Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informačních technologií

Optimalizace fulltextového vyhledávání

Bakalářská práce

Autor: Lukáš Kratochvíl

Studijní obor: AI-3

Vedoucí práce: Mgr. Jan Vaněk, Ph.D.

Hradec Králové duben 2018

Poděkování:

Chtěl bych poděkovat Mgr. Janu Vaňkovi, Ph.D. za odborné vedení mé bakalářské práce a cenné rady, které mi pomohly tuto práci zkompletovat.

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne   Lukáš Kratochvíl

Anotace

Práce se zabývá problematikou optimalizace fulltextového vyhledávání za využití technologie k tomu určené. Mimo optimalizaci dané technologie je zde ale také popsáno, jak se fulltextově vyhledávalo v minulosti a jaké jiné přístupy se využívají dnes. Je zde detailněji specifikována a rozebrána knihovna Apache Lucene, kterou jako svůj základ používá většina dnešních předních vyhledávacích technologií. Tyto technologie jsou zde popsány a je provedeno jejich porovnání, na základě kterého je vybrána jedna konkrétní technologie Elasticsearch, jejímž směrem se práce orientuje a na které jsou otestovány všechny navržené způsoby optimalizace, které se dají využít pro optimalizaci fulltextového vyhledávání širokého spektra aplikací. Kromě jednotlivých metod optimalizace si práce klade za cíl více přiblížit celkový proces zpracování dokumentu v moderních technologiích. Přes předpřípravu dokumentu, jeho zaindexování, až po následné fulltextové vyhledávání v něm. Na základě obsahu práce by měl být každý, kdo se chce informovat o současných technologiích v oblasti fulltextového vyhledávání nebo má v plánu do svého projektu použít některý fulltextový vyhledávač, schopen vybrat tu nejlepší technologii a měl by být schopen ji implementovat a optimalizovat pro svůj projekt.

Annotation

The paper deals with optimization of fulltext search using technology designed for this purpose. Besides the optimization of the technology, it is also described how fulltext search was provided in the past and what approaches are being used today. The Apache Lucene library, which is used by many today's technologies, is specified in greater detail. These technologies are described in the paper and their comparison is made. On the basis of the comparison, Elasticsearch technology is selected and is the focus of the rest of the paper. On this technology a number of proposed optimization methods are being tested, which can be used to optimize fulltext search in a wide range of applications. In addition to individual optimization methods, the paper deals with the overall process of document processing in modern technologies. It includes the principals of the preparation of a document, its indexing and fulltext search in it. Based on the content of the work, anyone who wants to know about current technologies in the field of fulltext searching or intends to use a fulltext search engine capable of selecting the best technologies and being able to implement and optimize for their project.

Obsah

1 ÚVOD 1

2 VYHLEDÁVÁNÍ 2

2.1 Fulltextové vyhledávání 2

2.2 Vyhledávací algoritmy 3

2.3 Moderní vyhledávací nástroje 5

3 TECHNOLOGIE PRO FULLTEXTOVÉ VYHLEDÁVÁNÍ 9

3.1 Apache Lucene 9

3.2 Elasticsearch 12

4 CÍL PRÁCE 20

5 VOLBA TESTOVACÍ METODIKY 21

6 NÁVRH ŘEŠENÍ 22

6.1 Implementace 22

6.2 Testovací aplikace 23

6.3 Úvod do základního nastavení 24

6.4 Optimalizace 27

7 VÝSLEDKY 38

7.1 Kategorie dotazů 38

7.2 Použité optimalizace 40

7.3 Výsledky navržených optimalizačních metod 41

8 ZÁVĚR 45

9 ZDROJE: 46

1 Úvod

Historicky prvním internetovým vyhledávačem, tak jak je známe dnes, se stal v dubnu roku 1994 vyhledávač WebCrawler, který na rozdíl od svých předchůdců dokázal indexovat celý obsah stránky a ne jen jejich hlavičky. Od této doby uplynulo 24 let a za tuto dobu se samozřejmě řada věcí změnila. S masivním nárůstem počítačů rostl také počet internetových stránek a celkově dat na internetu. Proto bylo a stále je za potřebí vyvíjet novější a lepší technologie, které tato data dokáží zpracovat. Podstata celé myšlenky internetových vyhledávačů a fulltextového vyhledávání obecně zůstává ale již od roku 1994 stále stejná, a sice indexovat obsah stránky, a tím tak umožnit jeho zpětné vyhledání. V dnešní době bychom si jen těžko dokázali představit prohledávání internetu stránku po stránce pro nalezení požadované informace bez použití některého z vyhledávačů. Nemusí to být ale hledání informací na celém internetu. Velice složitou operací by dnes bylo i prohledávání některé větší webové aplikace, a právě směrem k vyhledávačům pracujícím v rámci webové aplikace se tato práce ubírá.

V dnešní době již existuje, vedle modernějších nástupců WebCrawleru, jako je Google search, Yahoo, Bing a dalších, také celá řada technologií, jejichž fungování je možné přizpůsobit právě pro jednu webovou aplikaci. V práci je řada těchto technologií zmíněna, řada z nich je také porovnána a na základě tohoto porovnání je zde vysvětleno, proč to byla právě technologie Elasticsearch, která byla zvolena jako nejlepší.

Spíše než výběr technologie je ale v práci zajímavější přístup k optimalizaci. Ta není chápána z hlediska výkonosti ani paměťové náročnosti, jelikož tyto aspekty má technologie Elasticsearch dobře zvládnuté, ale spíše z hlediska návratnosti správných výsledků. Hlavním cílem práce je tedy navrhnout takové obecné optimalizační metody, které bude možné použít v širším spektru aplikací a které po jejich implementaci povedou k navrácení co možná nejrelevantnější výsledkové sady.

Optimalizační metody jsou v práci také otestovány, zda jsou ze zmíněného hlediska přínosné či nikoliv.

2 Vyhledávání

Na internetu existuje ohromné množství stránek a jejich počet se neustále zvyšuje. Podle [1] počet stránek na internetu od dob svého vzniku exponenciálně roste. Z čehož plyne, že každým dnem roste rozdíl mezi nově vzniklými webovými stránkami a těmi již nějakou dobu fungujícími, starými. O sběr dat z nových stránek, aktualizaci dat z upravených starších stránek a umožnění v nich vyhledat požadovanou informaci se starají fulltextové vyhledávače.

Fulltextové vyhledávače jsou zodpovědné za odpověď na položený dotaz a jedním z cílů práce je objasnit proces vyhledávání se zaměřením na vnitřní fungování vyhledávače. Jinými slovy, popsat vnitřní komplexní systém procesu, který je zodpovědný za odpověď na položený dotaz.

2.1 Fulltextové vyhledávání

Vyhledávače, které jsou označeny jako fulltextové, jsou vyhledávače fungující na základě porovnávání fráze se všemi ostatními slovy v daném dokumentu. Je známo více druhů vyhledávačů. Na jedné straně jsou to vyhledávače pracující přímo s textem. Neprobíhá tedy žádná předpříprava porovnávaného dokumentu, ale daná fráze (fráze, kterou chceme vyhledat) je v dokumentu hledána přímo za pomoci některého z vyhledávacích algoritmů, které jsou v práci zmíněny v kapitole “Vyhledávací algoritmy” nebo kterýmkoli jiným algoritmem pracujícím přímo nad textem. Na straně druhé existují algoritmy, konkrétněji souhrn algoritmů a datových struktur, které dokáží z daného dokumentu vytvořit podrobnou statistiku výskytu jednotlivých pojmů a tu uložit do vlastní databáze (říkáme, že dokument naindexují). Při vyhledávání je porovnána zadaná fráze s dokumenty v databázi. Vrácený výsledek je závislý na několika faktorech. Složitostí vyhledávacích algoritmů, schopností relevance, scoringu, atd. Existuje celá řada vyhledávacích algoritmů.

