dosahuje lepších výsledků u krátkých slov.

4.7.2 N-gramová podobnost

Vytváření n-gramů je používáno při indexaci, kdy jsou slova dělena na části dlouhé *n* znaků. Pokud bychom měli vygenerovat n-gramy o délce 3 (trigramy) pro slovo **telefon**, byly by to výrazy: **tel**, **ele**, **lef**, **efo** a **fon**. Lze si všimnout, že jednotlivé vygenerované výrazy se překrývají. Pokud pak uživatel zadá jako hledaný výraz **telef**, bude odpovídat trigramům **tel**, **ele** a **lef**, tedy třem z celkem pěti. Trigramy jsou společně s digramy (n-gramy o délce dvou znaků) nejvhodnější vzhledem k obvyklé délce slov. N-gram o délce 1 pak ztrácí smysl výše nastíněné logiky.

Porovnávání slov pomocí n-gramů je znázorněno na následujícím schématu. Zde je zadané slovo rozděleno na trigramy, z nichž každý je dále porovnávám s trigramy indexovaných slov.



Obrázek 4.4: Vyhledávání pomocí n -gramů

Tento mechanismus lze použít pro našeptávání i pro odhalení překlepů. Je snadno implementovatelný a poměrně rychlý, selhává však u překlepů v krátkých slovech [29]. Předpokládejme slovo o délce 5 znaků – **mobil**. Vzniklé trigramy jsou **mob**, **obi**, **bil**. Pokud uživatel vyhledá slovo s překlepem přesně uprostřed (**movil**), nebude nalezena shoda v žádném trigramu.

Řešením by mohlo být snížení počtu znaků v n -gramu, což může na druhou stranu mít negativní vliv na rychlost. Možným kompromisem tak může být různá délka n -gramů, kdy by vznikly kratší n -gram pro začátek a konec slov. Konkrétně pro slovo **mobil** by tak vznikly n -gramy **m**, **mo**, **mob**, **obi**, **bil**, **il**, **l**.

4.8 Relevance

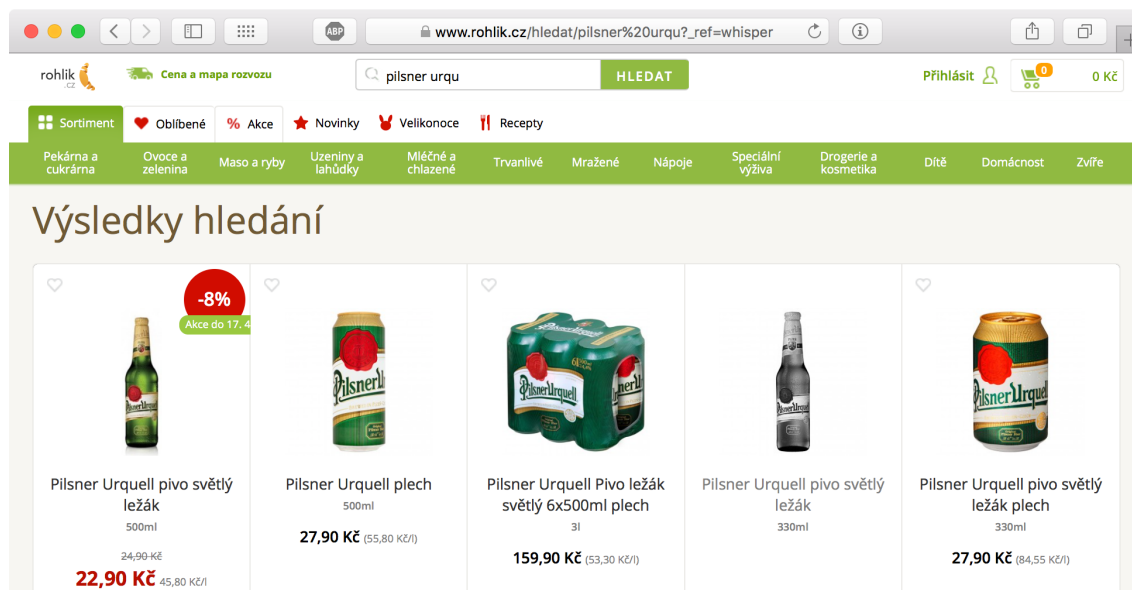
Relevancí rozumíme míru, jakou odpovídá nalezený dokument zadanému zadanému dotazu. Je použitelná pro řazení nalezených dokumentů, kdy jsou nejprve zobrazeny dokumenty více odpovídající hledanému výrazu. Dále je možné se na základě relevance rozhodnout, zda ještě nalezený dokument hledanému výrazu odpovídá, zda je vůbec třeba jej zobrazit.

Relevance je do jisté míry subjektivní, protože každý zákazník vyhledává na webu s jiným očekáváním, přestože třeba vyhledává pomocí stejných výrazů. Objektivně lze však relevanci vyjádřit pomocí TF-IDF, tedy frekvence nalezených výrazů vůči jejich frekvenci v invertovaném indexu.

V praxi elektronických obchodů však často do řazení vstupují další veličiny, jako je cena produktu, skladovost nebo výše marže. Pokud například uživatel vyhledá **iPhone**, může být žádoucí nejprve zobrazit produkt **iPhone SE 64GB Vesmírně černý**, než jeho příslušenství, například **Sportovní obal na iPhone SE** z důvodu několikanásobně vyšší marže na tomto produktu.

Elektronické obchody často také operují s akčním zbožím. Zejména v České re-

publice je obchodování se zlevněným zbožím populární, může být tak žádoucí zobrazit takové produkty dříve, než produkty prodávané za standardní cenu.



Obrázek 4.5: Řazení výsledků – upřednostnění akčního zboží na webu rohlik.cz

Pořadí výsledků nemusí být ovlivňováno jen zájmy provozovatele, mohou jej formovat sami zákazníci svým chováním. Konkrétně v případě vyhledávání lze pozorovat proklikovost jednotlivých produktů vzhledem k zadanému dotazu. Pokud se ukáže, že na základě daného dotazu zákazníci nejčastěji klikají na konkrétní produkt, je možné jej zobrazit ve výsledcích vyhledávání pro tento dotaz dříve.

S tímto přístupem je však možné jít ještě dál a začít vytvářet uživatelské profily zákazníků podle toho, o jaké zboží se zajímají, co vyhledávají, jaké jsou jejich údaje zadané při registraci a objednávce (lokalita, pohlaví), nebo v jaké časy nakupují prostřednictvím jakých zařízení. Tato problematika je již složitější, nicméně v posbíraných datech a vytvořených uživatelských profilech lze hledat další souvislosti pomocí statistických metod a strojového učení. Konkrétně lze zákazníka zařadit do určité skupiny pomocí shlukové analýzy a podle toho k němu přistupovat (upravit řazení výsledků vyhledávání) nebo využít algoritmů doporučovacího systému.

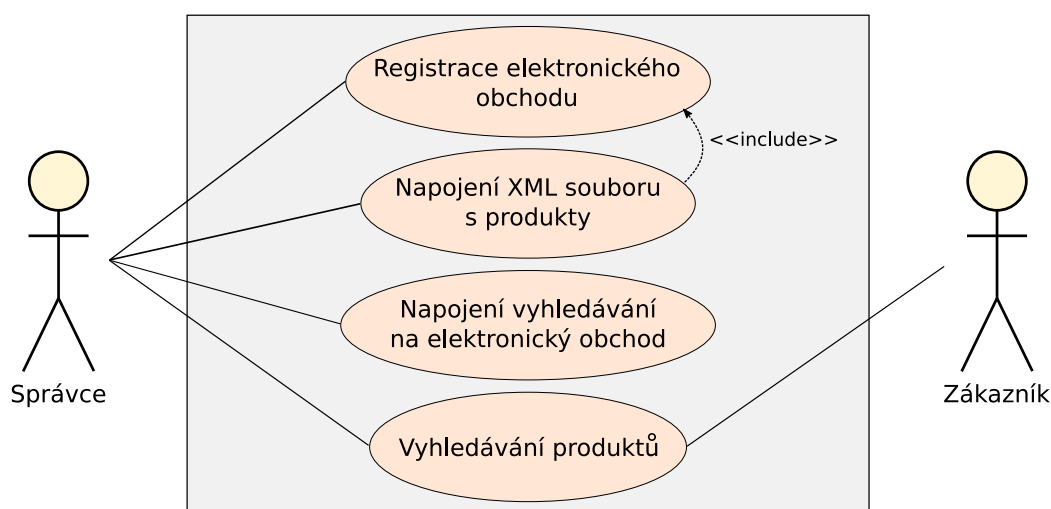
Samotné řazení všech uložených dokumentů může být výpočetně náročné, v tu chvíli lze řadit dokumenty ve dvou průchodech. V prvním průchodu jsou nalezeny dokumenty odpovídající zadanému dotazu a případně jřazeny s pomocí nenáročného výpočtu. Poté lze provést druhé řazení nad výběrem prvních dokumentů. Díky tomu, že už probíhá řazení nad malou množinou dat, je možné využít výpočetně náročnějších algoritmů.

5 Návrh řešení

V této kapitole vytvářím návrh aplikace umožňující implementaci plnotextového vyhledávání do elektronického obchodu, které bude možné provozovat jako službu. Popisuji způsob, jakým bude aplikace používána, aktéry, kteří v užívání aplikace vystupují a vytvářím její model, který bude požadavkům nejlépe odpovídat.

5.1 Definice případů užití

Aplikaci používají dva typy uživatelů. Správce (provozovatel elektronického obchodu), který zajišťuje její konfiguraci (například napojení vstupních dat) a zákazník elektronického obchodu, který provádí samotné vyhledávání. Z toho vyplývá, že ty části aplikace, které používá správce musí být zabezpečeny, aby k nim neměl přístup nikdo jiný. Navíc musí aplikace umět pracovat s více správci zároveň, kdy má každý přístup pouze ke svým datům. Pro zákazníka musí být naopak jeho část aplikace veřejně přístupná, musí být také co nejrychlejší. Právě zákazník bude provádět vyhledávání a rychlost, s kterou obdrží výsledky je proto klíčová.



Obrázek 5.1: Případy užití aplikace

Toto jsou základní případy užití z nejméně podrobného pohledu. Jednotlivé případy užití popisují podrobně níže.

5.1.1 Registrace do aplikace

Toto je první případ užití a jeho splnění je předpokladem pro další případy užití. Budoucí správce, který chce aplikaci používat musí mít vytvořen účet, který je přístupný po zadání jeho přihlašovacích údajů.

- Správce zobrazí registrační formulář
- Správce zadá údaje pro svůj účet (e-mailová adresa a heslo)
- Správce zadá adresu souboru XML s produkty a odešle formulář
- Správce je přihlášen pod novým účtem, je spuštěn import produktů

Výsledkem tohoto případu je vytvořený nový účet. Při ukládání systém kontroluje unikátnost zadaného uživatele a na základě zadané URL produktového XML vytváří novou konfiguraci daného uživatele a na pozadí spouští import produktů.

5.1.2 Napojení XML souboru s produkty

Tento případ užití lze jednak provést samostatně ve chvíli, kdy je správce přihlášený do aplikace, je ale také dílčí operací již při registraci.

- Správce zobrazí formulář pro zadání URL produktového XML
- Správce zadá adresu souboru XML s produkty a odešle formulář
- Je spuštěn import produktů na pozadí, je zobrazena informace správci

Po provedení tohoto případu užití je aktualizována (nebo vytvořena pokud neexistuje) konfigurace daného uživatele. Zároveň je na pozadí spuštěn import produktů, aby byla zajištěna aktuálnost produktové databáze.

5.1.3 Napojení vyhledávání na elektronický obchod

Ve chvíli, kdy má správce v aplikaci uložené produkty, může již využít samotné vyhledávání, které je poskytované jako služba. Toto napojení musí být maximálně intuitivní.

- Správce se přihlásí do aplikace
- Správce přejde na stránku zobrazující manuál k implementaci napojení
- Správce upraví elektronický obchod tak, aby byly požadavky na vyhledávání směřovány na URL uvedenou v manuálu

5.1.4 Vyhledávání produktů

Nejčastějším případem užití je vyhledávání prováděné zákazníkem. To je možné až ve chvíli, kdy je implementovaný jak import produktů, tak samotné vyhledávání.

- Uživatel v elektronickém obchodu začne psát dotaz do textového pole pro vyhledávání
- Při každém napsaném nebo smazaném písmenu je odeslán HTTP požadavek na API vyhledávání
- Pro každá odeslaný požadavek je aktualizován výpis nalezených produktů a tento požadavek je na serveru zaznamenán

Vzhledem k důležitosti tohoto případu je nutné aby bylo vyhledávání rychlé a stabilní. Každé zpoždění nebo výpadek totiž snižuje důvěru klienta používající vyhledávání.

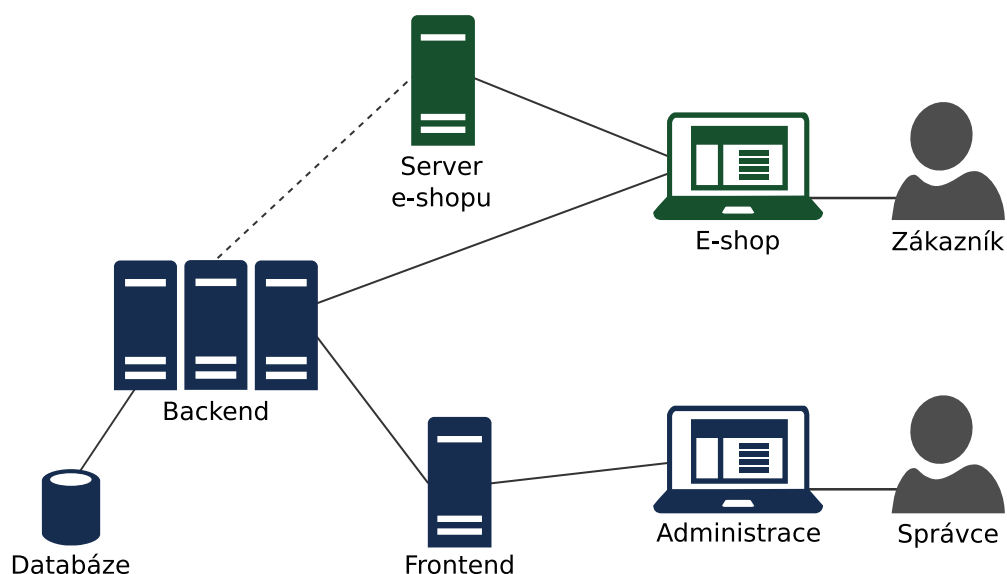
5.2 Návrh architektury aplikace

S aplikací bude komunikovat správce prostřednictvím grafického webového rozhraní a dále zákazník, který s aplikací komunikuje prostřednictvím grafického rozhraní elektronického obchodu. Ten musí s vyvíjenou aplikací komunikovat prostřednictvím jasně definovaného a jednoduše implementovatelného API.

Z tohoto důvodu se jeví jako výhodné rozdělit aplikaci na dvě části – první část (**backend**) bude obstarávat veškerou logiku, perzistenci dat a bude s ní možné komunikovat prostřednictvím API. Druhou částí (**frontend**) by byla samostatná webová aplikace s grafickým rohraním, která sama o sobě nepracuje s žádným úložištěm, ale všechny tyto požadavky nechává vyřídit backend [31]. Rozdělení na frontend a backend s sebou nese řadu dalších výhod – frontend může být vyvíjen v jazyce, který je pro to vhodnější, na obou částech můžou v budoucnu pracovat specializovaní vývojáři, obě aplikace je možné nezávisle škálovat. Zpřístupněním celého API navíc mohou vývojáři implementovat celé napojení plně automatizovaně.

Dále je třeba, aby byla architektura navržena tak, aby bylo možné aplikaci snadno škálovat. To se týká databáze i samotné aplikace dostupné prostřednictvím API. Databázi je tedy vhodné volit takovou, která umožní také horizontální škálování [33]. Dá se předpokládat, že s rostoucím počtem uživatelů služby bude rovnoměrně růst i potřeba na objem zpracovávaných dat. S horizontálním škálováním souvisí také možnost vytváření replik dat v úložišti, v takovém případě by byl cluster schopný se vyrovnat s případnou ztrátou některých dat. S možností snadného vertikálního škálování aplikace je třeba počítat při jejím nasazování. Prostředí, do kterého bude aplikace nasazována by mělo být schopné aplikaci škálovat obdobně jako databáze. Díky tomu, že veškerá data jsou persistována v databázi, je možné průběžně zapínat a vypínat jednotlivé instance aplikace bez obavy o ztrátu dat.

Na následujícím obrázku je znázorněn vztah mezi jednotlivými komponentami aplikace včetně napojovaného elektronického obchodu. Dále jsou zobrazeni oba aktéři.



Obrázek 5.2: Architektura aplikace – propojení důležitých komponent

Ze schématu je zřejmé, že zákazník si zobrazuje elektronický obchod, který je zobrazen na základě dat z serveru obchodu. Data pro vyhledávání jsou načtena prostřednictvím API aplikace, přičemž požadavek může být odeslán přímo z webového prohlézeče zákazníka (asynchronní HTTP požadavek) nebo prostřednictvím serveru obchodu (pokud probíhá vykreslení HTML stránky na serveru). Z druhé strany s aplikací operuje správce, který ji ovládá skrze administraci, která disponuje grafickým webovým rozhraním. Ta však veškeré požadavky směřuje na backend. Konečně backend používá pro persistenci dat (produktů i konfigurací) databázi, ostatním službám je pak dostupný skrze API.

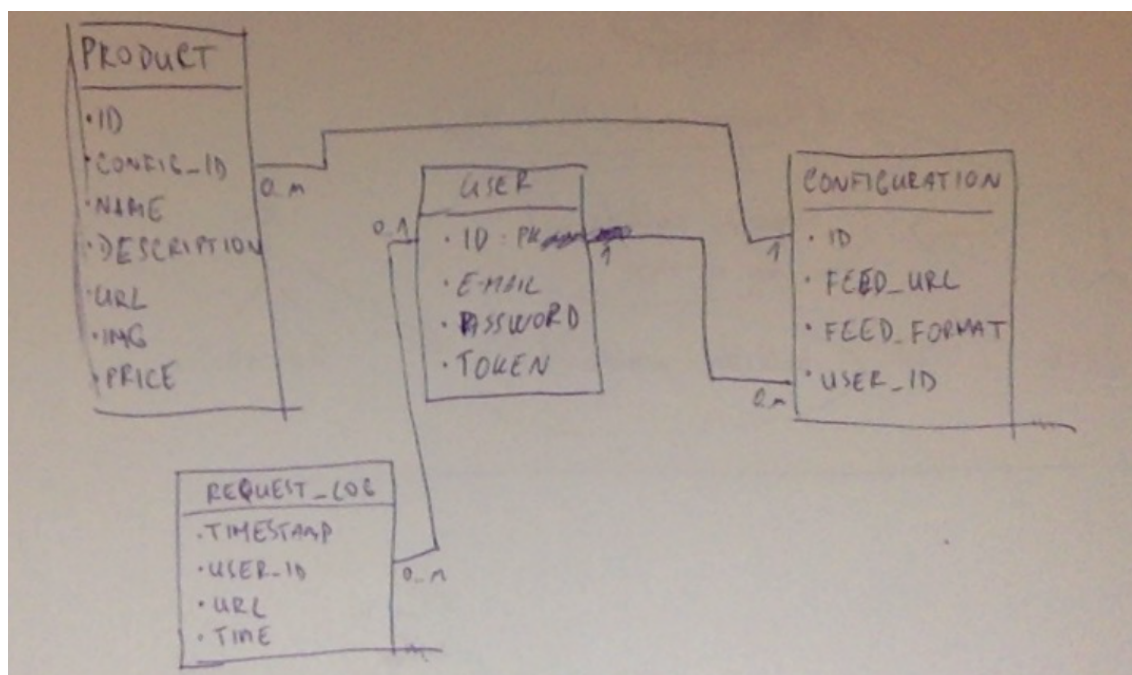
5.3 Datový model

Nyní definuji entity, které v aplikaci vystupují, jejich atributy a vztahy mezi nimi. V systému je primárně definován **uživatel** (správce), který se do aplikace registroval a vytváří konfiguraci pro import produktů. Pro každou tuto konfiguraci jsou do systému importovány produkty, jejichž struktura byla rozebrána v analýze dat, v kterých bude vyhledáváno.