Současné fulltextové vyhledávače mají algoritmy natolik propracované, že dokáží sledovat nejen počet slov v textu, ale i jejich hustotu, sémantiku, umístění v nadpisech, titulcích, apod. Běžná je také schopnost skloňování a časování.

Fulltextové vyhledávání, zkráceně fulltext, je možné si spojit s vyhledávači prohledávajícími celý internet (např. vyhledávač společnosti Google) ale také s vyhledávači fungujícími v rámci jedné webové aplikace. Aplikací může být například rozsáhlá dokumentace nebo e-shop, kde fulltextový vyhledávač bývá vyžadován a s rostoucí velikostí aplikace se stává nezbytným, jelikož prosté prohledávání stránky po stránce k nalezení informace je časově velice náročné.

Internetové fulltexty bývají značně složitější, neboť zpracovávají větší množství dokumentů. Pro představu, společnost Google a jejich produkt Google Search, který je od roku 2009 podle “comScore” (Americká společnost, která poskytuje měření a analýzu médií) nejvíce využívaným vyhledávacím enginem ve spojených státech, indexuje biliony webových stránek pro umožnění uživatelům získat z nich požadovanou informaci skrze zadání klíčových slov, viz [2][3].

Elasticsearch i další frameworky, které jsou v této práci zmíněny, patří mezi jednodušší vyhledávače fungující nad výrazně menším objemem dat, převážně v rámci jedné webové aplikace.

2.2 Vyhledávací algoritmy

Jednoduše se jedná o algoritmy, jejichž cílem je porovnat námi zadanou frázi se všemi dokumenty a vrátit výsledek, který obsahuje shody. Algoritmů pro fulltextové vyhledávání existuje několik. Od nejjednoduššího naivního algoritmu až po např. daleko propracovanější a složitější fragmentační algoritmus, který používá vyhledávací knihovna Apache Lucene, nad kterou je vystavěna řada dnešních předních vyhledávacích technologií.

2.2.1 Naivní algoritmus

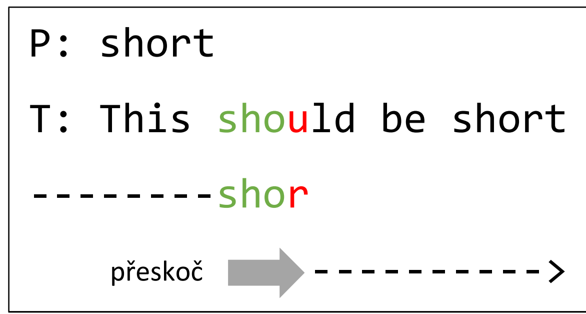
Při tomto způsobu se zkouší zadaná fráze přímo porovnat s dokumentem, ve kterém je fráze vyhledávána. Podstatou je cyklus, který běží od začátku dokumentu až do doby, kdy narazí na jeho konec. Cyklus začíná na prvním řádku a prvním znaku dokumentu a je porovnávána fráze se vzorem, který tvoří textový řetězec o délce fráze. Pokud v průběhu cyklu nastane shoda, je výsledek zaznamenán. V každém kole cyklu se vzor posune o jeden znak doprava.

Jedná se pravděpodobně o nejjednodušší algoritmus na porovnání textu. V algoritmu zcela chybí jakákoli předpříprava dokumentů, proto je jeho složitost rovna $O(m*n)$, kde m je délka fráze a n je délka textu v dokumentu. Při miniaturních projektech dokáže být při rychlostech dnešních počítačů efektivní. Jeho efektivita, ale značně klesá při větším a obsáhlejším množství dokumentů [4].

2.2.2 Boyer-Moore algoritmus

Tento algoritmus rozšiřuje myšlenku naivního algoritmu a zvyšuje jeho efektivitu. Stejně jako naivní algoritmus spočívá v porovnávání textu znak po znaku. Jeho složitost v nejhorším

případě je tedy stejná jako u Naivního algoritmu O(m*n), ale díky optimalizaci tato složitost může klesat. Pokud nalezne shodu prvního znaku ve frázi s textem, přejde na další znak a takto algoritmus pokračuje až do doby, kdy dorazí na konec fráze (v takovém případě se jedná o shodu) nebo do doby než narazí na neshodu. V takovém případě je porovnán znak v textu (ten který neseděl do sekvence znaků ve frázi) s kompletní frázi a pokud se ve frázi nevyskytuje, je zřejmé, že další porovnání je možné přeskočit a algoritmus začne běžet opět po daném nevyhovujícím znaku, viz Obr. 1.



Obr. 1: Boyer-Moore algoritmus

2.2.3 Rabin-Karp algoritmus

Jedná se o algoritmus, který používá hashovací funkci k nalezení shody v textu dokumentu. Hashovací funkce spočívá v převedení každého textového pole na číselnou hodnotu. Rabin-Karp algoritmus pak vychází z vlastnosti, že jsou-li slova stejná, pak mají i stejnou hash hodnotu.

Fungování tohoto algoritmu tedy spočívá v předpřípravě dokumentu, který je převeden na pole hash hodnot. Při vyhledávací fázi je pak i vyhledávaná fráze převedena na hash hodnotu a porovnávají se pouze tyto numerické hodnoty. Pro text, který má délku n a má p vzorků společné délky m je nejlepší složitost Rabin-Karpova algoritmu O(n+m) a nejhorší O(n*m).

2.2.4 Fragmentační algoritmy

Současné fulltexty využívají daleko důmyslnější prohledávání dokumentů. Ten spočívá v jeho předpřípravě, která např. u naivního nebo Boyer-Moorova algoritmu zcela absentovala. Předpřípravou je vytvořen takzvaný index, který lze přirovnat k tradiční relační databázi. Samotné hledání shod zadané fráze k vyhledání s dokumentem pak nemusí probíhat nad celým dokumentem, ale pouze nad jednotlivými částmi indexu. Moderní vyhledávače umožňují

vyhledávat jak přesné shody, tak dokáží pracovat se skloňováním, časováním pro jednotlivé jazyky, částečnou shodou, regulárními výrazy apod.

Jeden index může potenciálně uchovávat obrovské množství dat a může i překročit kapacitu disku nebo být z důvodu své velikosti příliš pomalý. Index je proto možné rozdělit do několika částí, tzv. fragmentů. Každý fragment je pak považován za vlastní index uvnitř indexu a dokáže plně manipulovat se svými daty stejně, jako by byl pouze jeden. Kromě manipulace s daty ale umí také spolupracovat a zprostředkovávat informace ostatním fragmentům.

Tento přístup umožňuje ukládat obsah na více místech. Také dovoluje distribuovat a paralelizovat operace napříč jednotlivými fragmenty, díky čemuž vyhledávač dosahuje vyššího výkonu a tím pádem i rychlosti. [5]

2.3 Moderní vyhledávací nástroje

V předchozích kapitolách byly přiblíženy jednotlivé vyhledávací algoritmy, které se i dnes v mnohých případech stále využívají. Moderní vyhledávací nástroje ale ke svému fungování využívají právě fragmentačního algoritmu a z toho důvodu se práce v dalších kapitolách věnuje pouze tomuto algoritmu.

Následující kapitoly popisují části fulltextového indexu a jednotlivé kroky a postupy při předzpracování dokumentů i následné vyhledávání v uložených dokumentech.