U uživatele je zvolen jako jeho přirozený jednoznačný identifikátor jeho e-mailová adresa, kterou zadává při registraci. Dále je třeba uložit jeho heslo (respektive jeho hash) pro umožnění přihlášení. Posledním atributem uživatele je náhodně vygenerovaný token, který bude využit při přístupu k API klientem. Je totiž třeba zajistit, aby nebylo možné triviálním způsobem procházet produkty dalších uživatelů [7, strana 16].

Uživatel si dále vytváří konfiguraci, ve které je uložena adresa s XML souborem obsahujícím produkty, informace o jeho formátu (v případě že bude podporováno více formátů, je to vhodné pro detekci formátu a použití vhodného parseru). Tato **konfigurace** náleží právě jednomu uživateli, v jiných případech nemá její existence význam.

Další entitou, s kterou je třeba pracovat je **produkt**. Právě ten bude vyhledáván. Jeho atributy v nejjednodušší podobě jsou jeho jednoznačný identifikátor nebo kód (v rámci daného obchodu), název, popis, URL, adresa obrázku a cena. Produkt je možné rozšířit o další atributy, nicméně toto jsou atributy, které by měly být u každého produktu známy.



Obrázek 5.3: Datový model aplikace

Poslední entitou znázorněnou v diagramu je zalogovaný požadavek. Vzhledem k tomu, že chceme mít jednak kontrolu nad rychlostí vyhledávání, tak je dobré mít přehled o využití aplikace ať už jako celku, tak vzhledem k jejímu využití jednotlivými obchody. Každý takový záznam musí obsahovat datuma čas, kdy byl zaznamenán, URL, na kterou byl požadavek odeslán a čas, který trvalo jeho vyřízení.

5.4 Návrh API

Klíčový je návrh API, jehož prostřednictvím je dostupná veškerá logika aplikace. Při jeho návrhu jsem se rozhodl využít přístupu design-first [32], kdy je nejprve navrženo samotné API (vytvořena jeho dokumentace) a až poté je přistoupeno k samotné implementaci. Pro jeho návrh použiji nástroj Apiary [13], který umožňuje pracovat s API pomocí tohoto přístupu.

Před konkrétním návrhem je třeba rozmyslet protokol a další principy, kterých API využívá. Přestože by se mohlo zdát, že to je až záležitost samotné implementace, tento krok zásadně ovlivňuje již způsob návrhu API. Dále je to důležité k použití přístupu design-first.

Pro naplnění požadavků vycházejících z informací o problematice elektronického obchodování je třeba volit takový protokol, který je běžně dostupný napříč všemi běžnými webovými prohlížeči a je možné jej využít v používaných webových technologiích. Takovým protokolem je HTTP (případně novější verze HTTP/2). V Nejnovějších prohlížečích by sice bylo možné používat pokročilejší protokoly (například WebSockets [35]), aktuálním cílem je však maximální možná kompatibilita a snadnost implementace. Samotná komunikace probíhá dle principu REST, data jsou zasílána ve formátu JSON, což jsou nejpřímochařejší způsoby návrhu a implementace [7]. Bylo by možné využít jiných mechanismů (XML-RPC nebo SOAP), nicméně pro tento konkrétní případ jsou méně vhodné vzhledem k jejich snadnosti implementace, pochopitelnosti a rychlosti [7, strana 31].

5.4.1 Datové struktury

Nejprve je třeba identifikovat zdroje (resources), což jsou datové struktury, které budou prostřednictvím API předávány. Dále je třeba definovat operace, které lze nad jednotlivými zdroji provádět (čtení, vytváření, mazání, aktualizace) a nakonec také vhodně navrhnout koncové body (endpoints) [7, strana 12], pod kterými bude každá operace dostupná.

Základní zdroje, s kterými API operuje jsou **produkt** a **konfigurace uživatele**. Tyto struktury nesou důležitou informační hodnotu, první veškeré informace o produktu, které může být třeba v elektronickém obchodu zobrazit, druhá pak informace o nastavení daného uživatele. Tyto struktury lze popsat pomocí syntaxe API Blueprint [7, strana 131] následovně:

***Výpis 5.1:** Dokumentace API – struktura produktu*

```
# Data Structures

## Product (object)
- id: `1mWa9` (string, required) - ID of product
- userId: `01dsdmmwv2dwkdla8di8` (string, required) - User ID
- name: `Postel bílá` (string, required) - Name of product
- description: `Postel vyrobená z masivu...` (string, required) -
  ↳ Product description
- url: `http://shop.tld/1mWa9` (string, required) - detail URL
- img: `http://shop.tld/1mWa9.jpg` (string, required) - image URL
- price: `1` (number, required) - Current price of product incl. VAT
- updated: `2016-12-04T23:22:16+01:00` (string, optional) -
  ↳ Timestamp of last update
```

***Výpis 5.2:** Dokumentace API – struktura nastavení uživatele*

```
## UserConfig (object)
- id: `01dsdmmwv2dwkd1am8di8` (string, required) - User ID
- token: `7Tnl6xPQhVPiv80ryboo4iTckqXVuG4xjWFXTC1mzU3EaGY5`
  ↳ (string, required) - security token for API access
- email: `user@email.com` (string, required) - e-mail address
- password: `p4ssw0rd` (string, optional) - new password
- feedUrl: `http://shop.tld/feed.xml` (string, required) - XML feed
- feedFormat: `heureka` (enum[string], required)
  - `heureka`
```

Zde struktura **Product** bude používána pouze pro čtení, produkty jsou vytvářeny na základě importovaného XML souboru. Konfigurace uživatele je používána jak při registraci (vytvoření nové konfigurace), tak při aktualizaci této konfigurace (čtení i změna).

Dalšími pomocnými strukturami jsou data pro provedení **přihlášení** správce do administrace. Ten při přihlášení zadává přihlašovací jméno a heslo, jako odpověď obdrží své ID a dále token, identifikující jeho sezení a použitelný pro vykonání dalších požadavků.

***Výpis 5.3:** Dokumentace API – struktura požadavků a odpovědí serveru*

```
## LoginRequest (object)
- email: `user@email.com` (string, required) - e-mail address
- password: `p4ssw0rd` (string, required) - password

## LoginResponse (object)
- userId: `01dsdmmwv2dwkd1am8di8` (string, required) - id of user
- token: `JT0n0rws8rRMx8P205hmc68c9yptDW91PkuLNcHe2J2NxQdy0`
  ↳ (string, required) - auth token
```

Poslední struktury, které API používají, definují formát odpovědi pro operace, které neočekávají žádná data, pouze je modifikují (mutace). Po provedení takových operací API informuje o úspěšnosti jejich provedení odpovídajícím stavovým kódem. V případě neúspěchu přidává srozumitelnou hlášku včetně detailního popisu o důvodu neúspěchu, což usnadňuje práci vývojářům napojovaných aplikací a odstraňování chyb [7, strana 42].

***Výpis 5.4:** Dokumentace API – struktura odpovědi v případě chyby*

```
## Error (object)
- status: `400` (number, required) - HTTP status code
- message: `Unable to create user` (string, required)
- description: `Email already exists` (string, optional)
```


5.4.2 Koncové body

Pro každou operaci je definován koncový bod (endpoint) a metoda, pomocí které je operace dostupná, přičemž koncový bod je URL, pod kterou je dostupný. Ta je uváděna relativně vzhledem k doméně, pod kterou je aplikace provozována. Dále jsou všechny mají adresy prefix `/api/v1`, který usnadní možné budoucí aktualizace, které nejsou zpětně kompatibilní.

Koncové body jsou pojmenovány podle zdrojů, s kterými pracují, aby bylo jejich používání intuitivní. Prvními jsou body pro provedení přihlášení a odhlášení, které operují s zdroji **LoginRequest** a **LoginResponse**, respektive **Ok** a **Error**.

- `POST /api/v1/sign/in` – přihlášení do administrace
- `POST /api/v1/sign/out/{token}` – odhlášení z administrace

Další množina bodů provádí operace se zdrojem **UserConfig**. Slouží pro registraci nového uživatele, aktualizaci údajů stávajícího uživatele, zobrazení registračních údajů a případnou deaktivaci účtu stávajícího uživatele.

- `GET /api/v1/user/{token}` – získání konfigurace uživatele
- `PUT /api/v1/user/{token}` – vytvoření nebo aktualizace konfigurace
- `DELETE /api/v1/user/{token}` – deaktivace uživatele

Posledním koncovým bodem je vyhledávání produktů. Tento bod je jediný veřejný a použitý v elektronickém obchodě. Tento bod přejímá dva parametry, kterými jsou ID uživatele (obchodu) a hledaný výraz. Data jsou získávána jako pole objektů **Product**, které je ještě zanořeno v dalším poli, kde je dostupné pod klíčem **products**. To usnadní budoucí možné rozšiřování o další data (výsledky agregačních dotazů nebo například stav stránkování) s zachováním zpětné kompatibility.

- `GET /api/v1/search/{userId}/{query}` – vyhledání produktů dle **query**

Tento koncový bod musí být v administraci aplikace zadokumentován a uveden s názornými příklady, jak jej napojit do elektronického obchodu na základě konkrétní použité technologie.

Celé API je dokumentováno v formátu API Blueprint a je distribuováno společně s zdrojovými kódy aplikace, které jsou přílohou této práce. Současně je vytvořen projekt v nástroji Apiary, ve kterém je dokumentace uložena a veřejně přístupná pod URL <http://docs.apisearch.apiary.io>. Ta obsahuje i vzorová data, je tedy možné využít serverů proxy, které umožňují přistupovat k API, které vrací ukázková data. Je tedy možné začít nezávisle na sobě navrhovat a vyvíjet stranu, která API poskytuje (backend) i konzumuje (frontend). Vytvořený soubor ve formátu API Blueprint budiž smlouvou, která mezi těmito aplikacemi vznikla.

6 Implementace

V této kapitole je popisována implementace na základě předchozího návrhu. Popisují zde vše spojené s implementací, od výběru vhodných nástrojů přes samotnou implementaci a konfiguraci zvolených nástrojů až po nasazení do produkčního prostředí. text může sloužit jako návod pro implementaci vyhledávání poskytovatelného jako služba, kompletní zdrojové kódy jsou dostupné v příloze.