2.3.1 Dotazy

Jako na každý dotaz v běžném životě, i na dotaz v kontextu fulltextového vyhledávání je očekávána odpověď. Jakkoli se dokážou vyhledávací algoritmy postarat o vyhledání zadané fráze ve své databázi, tak výsledky nemusí být vždy optimální. Jako ukázka neoptimální odpovědi může vypadat odpověď typu “ANO, danému dotazu odpovídají tyto objekty” nebo “NE, tomuto dotazu neodpovídá žádný z dokumentů”. Naštěstí ale v moderním vyhledávání není nutné se omezit pouze na takto triviální způsob odpovědi. Dnešní vyhledávače jako návratovou hodnotu vrací celé objekty, obvykle sadu vyhovujících dokumentů, u nichž lze ovlivnit v jakém pořadí, například dle relevantnosti nalezeného objektu, budou seřazeny. Lze využít řadu optimalizačních kroků, které jsou v následujících kapitolách objasněny.

2.3.2 Optimalizace

Bakalářská práce vychází z předpokladu, že čím výše je správný výsledek ve výsledkové sadě, tím spíše je uživatelem nalezen. Účelem optimalizace je tedy zajištění toho, aby požadovaný výsledek ve výsledkové sadě, kterou vrátí fulltextový vyhledávač, byl co možná nejvýše, v ideálním případě na prvním místě. Toho lze docílit řadou optimalizačních kroků. Fragmentační algoritmy jsou postavené na propracované předpřípravné fázi. Práce s dokumenty je tak rozdělena na indexovací fázi a vyhledávací fázi, kde ani v jedné fázi není nutné se omezovat na základní nastavení, ale je možné chování přizpůsobit dané problematice.

2.3.2.1 Optimalizace indexovací fáze

V indexovací fázi probíhá proces indexování, který spočívá v uložení dokumentů do indexu (zaindexování dokumentů). Pouze mezi indexovanými dokumenty může být následně vyhledáváno.

Nejpodstatnějším krokem této fáze je tedy vypracování samotného modelu indexu. Do jakých polí budou data ukládána (kam bude uložen obsah stránky, kam doplňující informace, atd.). Model je v ideálním případě třeba připravit pro budoucí rozšíření, které může spočívat například v přípravě na multijazyčnost, přidávání dalších polí, která budou ovlivňovat relevantnost výsledků apod. Je tedy třeba dobře zvážit nejen požadavky, které aplikace v danou chvíli má, ale i požadavky, které mohou přijít. Každá aplikace je unikátní a není tedy možné definovat přesný, univerzální a zároveň optimální model pro všechny aplikace.

Indexování je, jak bylo zmíněno, proces ukládání dokumentů do databáze ve formátu, který umožňuje jeho efektivní zpětné vyhledávání. Knihovna Apache Lucene indexuje data do speciálního typu databáze, tzv. invertovaného indexu (z anglického překladu inverted index) [6]. Nad invertovaným indexem později probíhá vyhledávání.

Součástí procesu indexování je analýza textu, která předchází samotné indexaci a kterou mají na starost analyzátory. Ty mohou být pro každé jednotlivé pole různé. Toho se dá v praxi velmi dobře využít. Například když je třeba ke každému jednotlivému poli objektu přistupovat různě. V návrhové části práce, kde je navržena také optimalizace analyzátorů, je jejich chování detailněji popsáno.

Další metodou optimalizace již při indexovací fázi může být přidání speciálních polí (obvykle obsahujících číselnou hodnotu), které budou později při vyhledávání použity k rekalkulaci relevantnosti daného výsledku. Metodu lze využít také při vyhledávací fázi. V indexovací fázi

se jí ale využívá v případě, že jednotlivé dokumenty mají různou důležitost, kterou by později již nebylo možné určit.

2.3.2.2 Optimalizace Vyhledávací fáze

Vyhledávání je v kontextu fulltextového vyhledávání proces, kdy se zadaná fráze (případně i další omezující parametry) porovnává s indexovanými dokumenty v indexu. Jako výsledek je pak vrácena sada dokumentů, u nichž byla nalezena shoda. Výsledková sada má dokumenty seřazené dle jejich relevance.

V dnešní době může hrát i několik málo milisekund výraznou roli. Pakliže vyhledávání trvá příliš dlouho, uživatel může ztratit trpělivost a stránku opustit. Přestože vyhledávací nástroje jsou v dnešní době natolik výkonnostně optimalizované, že dokáží i složité dotazy nad relativně velkým počtem dat v indexu provést v řádech milisekund, jak je vidět v tabulce (Tab. 2), je potřeba na tuto fázi dávat také pozor.

Ve vyhledávací fázi se formuluje dotaz, na základě kterého je fráze porovnávána s indexovanými dokumenty. A právě to, jakým způsobem je fráze s dokumenty porovnávána, má velký vliv jednak na výkon a rychlost poskytnutí výsledků, ale hlavně na požadovanou kvalitu výsledků. Protože ale každý projekt má na kvalitu jiné požadavky a měřítka, nelze obecně určit, jaké optimalizační kroky je třeba v daném případě učinit. V návrhové části práce je ale navržena a otestována řada optimalizačních kroků, které lze převzít a přizpůsobit danému projektu.

Při snaze o optimalizaci vyhledávací fáze je třeba zvážit, jaké shody jsou pro daný projekt ještě relevantní a jaké už ne. Při malém projektu a malém množství indexovaných objektů nemusí být rozdíl příliš znatelný, ale s jejich rostoucím počtem se proces vyhledávání dokáže výrazně zhoršit.

2.3.2.3 Regulární výrazy

Regulární výrazy slouží k nalezení shody na základě definovaného patternu (schéma), které popisuje určité množství textu. Jméno regulárních výrazů je odvozené od regulárních gramatik, na kterých jsou založeny [7].

Regulární výrazy se dají při fulltextovém vyhledávání použít několika způsoby. Základním využitím regulárního výrazu je vyhledání všech různých řetězců, které odpovídají zadanému patternu. Avšak použití regulárního výrazu při vyhledávací fázi může mít z důvodu velkého množství nutných porovnávání velký vliv na výkon.

Lze je také využít již při indexovací fázi. Konkrétně jako součást analyzátoru, kde je možné regulární výraz použít k vytvoření sady tokenů podle výrazem definovaného pravidla. Tímto způsobem je například možné za pomoci správně definovaného patternu pro CamelCase oddělit jednotlivá slova napsaná v dokumentu tímto způsobem.

Je třeba optimalizovat samotný regulární výraz. Špatně definovaný výraz dokáže výrazně zpomalit nebo i shodit celou aplikaci.

2.3.2.4 Sugesce

Ne vždy uživatel napíše přesně to, co chtěl, nebo záměrně nedokončí celý dotaz. Místo toho, aby se jako odpověď vrátila prázdná výsledková sada, je možné zkontrolovat index detailněji a nalézt podobnosti. Tomuto způsobu se říká sugesce, jež se dá také považovat za jeden ze způsobů optimalizace, protože díky ní je možné vrátit i výsledek pro nepřesně napsaný dotaz místo toho, aby musel být dotaz podán znovu. Stejně tak jako ohlídání uživatelských překliků je možné sugesci použít také pro tzv. “autocomplete” nebo “search-as-you-type” sugesci, která vyhledává ještě před tím, než je celý dotaz zformulován. Více o jednotlivých metodách sugesce v návrhové části práce v rámci implementace suggesterů.

2.3.2.5 Synonyma

Existuje spousta slov, která se dají říci několika různými způsoby. Proto synonyma hrají při fulltextovém vyhledávání významnou roli a lze tímto způsobem velmi dobře optimalizovat.

Fulltextový vyhledávač dokáže jako výsledek vrátit i ty dokumenty, které neobsahují přímo uživatelem zadané slovo ale jen některé z jeho synonym.

2.3.2.6 Strojové učení

Hlavním cílem strojového učení je navrhnout a vytvořit algoritmy, které umožňují systému používat empirická data, vyvíjet se na základě vlastních zkušeností a připravovat se na změny, které mohou v systému nastat [8]. Toho se dá využít i v případě fulltextového vyhledávání, a tak může být strojové učení velice mocným nástrojem v oblasti optimalizace a analýzy indexu i relevantnosti navrácených výsledků ve výsledkové sadě. Řada současných vyhledávacích technologií strojové učení podporuje a lze ho nějakým způsobem použít a přizpůsobit danému problému. Velikost tohoto tématu ale dalece přesahuje obsah této práce a je zde uvedeno pouze jako další možnost optimalizace.

3 Technologie pro fulltextové vyhledávání

Technologií, které umožňují fulltextově vyhledávat je mnoho. Existuje řada technologií, které jsou tzv. “open-source” (otevřený zdrojový kód) a ty jsou z převážné většiny zdarma. Zpoplatněny mohou být služby, které rozšiřují základní funkcionalitu, jako je například strojové učení u technologie Elasticsearch. Jedním z typů open-source projektů jsou technologie, které jsou nadstavbou knihovny Apache Lucene a zpřístupňují tedy její myšlenky a funkcionality v jednodušší podobě než při použití samotné knihovny. Další variantou je zvolit jednu z několika technologií, které jsou “closed-source” a poskytují tzv. “SaaS” (Search as a Service). Takovou technologií může být například technologie Splunk nebo Algolia. Za hlavní výhodu closed-source technologií se dá považovat jejich nezávislost na Lucene knihovně, která ačkoli přináší skvělé myšlenky a podporu velice rychlého indexování a vyhledávání, nepřináší takový výkon jako novější a modernější technologie, které jsou lépe optimalizované pro dnešní dobu a poskytují indexování a vyhledávání podle [9] až 200x rychleji než právě Lucene technologie. Za jejich další výhodu se dá považovat pokročilá podpora strojového učení, které už nyní začíná mít, a v budoucnu jistě mít bude, obrovské využití.

Ačkoli Splunk, Algolia nebo ostatní novější technologie mohou být v některých ohledech lepším řešením, Lucene technologie dokáží stále poskytnout velice vysoký výkon a podle [10] je to právě Elasticsearch technologie, která jako nadstavba knihovny Apache Lucene je nejvíce využívanou technologií pro fulltextové vyhledávání. Z tohoto důvodu a z důvodů které vyplývají z tabulky (Tab. 1), kde jsou jednotlivé Lucene technologie porovnány, je i pro projekt, nad kterým je tato práce sepsána, zvolena technologie Elasticsearch.

3.1 Apache Lucene

Apache lucene je fulltextový open-source engine napsaný v jazyce JAVA, který je přístupný jako knihovna, kterou je možné využít pro vyhledávací účely.

V jakékoli JAVA aplikaci je možnost využití vyhledávání postaveného pouze na této knihovně. Nicméně veškeré nastavení (nastavení clusteru, uzlů, fragmentů atd.) je potřeba ošetřit. Toto je hlavní nevýhodou a většinou rozhodujícím faktorem, proč si nevybrat pouze tuto knihovnu v její čisté podobě bez nadstavby, kterou poskytují frameworky vystavěné nad ní. Framework jako např. Elasticsearch má svůj vlastní Zen Discovery modul pro cluster management, naproti tomu Solr používá externí ZooKeeper a díky tomu není nutné se o Cluster management starat vždy od začátku.

3.1.1 Technologie nad Apache Lucene

Technologií postavených na základech Apache Lucene existuje celá řada. Pro porovnání v tabulce (Tab. 1) jsou zvoleny technologie Elasticsearch, Solr a Sphinx. Tyto technologie patří podle [10] mezi nejpopulárnější technologie v oblasti fulltextového vyhledávání a existují všechny ze stejného důvodu. Tím je zprostředkovat uživateli všechny výhody rychlého až real-time indexování a vyhledávání.

Pro tuto práci byla zvolena technologie Elasticsearch, která podporuje všechny základní funkcionality fulltextového vyhledávání, je stále ve vývoji a navíc tuto technologii využívá mnoho velkých a významných projektů, což je zárukou, že její podpora bude ještě nějakou dobu pokračovat. Mezi nejznámější firmy využívající technologii Elasticsearch patří například Wikipedie, StackOverflow nebo GitHub.

Tab. 1: Porovnání nejvyužívanějších technologií pro fulltextové vyhledávání.

Feature	Framework		
	ElasticSearch	**Solr**	**Sphinx**
Komunita a vývojáři	Společnost Elastic. Open-source projekt. Veřejně na GitHubu. Každý se může podílet na vývoji, ale Elastic tým má poslední slovo	Community over Code = kdokoli, kdo projeví zájem na vývoji může commitovat do projektu. Solr nemá centrální kontrolní společnost.	Sphinx Developers = malá skupina vývojářů, především studenti. Open-source projekt.
Poslední release	6.1.2, Leden 2018	7.2.1, Leden 2018	3.0.1, Prosinec 2017
Napsán v jazyce	JAVA	JAVA	C++
Cluster management	Vlastní produkt Zen - poskytuje úplný node management	Apache Lucene Zookeeper - vyžaduje externí soubor	Nemá vlastní implementaci, vyžadována externí. Př. BroControl
Cache	Pro každý segment = lepší pro dynamicky se měnící data	Globální = při změně segmentu je třeba změny celé cache	Od verze 2.3.1
Query optimalizace	Základní optimalizace + rychlejší query závislé na čase.	Základní	Základní
Rychlost	Rychlá - inverted index, rychlejší pro dynamicky se měnící data	Rychlá - inverted index, rychlejší pro statická data.	Rychlá - 7. září 2017 první úspěšné testy s indexy, od verze 3.0.1 podpora indexu, dříve potřeba externí databáze pro ukládání a získávání dat. (MySQL)
Featury	Highlighting, rescoring, zjednodušená Suggest implementace (snadnější na použití, možné více využití)	Více suggesterů: spell checkers, highlighting	Autocomplete, correction suggestion, highlighting
Query	JSON, URL parametry	JSON, XML, URL parametry	JSON, podpora XML, query s nastavením vytvořená v některém z podporovaných jazyků
Základní podporované jazyky	.Net, Clojure, Erlang, Go, Groovy, Haskell, JAVA, JavaScript, Lua, Perl, PHP, Pzthon, Ruby, Scala	.Net, Erlang, Java, JavaScript, Perl, PHP, Pzthon, Ruby, Scala	C++, Java, Perl, PHP, Python, Ruby

3.2 Elasticsearch

Elasticsearch je vyhledávací a analytický engine, který umožňuje prohledávat data téměř v reálném čase. Jeho použití je na fulltextové vyhledávání, strukturované vyhledávání, analýzu nebo kteroukoli kombinaci těchto tří možností. Je vhodný pro velké korporace a obrovské projekty, ale lze ho přizpůsobit i malému domácímu projektu. Stejně tak jako dokáže běžet na jednom notebooku, dokáže běžet na desítkách spolupracujících separátních serverů.

Elasticsearch nepřináší nic nového. Žádná z jeho částí (v open-source variantě zdarma bez rozšíření v podobě například zpoplatněného strojového učení) není novinkou, jelikož pouze zpřístupňuje funkcionalitu knihovny Apache Lucene. Ta je pravděpodobně nejvíce rozšířenou a plně podporovanou vyhledávací knihovnou dneška, ale je to jen knihovna. K jejímu využití je potřeba pracovat s jazykem JAVA a integrovat ji přímo do aplikace. Lucene je velice komplexní a složitá na pochopení a správu v případě jejího využití bez nadstavby. Elasticsearch je také napsán v jazyce JAVA a využívá Lucene knihovnu pro veškeré vyhledávání a indexování. Navíc ale, na rozdíl od samotné knihovny, veškerou funkcionalitu zabaluje na standalone server a díky tomu je veškerá funkčnost knihovny Apache Lucene daleko přístupnější. Se serverem je možné komunikovat několika způsoby. Skrze Restful API z webového prohlížeče, z vlastní aplikace nebo jednoduše z příkazového řádku.