6.1 Výběr nástroje pro vyhledávání

Nejprve je třeba zvolit nástroj, který bude použit pro samotné vyhledávání. To podporují jak běžně používané relační databáze, tak specializované nástroje. U relačních databází jde spíše o rozšíření jejich funkčnosti a naopak – specializované nástroje lze použít jako databáze. Zde je třeba rozhodnout, který z těchto nástrojů bude nejlépe vyhovovat požadavkům na vyhledávání.

6.1.1 PostgreSQL

PostgreSQL (nebo také Postgres) je jedna z open-source relačních databází, která podporuje mnoho funkcí, které jsou dobře použitelné pro implementaci kvalitního plnotextového vyhledávání. Pro vyhledávání textu, který nepodléhá tvarosloví je možné využít operátorů `~`, `~*`, `LIKE` a `ILIKE`. Dále tato databáze umožňuje zpracování textu již při ukládání (indexaci) [36]. Do databáze lze uložit slovníky, které lze následně využít pro:

- Definici stop slov
- Mapování slov na jiná – synonyma pomocí `Ispell` a fráze pomocí `tezaury`
- Transformaci slov do jejich základní formy pomocí slovníku `Ispell` a pomocí pravidel stemmeru `Snowball`

Dále disponuje funkcemi pro implementaci fuzzy vyhledávání, konkrétně `levenshtein()` a `metaphone()`. Mezi nevýhody patří obtížnější instalace a konfigurace. Zejména, pokud bude třeba databázi škálovat.

6.1.2 Elasticsearch

Elasticsearch vznikl primárně jako nástroj pro textové vyhledávání, podporuje tak mnoho funkcí, které jsou pro textové vyhledávání použitelné. Interně používá Apache Lucene, rozšiřuje jej však o řadu funkcí a usnadňuje práci s ním používáním vlastního API. Dále je od počátku navržen jako distribuované úložiště, škálování tedy spočívá pouze v přidávání uzlů. Elasticsearch poté sám vytváří repliky dat a distribuje je napříč clusterem [14].

Umožňuje import slovníků, nastavení řady filtrů pro indexaci (synonyma, n-gramy, stematizaci, lematizaci a celou řadu dalších filtrů a analyzátorů [37]). Zároveň umožňuje precizně nastavit výpočet relevance, nad daty lze vypočítávat agregace, vše je navíc velmi rychlé a rychlost se zásadně nemění s přibývajícím daty. Elasticsearch je výrazně rychlejší ve srovnání s PostgreSQL – [38] a [39].

6.1.3 Apache Solr

Dalším nástrojem, který přichází v úvahu je Apache Solr [40]. Shodně jako Elasticsearch využívá interně Apache Lucene. Co se funkcí týče, je tedy téměř srovnatelný s použitím Elasticsearch. Ten jej však převyšuje intuitivností instalace, snadnější možností použití analytických dotazů a výkonností a stabilitou vzhledem k škálování [41]. Přestože aktuálně nejsou zmíněné výhody tolik důležité, vzhledem k očekávanému rozšiřování je vhodné je vzít v potaz a dále tak uvažují pouze nad Elasticsearch. Do závěrečného porovnání jej nezahrnuji, protože kromě výše uvedených nuancí by bylo hodnocení obdobné.

6.1.4 Sphinx

Posledním nástrojem, který přichází v potaz je Sphinx Search [42]. Ten se zaměřuje především na rychlost, je napsán v C++, je také relativně nenáročný na paměť. Pro potřeby vyhledávání má dostatek funkcí, neumí ale nic jiného, zaměřuje se pouze na textové vyhledávání.

Na rozdíl od výše uvedených nástrojů Sphinx není úložiště. Je nutné jej provozovat jako nadstavbu nad stávající databází (například MySQL nebo Oracle). Sphinx umí pouze indexovat dokumenty a následně v nich vyhledávat, jejich načtení již však musí být provedeno odjinud. Provoz s další databází by tedy v tomto případě znamenal zbytečné zesložnění architektury.

Jeho použití by ale bylo daleko vhodnější, pokud by již byla jiná databáze využívána a pouze ji bylo třeba rozšířit o pokročilé textové vyhledávání. V takovém případě není třeba přecházet na jiné úložiště, pouze je rozšířena stávající funkčnost o plnohodnotné textové vyhledávání.

6.1.5 Porovnání nástrojů pro vyhledávání

V následující tabulce je zobrazeno přehledné porovnání dle parametrů, které jsou klíčové pro tento projekt.

Tabulka 6.1: Porovnání nástrojů pro vyhledávání

Obecný popis	
PostgreSQL	Relační databáze umožňující textové vyhledávání
Elasticsearch	Textové vyhledávání a škálovatelné úložiště
Sphinx	Textový vyhledávač
Snadnost instalace a konfigurace	
PostgreSQL	Snadná základní instalace, obtížnější pokročilá konfigurace
Elasticsearch	Intuitivní instalace, zabezpečení, monitoring nutné řešit pluginy
Sphinx	Snadná instalace
Škálování	
PostgreSQL	Obtížné na konfiguraci
Elasticsearch	Triviální, horizontální škálování je automaticky předpokládáno
Sphinx	Závislé na použité databázi
Výkonnost	
PostgreSQL	Rychlé pro jednoduché dotazy, s složitostí výkonnost klesá
Elasticsearch	Rychlé – automatická paralelizace, cachování
Sphinx	Velmi rychlé a nenáročné na paměť
Textové vyhledávání	
PostgreSQL	Podporuje většinu běžných funkcí
Elasticsearch	Propracované
Sphinx	Propracované
Podpora (dokumentace, literatura)	
PostgreSQL	Dostatečně dokumentováno, množství dostupné literatury
Elasticsearch	podrobná dokumentace, profesionální podpora za poplatek
Sphinx	Dostatečně dokumentováno

Na základě porovnání vychází vzhledem k požadavkům nejlépe použití Elasticsearch, přestože využití ostatních zmíněných nástrojů by bylo také možné. Elasticsearch je vhodný zejména díky výkonnosti, která se zásadně nemění díky možnostem škálování, množstvím podporovaných filtrů a analyzátorů a také množstvím dalších funkcí, které by mohly být využity v budoucnu, bez nutnosti přechodu na jinou technologii.

6.2 Nastavení a nasazení Elasticsearch

Aplikace využívá Elasticsearch verze 5.1.2. Jedná se o aktuální verzi, která však může být v době čtení této práce zastaralá, protože vydávání nových verzí probíhá

velmi často. Principy zde popsané se však nemění a s největší pravděpodobností budou ukázky fungovat i v novějších verzích Elasticsearch.

V rámci Elasticsearch se pracuje s pojmy, které je pro orientaci třeba vysvětlit:

- **Cluster** je několik (alespoň jeden) serverů komunikujících spolu
- **Node** je jedna spuštěná instance Elasticsearch, odpovídá jednomu serveru
- **Index** je skupina dokumentů, které mají společnou charakteristiku
- **Type** je skupina souvisejících dokumentů v rámci indexu
- **Document** je jedna zaindexovaná položka
- **Field** je atribut dokumentu (například název produktu, kde produkt je dokument a název je atribut)
- **Shard** je část indexu Elasticsearch (index Apache Lucene), umožňuje data fyzicky v indexu rozdělit a distribuovat je napříč clusterem
- **Replica** je kopie shardu, záloha umístěná na jiném node

S Elasticsearch se komunikuje přes REST API, kdy je možné zasílat dokumenty ve formátu JSON. To umožňuje snadnou implementaci klientských knihoven nebo komunikaci s serverem z příkazové řádky nebo pomocí univerzálních nástrojů pro práci s API.

6.2.1 Vytvoření Docker obrazu

Pro lokální vývoj i produkční nasazení je využíván Docker, což je nástroj umožňující běh aplikací v izolovaném prostředí [11]. Umožňuje vytvoření identického prostředí, ať už je aplikace spouštěna lokálně, při testech nebo v produkčním prostředí. Základním artefaktem je obraz (Docker image), který po spuštění vytvoří kontejner. Vztah obraz – kontejner lze chápat jako analogii k vztahu program – proces.

Pro vytvoření Docker image slouží soubor **Dockerfile**. V něm je definováno z jakého základního obrazu vychází a dále posloupnost kroků vedoucí k vytvoření požadovaného obrazu. Zde je zvolena oficiální verze Elasticsearch instalovaná v Alpine Linux, což je minimalistická linuxová distribuce zaměřená na bezpečnost [45]. Vychází z knihoven `musl libc` a programu `busybox` a přichází s principem, že Linuxová distribuce v základu nedisponuje téměř žádnými nástroji a je na uživateli, co si doinstaluje. To má příznivý vliv na velikost a rychlost. Taková distribuce je i bezpečnější – je minimalizováno množství míst, která mohou vést k zranitelnosti systému. Tento přístup je pro vytváření kontejnerů žádoucí a tak je distribuce Alpine Linux využívána stále častěji i v oficiálních obrazech [46].

Pro potřebu této práce je doinstalován plugin ICU, který je nutný pro korektní práci s českými znaky. Dále je do obrazu nakopírována složka `config` obsahující konfiguraci Elasticsearch v souboru `elasticsearch.yml` (výchozí název clusteru a

porty, přes které je Elasticsearch dostupný) a slovníky Hunspell [43]. To jsou slovníky obsahující českou slovní zásobu a veškeré informace potřebné pro morfologickou analýzu. Běžně jsou používány v volně dostupných aplikacích jako jsou LibreOffice nebo Mozilla Firefox. Slovník je tvořen dvěma soubory, v případě češtiny jsou to soubory `cz_CZ.dic` a `cz_CZ.aff`. První soubor obsahuje slovní zásobu, přičemž u každého slova jsou uvedeny příznaky, které definují, jakým slovem je slovo ohýbáno. V druhém slovu je popsána morfologie českého jazyka – pro každý příznak je definováno, jakým způsobem jsou odvozovány další tvary slova [44].