Každá databáze dokáže selektovat data na základě přesných údajů. Dokáže filtrovat na základě času, třídit, atd. Ale co nedokáže je fulltextové vyhledávání, nahrazování synonymy, hodnocení dokumentů dle jejich relevantnosti, nedokáže agregovat data a generovat analýzu. Toto vše se v případě technologií Apache Lucene a Elasticsearch děje, navíc téměř v reálném čase.

Elasticsearch je dokumentově orientovaný. To znamená, že jednak ukládá a indexuje celé objekty, ale také celé dokumenty hledá, řadí nebo třídí. Tím se liší od běžných relačních databází, u kterých se to děje jen v rámci sloupcových údajů. Ke správě dokumentů Elastic používá formát JSON, který je jednoduchý a dobře čitelný. Navíc práce s ním je velice rychlá a efektivní.

3.2.1 Cluster management - Zen discovery

Elasticsearch používá pro správu clusteru a jeho jednotlivých uzlů vlastní vestavěný modul nazvaný Zen discovery. Ten má na starosti volbu hlavního uzlu, detekci chyb a správu globálního stavu.

Ke zjištění stavu a objevování nových uzlů v clusteru používají jednotlivé uzly proces Ping. Ten je také základním procesem Zen discovery modulu a na základě něj probíhá komunikace mezi jednotlivými uzly. Definice toho, kam budou uzly "pingovat", jsou uloženy jako list hostů v Unicast discovery modulu. Jedním z úkolů modulu a také jedním z využití procesu ping je detekce hlavního ("master") uzlu. Takový uzel musí existovat právě jeden a je automaticky zvolen, pokud již neexistuje. Rozhodnutí o tom, zda se uzel připojí k existujícímu hlavnímu uzlu nebo proběhne volba nového, probíhá na základě 3 pingů, které jednotlivé uzly broadcastem odesílají a jejichž timeout je definovatelný. V případě překročení všech je za potřebí zvolit nový master uzel. Pokud ale přijde odpověď před timeoutem, uzel vyšle tzv. Join request, díky kterému se připojí k hlavnímu uzlu.

Zen discovery se dále stará o detekci chyb. Za prvé se stará o detekci chyb u master uzlu. Ostatní uzly odesílají ping na master uzel, aby zjistily, zda je stále aktivní. Zároveň ale také master uzel odesílá ping do ostatních uzlů za účelem zjištění, zda jsou stále aktivní. Timeout a časový interval mezi jednotlivými pingy jsou modifikovatelné.

Dalším úkolem Zen discovery je udržování a aktualizace stavu. Jedině master uzel může držet globální stav celého clusteru a stará se o aktualizace. Pouze jedna aktualizace může v jeden čas běžet. Master uzel aplikuje požadované změny a odešle publish všem ostatním uzlům. Ty tuto zprávu obdrží a odešlou acknowledge zprávu zpět do master uzlu. Rozhodnutí o změně učiní master uzel až v případě příchodu acknowledge zprávy od více než definovaného minima uzlů. Poté je aktualizace buď zamítnuta nebo je rozhodnuto o jejím aplikování a každý z uzlů si aktualizuje svůj lokální stav.

3.2.2 Uzel (=Node)

Kdykoli je spuštěna instance Elasticu, je spuštěn jeden uzel. Každý uzel je schopen samostatné existence a v základu dokáže přijímat HTTP a transportní requesty. HTTP pro komunikaci s restful API a transportní pro komunikaci navzájem mezi uzly a pro příjem requestů z JAVA API (za pomoci JAVA Transport Client). Všechny uzly uvnitř jednoho clusteru musí mít jedinečné jméno, navzájem o sobě vědí a dokáží si mezi sebou přeposílat zprávy. Funkčnost vzájemné komunikace je velice důležitá, protože díky ní je možné zátěž rozdělit mezi několik uzlů. Díky této komunikaci je možné při nedostatku výkonu nebo místa provést tzv. horizontální rozšíření, tj. rozdělení zátěže mezi několik uzlů, které je možné provozovat i na více zařízeních, a není nutné pouze radikální tzv. vertikální rozšíření, tedy rozšíření ve smyslu nákupu větších, výkonnějších zařízení.

Jeden uzel dokáže zároveň sloužit více účelům. Záleží na konfiguraci, k jakému účelu je uzel určen. Existují 4 typy uzlů. Je to hlavní uzel, datový uzel, ingest uzel a koordinační uzel.

Cluster obsahuje právě jeden hlavní uzel. Je velice důležité mít v clusteru stabilní hlavní uzel. Ten je zodpovědný za rovnováhu celého clusteru, monitoruje jednotlivé uzly a podle zátěže přiděluje fragmenty ("shards"). Kterýkoli uzel může být zvolen jako hlavní během volebního procesu, který je, jak bylo popsáno v kapitole Cluster management, automaticky prováděn k nalezení hlavního uzlu.

Jedním z dalších typů uzlů je datový uzel. Uzel, který je nastaven jako datový, dokáže udržet fragmenty, které obsahují indexovaná data, a provádět nad nimi operace jako jsou CRUD, vyhledávání nebo agregace. Tyto uzly jsou velice vytěžovány, je potřeba je monitorovat a v případě přetížení rozdělit zátěž mezi více datových uzlů.

V případě, že je potřeba, aby byl dokument před indexací modifikován, používá se ingest uzel. Ten definuje pipeline, která zapříčiní za pomoci definovaných ingest procesorů modifikaci dokumentu (například odebrání nebo přidání některého pole). Pipeline jsou poté uloženy ve stavu clusteru.

Při větší velikosti clusteru je možné připojit pouze koordinační ("coordination only") uzel. Ten pak přebírá koordinační roli od datového a hlavního uzlu a tím ulevuje jejich zátěži. Po připojení uzel obdrží kompletní stav celého clusteru a distribuuje requesty přímo na místo, kam patří.

3.2.3 Cluster

Cluster je balík, který obsahuje jeden nebo více uzlů. Jak jsou uzly přidávány nebo odebírány, cluster se sám dokáže reorganizovat a rovnoměrně rozdělovat data.

Díky tomu, že si cluster uchovává stav, je možné jednotlivé clustery monitorovat. Sledovat je možné například počet uzlů, kolik uzlů je datových nebo třeba status, jenž dokáže sdělit, zda jsou všechny primární a replikované fragmenty aktivní.

3.2.4 Fragment (=Shard)

Fragmenty se využívají k rozdělení indexu na menší části. Každý fragment je samostatná fungující jednotka (samostatná instance Apache Lucene), která dokáže plně manipulovat se svou částí dat. Dokumenty jsou indexovány do fragmentů, ale aplikace nekomunikuje přímo

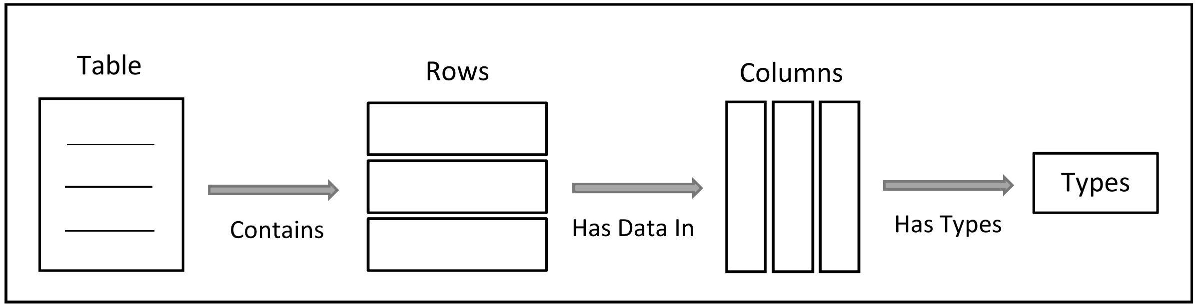
s nimi. Namísto toho komunikuje s indexem, který se stará o distribuci požadavku do správného fragmentu.

O fragmentech se dá přemýšlet jako o kontejnerech pro data. Dokumenty jsou ukládány do fragmentů, kde každý fragment náleží nějakému uzlu v clusteru. Jak cluster roste, Elasticsearch automaticky migruje fragmenty mezi uzly a tím zůstává cluster v rovnováze [5].

V primárním fragmentu jsou uložena indexovaná data. Počet takových fragmentů je pevně dán při vytváření indexu a později ho již nelze změnit. K tomu, aby nedošlo ke ztrátě indexovaných dat v primárním fragmentu, je možné využít několika replikovaných fragmentů. Replikované fragmenty existují pro případ, že primární fragmenty nebudou z nějakého důvodu k dispozici, například když uzel, ve kterém byly vytvořeny, náhle zkolabuje nebo je zkrátka jen offline a není přístupný. Replikované fragmenty pak suplují za primární fragment. Proto je důležité replikované fragmenty mít a mít je zároveň uložené v jiném uzlu. Počet replikovaných fragmentů je možné dynamicky měnit.

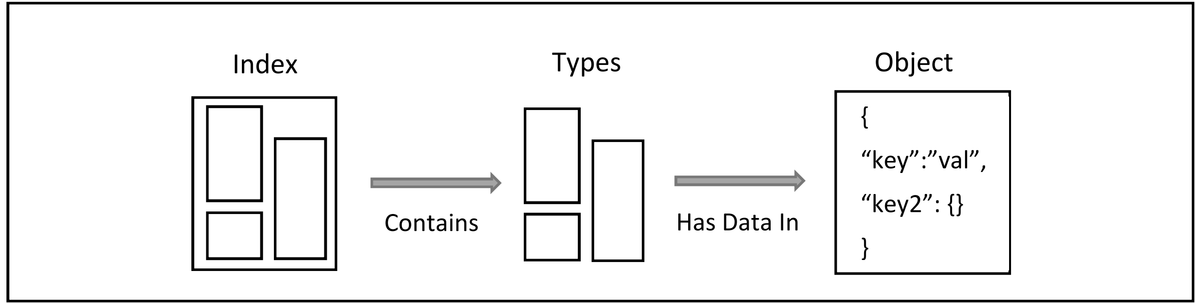
3.2.5 Ukládání dat

Tradiční relační databáze se skládá z tabulek navzájem provázaných pomocí primárních klíčů. Tabulky obsahují data uložená v řádcích s hodnotami uloženými ve sloupcích, které mají přesně definovaný typ (Obr. 2).



Obr. 2: Struktura dat v relační databázi

Naproti tomu Elasticsearch se dá spíše přirovnat k některé NoSQL databázi, jako je například MongoDB. Místo databáze má Elasticsearch index, který obsahuje jednotlivé typy. V každém typu jsou obsaženy dokumenty, jejichž data jsou rozčleněna do netypových polí (Obr. 3).



Obr. 3: Struktura dat ve fulltextovém indexu

3.2.5.1 Index

Index je kolekce dokumentů nebo kolekce typů, které mají podobnou charakteristiku. Například je možné mít jeden index pro zaměstnance a druhý index pro objednávky. Index je definovaný jménem, podle kterého je správný index během průběhu všech procesů (indexace, vyhledávání, atd.) vybírán, proto musí být jméno unikátní.

3.2.5.2 Typ

V rámci jednoho indexu je možné definovat jeden nebo více typů. Počet typů je naprosto libovolný. Slouží zejména pro přehlednost. Pro představu, v indexu „Zaměstnanci“ je možné vytvořit typ “Manažer” nebo “Prodejce”.

3.2.5.3 Dokument

Dokument je základní informační jednotkou pro Elasticsearch. Nese v sobě informace o objektu, který popisuje. Například to může být jeden Zaměstnanec. Dokument, nebo spíše virtuální dokument, jelikož se nemusí jednat o reálný dokument, je uložen ve formátu JSON jako množina polí. Apache Lucene, tedy i Elasticsearch, využívá dokumenty jako objekty, které se indexují a mezi kterými je možné vyhledávat.

3.2.5.4 Pole

Dokumenty knihovny Apache Lucene jsou složeny z polí. Pole se skládá z klíče a hodnoty. Všechny hodnoty jsou netypové. Knihovnu nezajímá, zda pracuje s číslem nebo textem. Všechny hodnoty si knihovna převádí na sérii bajtů.

3.2.5.5 Token

Hodnota pro jednotlivá pole je dále rozdělena na tzv. tokeny. Token si můžeme představit jako například jedno slovo ve větě. Jejich tvorba je nedílnou součástí indexovacího procesu. Při fulltextovém vyhledávání se vyhledává právě mezi tokeny.

3.2.6 Query

Elasticsearch podporuje velmi flexibilní dotazovací jazyk Query DSL, který umožňuje psát i velice komplikované a robustní dotazy, které jsou psány ve formátu JSON.

Uvnitř dotazu jsou napsány jednotlivé query, případně filtry. Elasticsearch vyhodnotí query a vypočítá výsledné score dokumentu, které určuje, jak moc je vyhledaný dokument relevantní vzhledem k ostatním indexovaným dokumentům. Dotazy se vykonají a jejich score se vypočítá vždy, když uvnitř napsaného dotazu použijeme parametr query.

Naproti tomu filtr výsledné score nepočítá. Odpovídá pouze na otázku, zda dokument sedí na položený dotaz či nikoliv. Filtr se použije v případě, že je uvnitř dotazu použit parametr filter.

Často používané dotazy si Elasticsearch dokáže cachovat, a tím tak umožnit pozdější efektivnější a rychlejší použití toho samého dotazu.

3.2.7 Komunikace

To jak probíhá komunikace s Elasticsearch serverem záleží na tom, odkud jsou volány dotazy. Komunikace může probíhat dvěma způsoby. Za použití JAVA API přes port 9300 nebo přes restful API s použitím portu 9200.

3.2.7.1 JAVA API

Jestliže je aplikace, ve které je za potřebí fulltextové vyhledávání, napsána v jazyce JAVA, pak lze s Elasticem komunikovat přes JAVA API. Elasticsearch při této formě komunikace přichází se dvěma vestavěnými klienty. Jedním je Node Client a druhým Transport Client. Oba klienti komunikují s clusterem prostřednictvím http protokolu na portu 9300. Stejně tak jednotlivé uzly uvnitř clusteru komunikuji přes tento port mezi sebou.

Node client se připojí k lokálnímu clusteru jako coordination-only uzel, jehož funkce je popsána v podkapitole Uzel kapitoly Cluster management. Tento uzel neobsahuje žádná data, ale pouze ví, jaká data jsou v jednotlivých uzlech clusteru a posílá dotazy přímo do správných uzlů, kterým dotaz náleží.

Naproti tomu transportní klient je jednodušší ("lehčí") klient, který může být použit k odesílání requestu na vzdálený cluster. Nepřipojuje se ke clusteru přímo, ale jednoduše přeposílá requesty do uzlů v clusteru, jejichž stav si udržuje díky tzv. Sniffing technologii, která spočívá v pravidelných requestech ke zjištění všech datových uzlů a na základě jejích výsledků je

aktualizován list těchto datových uzlů. Requesty jsou poté přeposílány přímo těmto datovým uzlům.

3.2.7.2 Restful API

Druhou možností, jak komunikovat s Elasticsearch serverem, je přes restful API. Odkudkoli je možné s Elasticsearch serverem komunikovat za použití http protokolu přes port 9200. Je možné posílat requesty z aplikace v kterémkoli jazyce, z webového prohlížeče nebo klidně z příkazového řádku.