Konečná podoba souboru `Dockerfile` je tedy následující:

Výpis 6.1: Dockerfile pro Elasticsearch

```
# Základní obraz - oficiální Elasticsearch v Alpine Linux
FROM elasticsearch:5.1.2-alpine

# Instalace pluginu ICU
RUN bin/elasticsearch-plugin install analysis-icu

# Kopírování složky obsahující konfiguraci a slovníky
ADD config config/
```

Tento kód je sdílen prostřednictvím služby GitHub, která je napojena na Docker Hub. Obdobným způsobem, jako je sdílen kód prostřednictvím verzovacích nástrojů (VCS – například GIT nebo SVN) lze i sdílet obrazy Docker. Jako vzdálené úložiště (obdoba `origin` pro GIT) je použito tak zvaný registr (Registry), kterým je standardně Docker Hub. Provázání mezi GitHub a Docker Hub je na té úrovni, že při aktualizaci kódu v repozitáři (provedení příkazu `git push`) je automaticky spuštěno sestavení nového obrazu v službě Docker Hub (příkaz `docker build` a následně `docker push`). Po sestavení obrazu je nahrán do registru a od té doby je připraven ke stažení a spuštění. Sestavený obraz je možné stáhnout prostřednictvím příkazu `docker pull apisearch/elasticsearch` a lze jej využít jak pro lokální vývoj, tak pro produkční nasazení.

6.2.2 Nastavení indexace v Elasticsearch

Dále je třeba Elasticsearch nastavit pro správnou indexaci dokumentů. Jednotilá pole indexovaného dokumentu jsou rozdělena na slova pomocí tokenizéru, vzniklé tokeny jsou dále zpracovány filtry, které provádí dílčí transformace. Ty lze řetězit a vytvořit tak analyzátor, který popisuje celý proces.

První používaný filtr je typu `length`, který má na starosti odstranění slov skládajících se z jediného písmena. Následně je definován filtr pro odstranění stop slov `stop`. Ten využívá výchozí databázi stop slov rozšiřitelnou o libovolná další slova. Další filtr je typu `unique` a stará se o odstranění duplicitních slov. Nemá totiž smysl mít v indexu uloženo slovo vícekrát, již nepřináší další informační hodnotu. Tyto filtry rozhodují, zda bude dané slovo uloženo do invertovaného indexu. Uložit je lze do nastavení indexu následovně voláním HTTP požadavku metodou PUT na koncový bod `/products/_settings` (předpokládán je název indexu `products`) s následujícími daty:

***Výpis 6.2:** Nastavení filtrů pro indexaci v Elasticsearch*

```
{ "settings": {
  "analysis": {
    "filter": {
      "min_length": {
        "type": "length", "min": 2
      },
      "stopwords": {
        "type": "stop", "ignore_case": true,
        "stopwords": ["že", "právě", "_czech_"]
      },
      "unique": {
        "type": "unique", "only_on_same_position": false
      }
    }
  }
}
```

Další filtry mají na starost transformaci slov – převedení do základního tvaru, úpravu velikosti písmen nebo generování n-gramů. Velká písmena n malá převádí filtr `lowercase`. Díky tomu lze vyhledávat nezávisle na velikosti písmen, která uživatel zadává. Filtr, který převádí slova na jejich základní tvar se nazývá `hunspell`. Ten využívá k převodu slovník `hunspell`, který je již nainstalován do používané distribuce `Elasticsearch`. Zvláštním filtrem je `shingle`, který v podstatě vytváří n-gramy, činí tak ale na úrovni celých slov. Nastavení těchto filtrů lze provést následovně:

Výpis 6.3: *Nastavení filtrů pro indexaci v Elasticsearch*

```
{ "settings": {
  "analysis": {
    "filter": {
      "lowercase": {
        "type": "lowercase", "min": 4
      },
      "shingle": {
        "type": "shingle", "max_shingle_size": 3
      },
      "hunspell": {
        "type": "hunspell", "locale": "cs_CZ", "dedup": true
      }
    }
  }
}
```

Analytéry lze definovat jako řadu filtrů, které jsou zřetězeny, sekvenčně zpracovávají obdržený text, přičemž vstupem filtru je výstup filtru předchozího. Využít lze filtrů standardních nebo vlastních (definovaných výše). Dále je možné jedno pole nechat indexovat prostřednictvím více analyzérů a v těchto polích vyhledávat s rozdílnou váhou.

Analýzátor používaný pro vyhledávání v názvu produktu se skládá z filtru `icu_folding` (který byl instalován při vytváření Docker obrazu jako plugin) a z filtrů definovaných výše. Dále je v rámci nastavení analyzátoru zvolit tokenizér, což je nástroj, který dělí textový řetězec na slova. Nastavení analyzátoru vypadá následovně:

Výpis 6.4: *Nastavení analyzátorů v Elasticsearch*

```
"..."
  "filter": { "..." }
  "analyzer": {
    "icu": {
      "filter": [
        "icu_folding",
        "stopwords",
        "hunspell",
        "stopwords",
        "unique"
      ],
      "tokenizer": "standard"
    }
  }
"..."
```

Zbývá doplnit nastavení indexu, které nesouvisí přímo s indexací, ale s provozem Elasticsearch. Jednou z těchto konfigurací je nastavení počtu fragmentů (shards) a replik (replicas). Pro lokální vývoj nemá smysl jiné nastavení než počet fragmentů roven jedné a počet replik roven nule. Pokud není spuštěn Elasticsearch v režimu clusteru (neexistuje více uzlů), není jednoduše kam repliky ukládat. Stejně tak nemá smysl vytvářet více fragmentů, které znamenají pouze vyšší režii souborového systému. V produkčním prostředí je však situace jiná – vše se však odvíjí od množství dat a dostupných serverů., případně jader procesorů. První verzi aplikace je však spouštěna na jediném serveru s dvěma vlákny procesoru, je tedy vhodné začít s nastavením shodným s lokálním prostředím. S přibývajícími daty a servery je možné vytvářet index znovu s jinou konfigurací a data do nich přenést pomocí **Reindex API**. Konfigurace indexu tedy vypadá následovně:

Výpis 6.5: *Nastavení indexu Elasticsearch*

```
{
  "settings": {
    "number_of_shards": 1,
    "number_of_replicas": 0
  }
}
```


Nyní zbývá použít konfigurované analyzéry pro definici mapování (mapping), tedy přiřadit jednotlivým polím způsob, jakým jsou analyzovány. Obecně zde platí, že je vhodné každé pole ukládat podle odpovídajícího datového typu případně povahy dat. Například cenu ukládat jako `long` případně `double`, datum uložení produktu jako `date`, případné příznaky jako `boolean`. Speciální přístup platí pro ukládání textových řetězců, kdy je třeba rozhodnout, zda má dané pole podléhat textové analýze. V některých případech je to nežádoucí (EAN, ID uživatele), v některých naopak velmi důležité (název produktu). Pro první případ lze využít datového typu `keyword`, v druhém případě typu `text`. V tomto případě je třeba definovat také analyzér, který bude pro dané pole použit, přičemž jich může být více.

Výpis 6.6: *Nastavení mapování v Elasticsearch*

```
{
  "mappings": {
    "product": {
      "properties": {
        "id": {"type": "keyword"},
        "name": {
          "type": "keyword"
          "fields": {
            "hunspell": {"type": "text", "analyzer": "hunspell"}
          }
        },
        "price": {"type": "float"},
        "updated": {"type": "date", "format": "date_time_no_millis"}
      }
    }
  }
}
```

Kompletní podoba konfigurace indexu je součástí projektu a je k dispozici ve zdrojových kódech v příloze této práce.

6.2.3 Vyhledávání v Elasticsearch

Dále je třeba připravit dotaz, pomocí kterého bude provedeno vyhledávání v Elasticsearch. Celý dotaz může obsahovat několik částí – samotný dotaz (`query`), stránkování (`from` a `size`), řazení (`sort`), agregace (`aggs`), projekci (`_source`). V tomto případě je třeba získat celé dokumenty bez dalších modifikací nebo přidanych výpočtů, jde tedy o vytvoření dotazu v sekci `query`. Řazení v tuto chvíli probíhá dle relevance a taktéž stránkování je řešeno až dle dotazu klienta.

Celý dotaz se skládá z dílčích dotazů, které jsou následně spojeny v jeden. Pro textové vyhledávání dle zadaného dotazu je využito `multi_match_query`, což umožňuje vyhledávání ve více polích s definicí priority jednotlivých polí. Vyhledává se v analyzované titulku s nejvyšší prioritou a v neanalyzovaném titulku (pro posunutí produktů přesně odpovídající zadanému dotazu v pořadí výsledků) a v popisku s prioritou nižší. Dále je upraven způsob vyhodnocení výsledů na `most_fields`, což na rozdíl od výchozího chování bere v potaz výsledky vyhledávání ve všech polích (standardně je počítáno pouze s nejlepším z nich). Míra tohoto efektu je nastavena parametrem `tie_breaker`.

Výpis 6.7: Textové vyhledávání v Elasticsearch

```
{
  "query": {
    "multi_match": {
      "query": "hledaný výraz",
      "fields": ["name.hunspell^3", "name", "description"],
      "type": "most_fields",
      "tie_breaker": 0.3
    }
  }
}
```

Dále je dotazováno na název produktu s překlipy. Toho je docíleno pomocí `match_query` s nastaveným parametrem `fuzziness`. Překlipy jsou předpokládány pouze pro název produktu, pro dlouhé popisky by to mělo negativní vliv na rychlost navíc s pravděpodobně rostoucí nepřesností. Pro míru podobnosti textových řetězců je použita Levenshteinova editační vzdálenost, přičemž argumentem dotazu je velikost této vzdálenosti. Tolerovatelná vzdálenost může růst s délkou textového řetězce, kdy roste pravděpodobnost vzniklých chyb. Jiné nastavení tolerované vzdálenosti vzhledem k délce textového řetězce je nutné nastavit v prostředí, ve kterém je dotaz sestavován, přímo v rámci Elasticsearch to není možné. Konečně je také definován analyzer, kterým je hledaný výraz zpracován, aby probíhalo porovnání na řetězci, které prošly shodnou transformací.

Výpis 6.8: Vyhledávání s překlady v Elasticsearch

```
{
  "query": {
    "match": {
      "message": {
        "query": "hledXný výraz",
        "fuzziness": 1,
        "analyzer": "hunspell"
      }
    }
  }
}
```

Pro aktivaci funkce našeptávání (vyhledávání nedopsaných výrazů) lze využít `prefix_query`. Zde je předpokládáno, že poslední zadané slovo není uvedeno kompletní a je vyhledáváno dle jeho prefixu. Takový výraz může být zapsán jako **hledaný výr**, přičemž dotaz je chápán jako **hledaný výr***, kde hvězdička značí libovolnou sekvenci znaků.

Výpis 6.9: Našeptávání v Elasticsearch

```
{
  "query": {
    "prefix": {
      "name.hunspell": "hledaný výr"
    }
  }
}
```

Celý dotaz je obaloen do `bool query`, což umožní spojit výše uvedené dílčí dotazy pomocí logických spojek. Dotazy jsou vloženy do bloku `should`, dále je nastaven paramter `minimum_number_should_match` na hodnotu rovnou jedné, což znamená, že dokument odpovídá hledanému dotazu, pokud je nalezen pomocí alespoň jednoho dílčího dotazu.

Výpis 6.10: Složení dotazů pro Elasticsearch

```
{
  "query": {
    "bool" : {
      "should": [
        {"multi_match": {"..."}}},
        {"match": {"fuzziness" : 1, "..."}},
        {"prefix": {"..."}}
      ],
      "name.minimum_number_should_match": 1
    }
  }
}
```

Vzhledem k tomu, že v jednom indexu jsou ukládány data všech uživatelů, je třeba vyfiltrovat jen produkty daného uživatele. Každý produkt má definované ID uživatele, kterému náleží. Zde jde o přesnou shodu, takže je vyhledáváno (nebo spíše filtrováno) pomocí klauzule **term**, která nepracuje s analyzátory. Tento dotaz je vložen do booleovského dotazu do sekce **filter**. Bylo by sice možné použít sekci **must**, nicméně pro případ, kdy probíhá filtrace je prvně jmenovaná sekce lepší. Její výsledky totiž mohou být interně ukládány do mezipaměti, filtrace tak bude rychlejší.

Výpis 6.11: Filtrování dle ID uživatele v Elasticsearch

```
{
  "query": {
    "bool" : {
      "...",
      "filter": [
        {"term": {"userId": "aKi2mc7mckID0hyMsk9oQ"}}
      ]
    }
  }
}
```

6.3 Výběr nástroje pro backend a frontend

Dále je třeba rozhodnout, v čem bude psán backend, pomocí jakého programovacího jazyka nebo frameworku. Možností je celá řada, v úvahu přichází jak využití dynamických skriptovacích jazyků běžně využívaných pro web (nejpoužívanější PHP nebo JavaScript), přes jazyky s komplexnějším prostředím využívající k běhu virtuální stroj (Java, .NET), až po statické překládané jazyky (C, C++, GoLang). Je třeba zvolit takový jazyk, který bude rozumně použitelný, bude k němu dostatek knihoven pro tvorbu API a práci s Elasticsearch, aby netrvalo samotná implementace neúměrně dlouho. Na druhou stranu je však třeba, aby daný jazyk umožňoval vytvoření serveru, který bude schopný vyřizovat požadavky co nejrychleji.

Vzhledem k požadované rychlosti nepřichází v úvahu skriptované jazyky, které sice mohou poskytovat rozumnou dobu odezvy, pokud je ale možnost ji snížit pomocí jiného jazyku, není důvod toho nevyužít. Volba padla na jazyk GoLang, který je staticky typovaný, překládaný, takže výstupem je binární spustitelný soubor. Disponuje knihovnami pro implementaci API, HTTP serveru i komunikace s Elasticsearch. Nebrzdí také případné další rozšiřování, je možné s jeho pomocí implementovat podporu dalších protokolů jako WebSockets nebo gRPC. Zároveň však téměř odstraňuje nevýhody jazyků vyžadující překlad. Samotný překlad je velmi rychlý, v řádu jednotek sekund, což je ve srovnání s ostatními jmenovanými jazyky výrazně rychlejší. Další výhodou je, že po překladu vznikne jeden binární soubor, který nevyžaduje žádné běhové prostředí a není závislý na jiných knihovnách, což výrazně zkrátí dobu sestavení i nasazení.

6.4 Backend

Backend je napsán v jazyce Go, který vznikl ve společnosti Google a měl přinést některé výhody dynamicky typovaných jazyků do jazyků staticky typovaných. Přináší zejména stručný zápis některých konstrukcí, snadný a rychlý překlad. Součástí distribuce SDK je také balíčkovací manažer, nástroje pro formátování zdrojových kódů, statickou analýzu nebo dokumentaci.

Pro zpracování spuštění programu z příkazové řádky je použita knihovna `urfave/cli`, která umožňuje definovat jednotlivé argumenty, s nimiž může být aplikace spuštěna. Obsahuje také sadu základních příkazů jako je zobrazení nápovědy nebo verze programu. V této aplikaci jsou definovány celkem tři příkazy – spuštění serveru, spuštění asynchroního importu produktů a vytvoření indexu v Elasticsearch.

Vytvoření indexu je možné spuštěním přeloženého programu s argumentem `createIndex`. Po spuštění je spuštěna funkce `CreateIndex` v balíčku `elastic`. Ta vytvoří indexy v Elasticsearch (pokud neexistují) a následně u nich aktualizuje mapování. Lze tak tento příkaz použít pro přidávání nových polí i v produkčním prostředí.

Druhým příkazem je spuštění importu produktů po spuštění programu s argumentem `import`. To je vykonáno spuštěním funkce `ImportXmlFiles` v balíčku `importer`. Ta načte veškeré konfigurace uživatelů a pro každou z nich spustí im-

port produktů prostřednictvím funkce `ImportXmlFile`. Pro samotnou komunikaci s Elasticsearch je použita knihovna `olivere/elastic`. Import produktů je implementován tak, že se nejprve načte zdrojový XML soubor, který se namapuje na pole produktů pomocí interní prostřednictvím vestavené funkce `Unmarshal` [10]. Produkt je zde struktura, která je definována následujícím způsobem:

Výpis 6.12: *Struktura produktu*

```
type Product struct {
    Id          string `xml:"ITEM_ID" json:"id"`
    UserId      string `xml:"- " json:"userId"`
    Name        string `xml:"PRODUCTNAME" json:"name"`
    Description string `xml:"DESCRIPTION" json:"description"`
    Url         string `xml:"URL" json:"url"`
    Img         string `xml:"IMGURL" json:"img"`
    Price       int    `xml:"PRICE_VAT" json:"price"`
    Updated     string `xml:"- " json:"updated"`
}
```

Součástí definice struktury není pouze výpis jejích prvků s odpovídajícími datovými typy, ale také nastavení pro převod struktury do textového řetězce ve formátu XML a JSON. Každý produkt který vznikl převodem z XML je následně převeden s pomocí interní funkce `Marshall` na textový řetězec ve formátu JSON. To už je formát, ve kterém může být produkt ukládán do Elasticsearch. Ukládání produktů je provedeno pomocí hromadné indexace.

Výpis 6.13: *Hromadná indexace produktů*

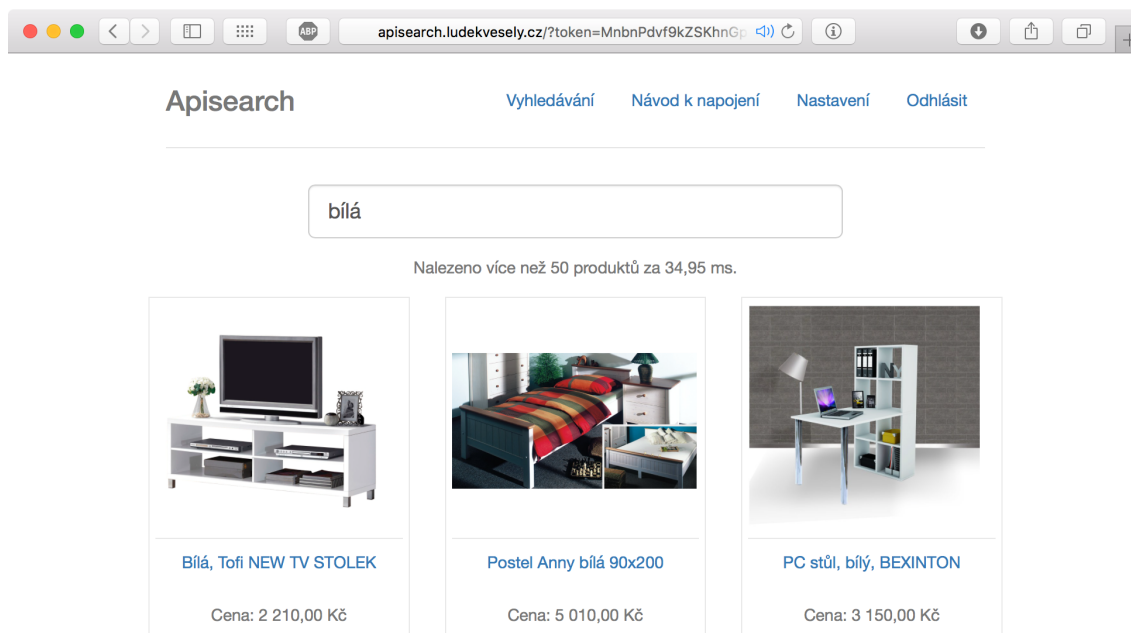
```
// Vytvoření procesoru pro hromadný import
bulk, err := products.BulkStart()
// Indexace každého produktu
for i, _ := range productList.ProductList {
    productList.ProductList[i].UserId = s.UserId
    productList.ProductList[i].Updated =
    ↪ time.Now().Format("2006-01-02T15:04:05-07:00")
    productList.ProductList[i].BulkIndex(bulk)
}
// Dokončení indexace - uložení zbývajících produktů a zavření
↪ procesoru
err = products.BulkFlush(bulk)
```

Ta je na pozadí implementována tak, že ukládání probíhá paralelně a navíc Elasticsearch provádí samotnou indexaci v dávkách. Díky tomu je import extrémně rychlý a přitom není třeba se starat o spouštění dalších procesů – to provádí knihovna s pomocí příkazu `go` [9].