3.2.8 Rozšířeni

Elasticsearch kromě řady vestavěných funkcionalit podporuje také několik externích rozšíření, které základní funkcionalitu doplňují o další užitečné funkce. Díky nim je možné například lépe monitorovat stav serveru, díky čemuž je snadnější detekce chyb nebo omezení, které by mohly snižovat výkon.

3.2.8.1 X-pack

X-pack je balíček rozšíření pro Elasticsearch, který v sobě zahrnuje řadu pluginů a který stačí pouze nainstalovat. Sám se stará o správu a kompatibilitu jednotlivých pluginů a díky tomu tedy odpadá starost s různými verzemi pluginů apod.

Mezi pluginy můžeme nalézt nástroje starající se o bezpečnost, monitoring, reporting nebo schopnost vizualizace dat pomocí práce s grafy.

3.2.8.2 Kibana

Kibana je open-source analytická a vizualizační platforma vytvořena pro práci s Elasticsearch. Dokáže přehledně zprostředkovat data uložená v Elasticsearch indexech, provádět nad nimi analýzu, případně je zanášet do grafu, map apod. Díky prostředku Kibana je možné také snadno interagovat s indexovanými daty, dále je k dispozici vizualizace dat posbíraných z pluginů nainstalovaných v balíčku X-Pack.

3.2.8.3 Logstash

Logstash je open-source datový engine, díky kterému je možné dynamicky, v reálném čase sjednotit data z různých zdrojů a podle požadavků je normalizovat. Díky tomuto nástroji je možné snadno pročistit nebo přizpůsobit data pro různorodé případy užití.

3.2.8.4 Beats

Beats je opět open-source projekt, který je nainstalován jako agent na serveru aplikace a zprostředkovává pro Elasticsearch různé informace (buď přímo nebo za pomoci Logstash), dle toho, jaký beat máme nakonfigurován. Existuje několik různých typů beatů. Například to mohou být Packetbeat, Filebeat nebo Metricbeat. Packetbeat je síťový analyzátor, který posílá informace o transakcích provedených mezi aplikačními servery. Filebeat posílá log soubory ze serveru. Metricbeat je zase monitorovací agent, který periodicky sbírá data z operačního systému a z procesů běžících na serveru.

4 Cíl práce

Hlavní otázkou je, jak dosáhnout optimálních výsledků vyhledávání. Podle [11] je programová nebo také softwarová optimalizace proces modifikace systému s účelem učinit systém více efektivní. Efektivitu systému lze chápat z několika úhlů. Optimalizovaný systém je takový systém, který k fungování potřebuje méně zdrojů, systém, který pracuje rychleji, nebo systém pracující s daty, který vrací pouze požadované výsledky navíc ve správném pořadí, které určí uživatel nebo systém sám.

Podle [12] existuje několik úrovní optimalizace, z nichž minimálně dvě úrovně jsou v kontextu fulltextového vyhledávání podstatné, a ty je třeba optimalizovat. Je to úroveň designová a úroveň algoritmů a datových struktur. Existují i další úrovně optimalizačního procesu. Je možné optimalizovat zdrojový kód aplikace nebo v neposlední řadě optimalizovat hardware pro zajištění efektivního běhu aplikace, ale tyto kroky bývají velice obtížné, velice nákladné až nemožné. Mnohdy navíc mohou být zbytečné. To v případě špatného designu aplikace nebo špatně navržené datové struktury, která může být právě důvodem horšího výkonu aplikace.

Cílem práce je tedy navrhnout řešení dvou zmíněných úrovní optimalizace.

5 Volba testovací metodiky

V dalších kapitolách je popsána aplikace, která slouží jako podklad pro tuto práci a na základě které jsou navrženy metody optimalizace, které jsou postupně implementovány.

Řešení obsahuje pět verzí, kde první je bez optimalizací a v každé další je implementována jedna optimalizační metoda. Na nich je provedena série testů, ze kterých jsou následně sbírána sledovaná data, jako jsou pozice hledané stránky ve výsledkové sadě a počet navrácených výsledků. To jsou data, která jsou z hlediska navržené optimalizace podstatná a v následujících kapitolách budou blíže specifikována.

Po získání dat z testování jednotlivých optimalizačních metod v aplikaci jsou tato data zanesena do grafů a je na nich provedena statistická analýza pro vypočítání významnosti výsledků. Konkrétně je zvolen krabicový graf pro vizualizaci pozice stránky a sloupcový graf pro znázornění počtu vrácených stránek po zadání jednotlivých dotazů.

Pro statistickou analýzu pozice nalezených výsledků byl zvolen Kruskal-Wallisův test, který podle [13] testuje některé rozdíly mezi skupinami, ale neposkytuje žádná post hoc párová porovnání mezi skupinami. Za tímto účelem se dá vyžít několik možností. Například Dunnův test nebo Tukeyho test, ale podle [14] je nejvhodnější použití právě Dunnova testu v případě, že známe přesný počet porovnání již před tím, než provedeme analýzu rozptylu za pomoci Kruskal-Wallisova testu.

Pro analýzu počtu vrácených stránek je zvolena metoda pro hodnocení čtyřpolních tabulek (kontingenční tabulka obsahujících 2 veličiny, kde každá veličina může nabývat pouze dvou různých variant). Pro hodnocení statistické významnosti je zvolen Fisherův exaktní test, díky kterému je možné vypočítat, jak je řečeno v [15] "přesnou (exaktní) pravděpodobnost, se kterou bychom za platnosti nulové hypotézy o nezávislosti veličin X a Y získali naší konkrétní realizaci čtyřpolní tabulky".

6 Návrh řešení

Tuto práci lze brát jako doporučení pro optimalizaci fulltextového vyhledávání, které lze obecně aplikovat pro široké spektrum aplikací, které ho chtějí využívat. V následujících příkladech a případech užití jsou veškeré obrázky a ukázky kódu demonstrované pro technologii Elasticsearch. Doporučení lze převzít i pro ostatní technologie. Uvedené ukázky se ale mohou pro odlišné technologie lišit, jelikož každý framework užívá odlišnou syntaxi a také výsledky pro jednotlivé příklady nemusí být zcela přesné, protože každá technologie má implementované jiné algoritmy a modifikátory pro výpočet významnosti vyhledaných dokumentů, pokud tuto hodnotu vůbec počítají.

6.1 Implementace

Jak již bylo v dřívějších kapitolách zmíněno, pro aplikaci sloužící jako podklad pro práci byla zvolena technologie Elasticsearch, která je velice dobře optimalizovaná z hlediska zdrojového kódu co do časové efektivnosti a paměťové náročnosti (viz Tab. 2). Optimalizace hardwaru, ve smyslu nákupu lepších a výkonnějších zařízení bývá často až poslední, a ne vždy nezbytnou možností. Mezi hlavní úkoly vývojáře, který se chystá implementovat fulltextové vyhledávání, je tedy správný designový návrh aplikace, návrh datových struktur a rozhodnutí o správném použití té dané vlastnosti systému, který implementuje.

Tab. 2: Znázornění výkonové efektivity Elasticsearch

Počet indexovaných stárnek	Čas indexace (ms)	Čas vyhledávání (ms)	Velikost indexu (B)
1	48	8	7592
10	87	12	9647
100	122	19	47703
1000	254	35	449318
5000	913	87	1600114

6.2 Testovací aplikace

Pro účely testování optimalizačních metod byla vyvinuta aplikace uuFulltextovéVyhledávání. Je to aplikace vytvořená jako služba, jejíž cílem je umožnit webovým aplikacím pod zadaným URI zaindexovat svůj obsah pro fulltextové vyhledávání a zpřístupnit tak vyhledávání podle vytvořeného indexu.