Poslední možností jak přeložený program spustit je nastartování HTTP serveru. Server je použit z standardní knihovny `net/http`. Pro routování příchozích požadavků na odpovídající funkce na základě použité URL je použita knihovna `gorilla/handlers`. Každý možný tvar požadavku je definován jako struktura:

Výpis 6.14: *Routování HTTP serveru*

```
route := Route{
    "Search",           // Nazev
    "GET",              // HTTP metoda
    "/api/v1/search/{userId}/{query}", // Vzor URL
    v1.Search,          // Obslužná funkce
    []string{},         // Validace argumentů URL
}
```



Obrázek 6.1: Frontend – kontrola funkce vyhledávání

Toto je konkrétní zápis definice koncového bodu pro vyhledávání v produktech daného uživatele. Požadavek poté obsloužen funkcí, která má v tomto případě předpis `func Search(w http.ResponseWriter, r *http.Request) {}`. Může tedy číst veškeré parametry s nimiž byla spuštěna (`*http.Request`) a také má kam zapsat výstup, ať už se jedná o odpověď ve formátu JSON nebo odpovídající stavové kódy (`http.ResponseWriter`). Uvnitř této funkce je provedeno samotné vyhledávání pomocí funkce `products.Search`, která vrací pole produktů. Ty jsou následně transformovány do formátu JSON a vráceny jako odpověď serveru.

6.5 Frontend

Frontend je grafickým rozhraním pro komunikaci s backendem. Sám o sobě je bezstavový, je pouze grafickou obálkou nad API. Implementován je v jazyce PHP a frameworku Nette, což umožňuje rychlý vývoj aplikace při dodržení dobrých programátorských návyků. Využívá vzoru MVC pro oddělení logiky a zobrazení a Dependency Injection pro předávání závislostí. Pro napojení na API využívá knihovnu `GuzzleHttp`, což je jediná externí závislost projektu.

Po registraci do aplikace je na pozadí spuštěn import produktů. Již v průběhu import však může uživatel testovat funkčnost vyhledávání, aby si jej mohl ověřit ještě před samotným napojením na obchod. Zároveň je zobrazována rychlost odezvy, aby uživatel viděl, jak dlouho reálně každý požadavek trvá.

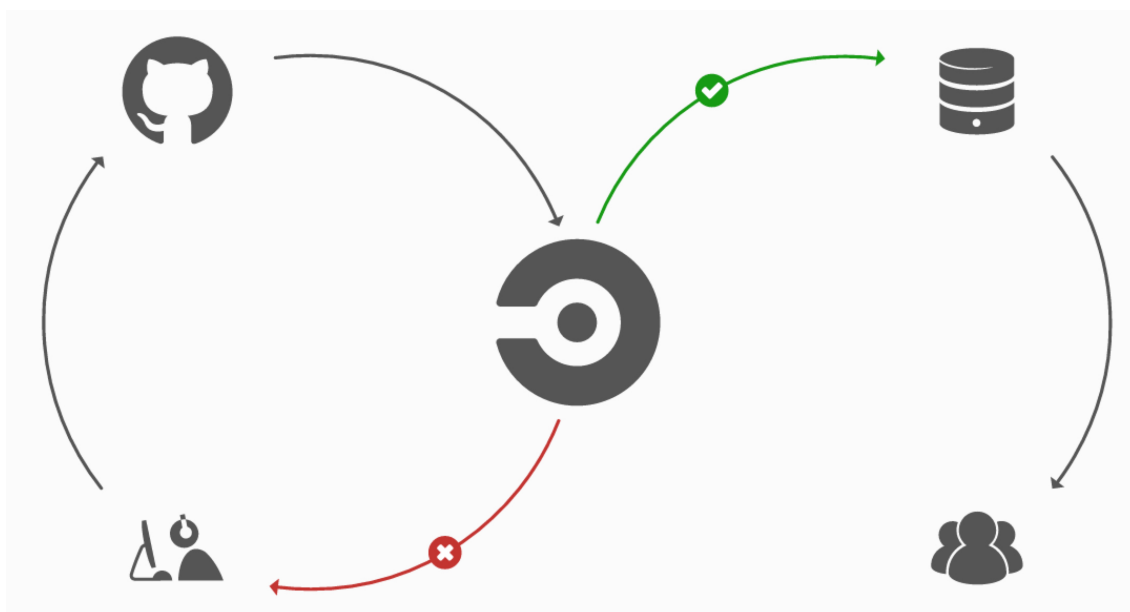
Dále je součástí této aplikace návod na napojení elektronického obchodu. To je popsáno jednak principiálně (s odkazem na dokumentaci v Apiary), tak je k dispozici příklad implementace, který je možné rovnou zkopírovat do webu obchodu. Příklady

jsou pro nejpoužívanější jazyky z prostředí webu (PHP a JavaScript), přičemž jsou navrženy tak, aby nebylo třeba instalovat další závislosti a napojení tak bylo co nejjednodušší.

Poslední možností, kterou aplikace nabízí je úprava nastavení – uživatel zde může změnit přihlašovací údaje (e-mail nebo heslo) a adresu XML souboru s produkty. Z aplikace se může uživatel odhlásit, veškeré jeho nastavení je dostupné pouze po přihlášení, což je nutné pro možnost využívání aplikace více uživateli (elektronickými obchody).

6.6 Deployment

Posledním krokem k zveřejnění aplikace a jejímu zpřístupnění veřejnosti je její nasazení do produkčního prostředí. Pro vytvoření nasaditelného artefaktu je použit nástroj Docker, obdobně jako při distribuci Elasticsearch. Aby nemusel vývojář po každé změně zdrojových kódů znovu ručně vytvářet nasazovat novou verzi aplikace (což je zdlouhavé a navíc může vést k chybám [12, strana 23]), je vzdálený GIT repozitář propojen s integračním serverem, který provede sestavení a nastavení aplikace automaticky. Konkrétně je používán CircleCI [47].



Obrázek 6.2: Kontinuální nasazování aplikace pomocí CircleCI

Pomocí CircleCI je nasazován jak frontend, tak backend. Vývojář nahraje provedené změny v lokálním repozitáři do repozitáře vzdáleného (GitHub), ten poté odešle požadavek do napojeného integračního serveru (CircleCI) s informací o repozitáři, který byl aktualizován. V rámci CircleCI je spuštěn nový virtuální stroj, který stáhne aktuální verzi zdrojových kódů z vzdáleného repozitáře a dále postupuje podle příkazů definovaných v souboru `circle.yml`. V něm je definováno jak mají být získány závislosti aplikace (příkazem `go get` v případě backendu) a jak má

být aplikace sestavena (`go build` v případě backendu). Vytvořený binární soubor je poté nakopírován do prázdného Docker obrazu, který je poté spuštěn příkazem `docker run`. Tím je ověřeno, že byl obraz sestaven v pořádku a je nahrán do registru příkazem `docker push`. Nakonec je spuštěno samotné nasazení, které je provedeno pomocí nástroje `docker-compose` na vzdáleném serveru.

Všechny kontejnery jsou spravovány prostřednictvím nástroje Docker Compose [48]. Ten usnadňuje práci s Docker démonem – používá vlastní sadu příkazů a veškerá konfigurace se provádí pomocí YAML souboru. Díky tomu, že jde o deklarativní přístup, je možné příkazy opakovat a vždy je dosaženo stejného očekávaného výsledku. Tento konfigurační soubor je pojmenován `docker-compose.yml` a má pro tuto aplikaci následující podobu:

Výpis 6.15: Konfigurace pro Docker Compose

```
web-admin:
  image: apisearch/web-admin
  ports:
    - "7010:8080"
  links:
    - apisearch
  restart: always
apisearch:
  image: apisearch/apisearch
  command: server
  ports:
    - "7011:8080"
  links:
    - elasticsearch
  restart: always
elasticsearch:
  image: apisearch/elasticsearch
  volumes:
    - /mnt/apisearch/elasticsearch:/usr/share/elasticsearch/data
  restart: always
```

Tento soubor definuje takzvaný *stack*, což je skupina několika souvisejících služeb. V tomto případě jsou definovány tři služby (**web-admin** – frontend, **apisearch** – backend a **elasticsearch**). U všech služeb je nastaveno, aby se v případě chyby automaticky restartovaly. Dále je u prvních dvou služeb definován port, přes který jsou dostupné. Na serveru je pak nastavena proxy, která na základě celé URL směruje příchozí požadavky na jednu z těchto služeb. U služby **elasticsearch** je v **volumes** definována složka, do které jsou persistována data. Po restratu kontejneru totiž dochází ke ztrátě jeho dat, což by byl problém vzhledem k stavovosti služby. Je tedy zvolena složka v hostitelském stroji, která je použita pro persistenci těchto dat. Celou skupinu služeb je možné spustit jediným příkazem **docker-compose up -d**. Zároveň je možné pomocí tohoto nástroje sledovat logy všech běžících služeb, jejich stav a případně je aktualizovat nebo zastavovat.

Použití tohoto nástroje má výhodu v tom, že vzhledem k tomu, že jej oficiálně vydává Docker, v dobré podpoře a kompatibilitě s ostatními nástroji. V případě nutnosti horizontálního škálování lze využít některý z orchestrátorů [12, strana 300] (například Docker Swarm, Docker Cloud nebo Rancher), které jsou s tímto formátem zápisu kompatibilní.

7 Testování, ověření

Tyhle body dopsat i do analýzy jako požadavky v tabulce:

FUNKCNOST: - Otestovat, že to umí to co má (fulltext case insensitive, preklepy, naseptávání, sklonování, synonyma...)

- tohle kdyžtak dodat aby bylo přesně do požadavku
- udělat CLI nástroj co udělá request a vypíše na CLI název produktu odpovídajících
- udělat testovací množinu dat (pro testy výše) a popsat ji
- -> výpisy CLI nebo screeny GUI z testu vyhledávání

RYCHLOST:

- nasypat >100k produktů a prohnat asi nástrojem ab, otestovat dobu response
- -> výpisy, grafy...

SNADNOST IMPLEMENTACE

- popsat implementaci na velingu, kroky co je potřeba udělat, případně odhad doby implementace

8 Závěr

V této diplomové práci jsem...

8.1 Dosažení vytčených dílů

...

8.2 Diskuze možného budoucího rozšiřování

...

A Obsah přiloženého CD

- Soubor Diplomova_prace_2017_Ludek_Vesely.pdf
 - Text diplomové práce

B Terminologický slovník

Termín	Zkratka	Význam
Uniform Resource Locator	URL	Adresa dokumentu na webu
Extensible Markup Language	XML	Značkovací jazyk
European Article Number	EAN	Mezinárodní číselná identifikace výrobků
International Standard Book Number	ISBN	Číselná identifikace knižních vydání
Structured Query Language	SQL	Jazyk určený pro dotazování v relačních databázích
Hypertext Transfer Protocol	HTTP	Protokol pro přenos souborů po síti internet
Application Programming Interface	API	Rozhraní pro programování aplikací
Representational State Transfer	REST	Architektura rozhraní pro komunikaci mezi počítačovými systémy
Simple Object Access Protocol	SOAP	Protokol pro výměnu zpráv přes síť
Formální jazyk		
Počítačová lingvistika		
Morfologie		
Deployment		
Frontend		
Backend		

Literatura

- [1] SUCHÁNEK Petr. *E-commerce*. Vyd. 1. Praha: Ekopress, s.r.o., 2012 144 s. ISBN 978-80-86929-84-2.
- [2] STROSSA, Petr. *Počítačové zpracování přirozeného jazyka*. Vyd. 1. Praha: Oeconomica, 2011 316 s. ISBN 978-80-245-1777-3.
- [3] AYSE Göker a DAVIES John. *Information Retrieval: Searching in the 21st Century*. Vyd. 1. Library of Congress Cataloging-in-Publication Data, 2009 295 s. ISBN: 978-0-470-02762-2.
- [4] AGGARWAL Charu C., ZHAI ChengXiang. *Mining Text Data*. Springer New York Dordrecht Heidelberg London, 2000 522 s. ISBN 978-1-4614-3222-7.
- [5] CLINTON Gormley, ZACHARY Tong. *Elasticsearch: The Definitive Guide*. O'Reilly Media, 2015 724 s. ISBN 978-1-4493-5854-9.
- [6] Elasticsearch. *Elasticsearch Reference* [online]. 2017 [cit. 2017-03-03]. Dostupné z: <https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html>
- [7] STURGEON Phil. *Build APIs You Won't Hate*. Vyd. 1. Philip J. Sturgeon, 2015 188 s. ISBN 978-0692232699.
- [8] KENNEDY William. *Go in Action*. Manning Publications Co, 2016 264 s. ISBN 978-1-6172-9178-4.
- [9] KOZYRA Nathan. *Mastering Concurrency in Go*. Packt Publishing, 2014 301 s. ISBN-13 978-1783983483.
- [10] ASTAXIE. *Build web applications with Golang* [online]. 2017 [cit. 2017-04-14]. Dostupné z: <https://astaxie.gitbooks.io/build-web-application-with-golang/en/07.1.html>.
- [11] MATTHIAS Karl, KANE P. Sean. *Docker: Up & Running*. O'Reilly Media, 2015 232 s. ISBN 978-1-4919-1754-1.
- [12] FARCIC Viktor. *The DevOps 2.0 Toolkit: Automating the Continuous Deployment Pipeline with Containerized Microservices*. Packt Publishing, 2016 462 s. ISBN:978-1-78528-031-3.

- [13] Apiary. *Platform for API Design, Development & Documentation* [online]. 2017 [cit. 2017-03-11]. Dostupné z: <https://apiary.io>.
- [14] Elastic. *Elasticsearch: RESTful, Distributed Search & Analytics* [online]. 2017 [cit. 2017-04-12]. Dostupné z: <https://www.elastic.co/products/elasticsearch>
- [15] Algolia. *Hosted cloud search as a service* [online]. 2017 [cit. 2017-03-03]. Dostupné z: <https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html>
- [16] Swiftype. *Site Search by Swiftype* [online]. 2017 [cit. 2017-03-30]. Dostupné z: <https://swiftype.com/site-search>
- [17] AWS. *Amazon CloudSearch* [online]. 2017 [cit. 2017-03-30]. Dostupné z: <https://aws.amazon.com/cloudsearch>
- [18] Google. *Custom Search Engine* [online]. 2017 [cit. 2017-03-30]. Dostupné z: <https://cse.google.com/cse>
- [19] GigaSpaces Blog. *Amazon found every 100ms of latency cost them 1% in sales* [online]. 2017 [cit. 2017-03-26]. Dostupné z: <https://blog.gigaspaces.com/amazon-found-every-100ms-of-latency-cost-them-1-in-sales>
- [20] StackShare, Inc. *How Algolia Reduces Latency For 21B Searches Per Month* [online]. 2017 [cit. 2017-04-05]. Dostupné z: <https://stackshare.io/algolia/how-algolia-reduces-latency-for-21b-searches-per-month>
- [21] PostgreSQL: Documentation. *Full Text Search* [online]. 2017 [cit. 2017-03-26]. Dostupné z: <https://www.postgresql.org/docs/9.5/static/textsearch.html>
- [22] The Apache Software Foundation. *Apache Lucene* [online]. 2017 [cit. 2017-03-26]. Dostupné z: <https://lucene.apache.org>
- [23] Sdružení pro internetový rozvoj. *NetMonitor* [online]. 2017 [cit. 2017-04-01]. Dostupné z: <http://www.netmonitor.cz>
- [24] Heureka Shopping s.r.o. *Specifikace XML souboru* [online]. 2017 [cit. 2017-04-01]. Dostupné z: <https://sluzby.heureka.cz/napoveda/xml-feed>
- [25] Seznam.cz, a.s. *Specifikace XML feedu pro internetové obchody* [online]. 2017 [cit. 2017-04-01]. Dostupné z: <https://napoveda.seznam.cz/cz/zbozi/specifikace-xml-pro-obchody/specifikace-xml-feedu>
- [26] TĚŠITELOVÁ Marie. *O češtině v číslech*. Praha: Academia, 1987 205 s.

- [27] Elasticsearch. *The Definitive Guide - Fuzziness* [online]. 2017 [cit. 2017-04-08]. Dostupné z: <https://www.elastic.co/guide/en/elasticsearch/guide/current/fuzziness.html>.
- [28] DAMERAU, Fred J. *A Technique for Computer Detection and Correction of Spelling Errors*. Communications of the ACM (ACM), 1964, s. 171–176. ISSN 0001-0782. doi: 10.1145/363958.363994. Dostupné z: <http://doi.acm.org/10.1145/363958.363994>
- [29] SMETANIN Nikita. *Fuzzy string search* [online]. 2017 [cit. 2017-04-08]. Dostupné z: <http://ntz-develop.blogspot.cz/2011/03/fuzzy-string-search.html>.
- [30] CHRISTEN, Peter. *A Comparison of Personal Name Matching: Techniques and Practical Issues*. ICDMW '06 Proceedings of the Sixth IEEE International Conference on Data Mining, 2006, s. 290-294, IEEE Computer Society Washington, DC, USA. doi: 10.1145/363958.363994. Dostupné z: <http://users.cecs.anu.edu.au/~Peter.Christen/publications/tr-cs-06-02.pdf>. ISBN: 0-7695-2702-7.
- [31] HUS Maarten. *The case for separating front- and back-end* [online]. 2014 [cit. 2017-04-11]. Dostupné z: <http://dontpanic.42.nl/2014/10/the-case-for-separating-front-and-back.html>.
- [32] Apiary. *How to Build an API* [online]. 2017 [cit. 2017-04-11]. Dostupné z: <https://apiary.io/how-to-build-api>.
- [33] IBM. *How to explain vertical and horizontal scaling in the cloud* [online]. 2014 [cit. 2017-04-11]. Dostupné z: <https://www.ibm.com/blogs/cloud-computing/2014/04/explain-vertical-horizontal-scaling-cloud>.
- [34] API Blueprint. *A powerful high-level API description language for web APIs* [online]. 2017 [cit. 2017-04-12]. Dostupné z: <https://apiblueprint.org>.
- [35] Mozilla Developer Network. *WebSockets – Web APIs* [online]. 2017 [cit. 2017-04-12]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API.
- [36] The PostgreSQL Global Development Group. *Documentation: Chapter 12. Full Text Search* [online]. 2017 [cit. 2017-04-12]. Dostupné z: <https://www.postgresql.org/docs/9.1/static/textsearch-intro.html>.
- [37] Elastic. *Elasticsearch Reference: Analysis* [online]. 2017 [cit. 2017-04-12]. Dostupné z: <https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis.html>.
- [38] ROWE Nathaniel. *Performance Testing a Postgres Database vs Elasticsearch 5: Column Statistics* [online]. 2017 [cit. 2017-04-12]. Dostupné z: <http://blog.nrowegt.com/database-vs-elasticsearch-speed-column-statistics>.

- [39] ROWE Nathaniel. *How To Use Sub Aggregations With Searchkick To Return Multiple Terms Per Document* [online]. 2017 [cit. 2017-04-12]. Dostupné z: <http://blog.nrowegt.com/how-to-use-sub-aggregations-with-searchkick-to-return-multiple-terms-per-document>.
- [40] The Apache Software Foundation. *Apache Solr* [online]. 2017 [cit. 2017-04-12]. Dostupné z: <http://lucene.apache.org/solr>.
- [41] YIGAL Asaf. *Solr vs. Elasticsearch: Who's The Leading Open Source Search Engine?* [online]. 2016 [cit. 2017-04-12]. Dostupné z: <https://logz.io/blog/solr-vs-elasticsearch>.
- [42] Sphinx Technologies Inc. *Sphinx Search* [online]. 2017 [cit. 2017-04-12]. Dostupné z: <http://sphinxsearch.com>.
- [43] NÉMETH László. *Hunspell* [online]. 2017 [cit. 2017-04-13]. Dostupné z: <http://hunspell.github.io>.
- [44] Canonical Ltd. *Ubuntu Manpage: hunspell - format of Hunspell dictionaries and affix files* [online]. 2017 [cit. 2017-04-13]. Dostupné z: <http://manpages.ubuntu.com/manpages/trusty/en/man4/hunspell.4.html>.
- [45] Alpine Linux Development Team. *Alpine Linux* [online]. 2017 [cit. 2017-04-13]. Dostupné z: <https://alpinelinux.org>.
- [46] WALLIN Marko. *Lean Docker Containers with Alpine Linux* [online]. 2017 [cit. 2017-04-13]. Dostupné z: <https://gofore.com/en/lean-docker-alpine-linux>.
- [47] CircleCI. *Continuous Integration and Delivery* [online]. 2017 [cit. 2017-04-14]. Dostupné z: <https://circleci.com>.
- [48] DOCKER. *Compose: Define and run multi-container applications with Docker* [online]. 2017 [cit. 2017-04-14]. Dostupné z: <https://github.com/docker/compose>.