Služba využívá kromě již zmíněné technologie Elasticsearch určené pro fulltextové vyhledávání také technologii MongoDB, což je NoSQL databáze, sloužící pro zálohování podstatných údajů. Elasticsearch instance je nasazena na 3 uzlech kvůli zajištění stability. Jelikož data z těchto uzlů nejsou zálohovaná, tak při výpadku všech 3 uzlů dojde ke ztrátě dat. Ovšem díky databázi, která drží podstatná data, jako jsou uri a další, je možné data v Elasticsearch instanci obnovit.

Dalšími technologiemi použitými v aplikaci jsou Sellenium a Chrome driver. Tyto technologie se starají o vyrenderování stránky definované danou uri na serveru v textové podobě. Odtud si aplikace vezme textová data, která zaindexuje.

6.2.1 Business procesy

Mezi základní a klíčové procesy, které aplikace přináší, patří indexování, vyhledávání, monitoring indexu a správa indexu. Procesy jsou vyvolávané buď přímo uživatelem (například přímé odeslání uri k zaindexování) nebo jsou volány systémem samým (například automatické renderování stránky do paměti a následná indexace do fulltextového indexu) jednou za definovaný interval nebo jako reakce na událost.

Proces indexování zajišťuje uložení dat do fulltextového indexu pro definovanou uri. Proces probíhá tak, že na server přijde request s několika uri k zaindexování. Aplikace tyto údaje uloží do noSQL databáze a procesor, který jednou za definovaný časový interval tuto databázi prohlíží, zda-li neobsahuje dokumenty, které čekají na zaindexování, v případě že nalezne dokument, spustí proces renderování stránky a její následné zaindexování.

Proces vyhledávání umožňuje vyhledávání dat uložených pod definovanou uri ve fulltextovém indexu. Opět na server přijde request, který obsahuje údaje, jež mají být vyhledány v indexu. Aplikace složí všechna známá data a vytvoří dotaz, který odešle za pomoci transportního klienta na Elasticsearch server, odkud se vrátí odpověď. Odpověď obsahuje sadu dokumentů seřazených podle hodnoty score, která určuje relevantnost daného dokumentu.

V poslední řadě procesy Monitoring indexu a Správa indexu slouží pro zjišťování informací a modifikaci některých údajů ve fulltextovém indexu. Dokáží zjistit velikost nebo limity indexu a také provádět nastavování některých údajů.

6.2.2 Business role

Pro zajištění správného fungování a organizovatelnosti indexu aplikace dokáže pracovat s různými typy uživatelů (různými rolemi). Každá role má nebo naopak nemá právo využívat některou z funkcionalit.

6.2.3 Klíčové vlastnosti

Mezi klíčové vlastnosti, kromě realizace základních business procesů a práce s business rolemi, patří v případě aplikace uuFulltextovéVyhledávání podpora multijazyčnosti, přímé indexování dokumentů bez jejich renderování a jejich záloha, rescoring dokumentů na základě vlastností dokumentu nebo filtrování výsledků na základě práv uživatele, který vyhledávání inicializuje.

Tyto klíčové vlastnosti mohou být důležité konkrétně pro aplikaci uuFulltextovéVyhledávání, ale ne tak moc pro ostatní aplikace. Ovšem každá aplikace může mít jiné specifické myšlenky, proto je třeba dbát na správném designovém návrhu a návrhu správné datové struktury, ve které se budou data do indexu a do zálohované databáze ukládat. Objekt datové struktury by měl v ideálním případě obsahovat vše podstatné a být snadno udržitelný, modifikovatelný a dále jednoduše rozšiřitelný.

6.3 Úvod do základního nastavení

Pro splnění požadavků dvou základních business procesů, mezi které patří indexování dokumentů a fulltextové vyhledávání v nich, což je zároveň i klíčovou myšlenkou celé služby pro fulltextové vyhledávání, postačuje základní návrh datové struktury. Struktura (Code 1), je strukturou objektu v databázi, která, jak již bylo zmíněno, slouží jako držitel podstatných údajů. Údaje, jako jsou index, uri a indexInfo, slouží jednak jako záloha v případě neočekávaného výpadku a spadnutí běžící Elasticsearch instance ve všech uzlech, ale zároveň slouží také jako “mezikrok” v procesu objektů k indexaci. Poté co aplikace obdrží tyto objekty, uloží data do mongo databáze se stavem, který indikuje čekání na indexaci do Elasticsearch. Na tento stav dokáže zareagovat procesor, který údaje převezme, nechá stránku za pomoci technologií k tomu určených vyrenderovat a její obsah zaindexuje ve formátu, který je vidět na ukázce (Code 2).

Code 1: Struktura objektu v mongo databázi. Základní nastavení.

```
{
      "index": "exampleIndex",
      "uri": "http://www.example.com",
      "code": "example",
      "indexInfo": [
            {
                  "language": "cs",
                  "state": "WAITING/COMPLETED/FAILED",
            }
      ]
}
```

Code 2: Struktura fulltextového indexu. Základní nastavení.

```
{
      "_source": {
            "cs": "Lorem ipsum dolor sit amet",
            "code": "example",
            "uri": "http://www.example.com",
            "mts": "2018-04-13T09:21:44.835"
      }
}
```

Objekty při indexaci Elasticsearch ukládá do pojmenovaného indexu. Jak je ze struktury v ukázce Code 2 patrné, objekt obsahuje několik polí, mezi kterými může být dále fulltextově vyhledáváno. To umožňuje analyzátor textu, který daný text zpracuje, rozdělí na tokeny a uloží do invertovaného indexu. Elasticsearch podporuje širokou škálu již implementovaných analyzátorů s předdefinovaným chováním. Mezi ty patří, mimo jiné, například jazykové analyzátory pro analýzu textu s ohledem na jazyk, ve kterém je text napsán. Je možné použít pro každé pole jiný analyzátor nebo také odlišný analyzátor pro indexaci a pro vyhledávání, ale pokud není přesně specifikováno, použije se základní analyzátor pro každé pole.

Pro specifikaci analyzátorů pro jednotlivá pole slouží mapovací dokument, který obsahuje souhrn všech polí v daném typu, u kterých chceme specifikovat chování při analýze textu. V dokumentu je také možné definovat vlastní pojmenované analyzátory nebo tokenizéry, které analyzátory využívají. Vlastní analyzátory je možné definovat v případě potřeby podrobněji specifikovat zpracování textu. Základní mapovací dokument pro aplikaci uuFulltextovéVyhledávání je zobrazen na ukázce Code 3.

Code 3: Základní mapovací dokument pro aplikaci uuFulltextovéVyhledávání

```
{
     "mappings": {
         "page": {
           "properties": {
             "mts": {
               "type": "date"
             },
             "code": {
               "type": "keyword"
             },
             "uri": {
               "type": "keyword"
             },
             "cs": {
               "type": "text",
               "analyzer": "cs",
             },
             "en": {
               "type": "text",
               "analyzer": "en",
             }
         }
    }
}
```

6.4 Optimalizace

Do indexu, který je definovaný podle předešlé kapitoly, lze data vkládat a lze v něm také vyhledávat. Díky výkonnostním optimalizacím technologie Elasticsearch se navíc vše děje velice rychle. To je zřejmé z tabulky (Tab. 2) a téměř zde není prostor pro optimalizaci.

Hlavním cílem pro optimalizování je přizpůsobení výsledků, které Elasticsearch vrací jako odpověď na položený dotaz.

Výsledek, který Elasticsearch vrací, je sada objektů, kde každý objekt nějakým způsobem vyhovuje položenému dotazu. Jak moc je daný objekt relevantní specifikuje hodnota score, což je hodnota, kterou má každý objekt přiřazenou a podle které je výsledné pole objektů seřazeno. Algoritmy pro výpočet score jsou již v rámci Elasticsearch technologie implementovány. Existuje ale celá řada možností, jak výsledné score modifikovat.

Práce vychází z předpokladu, že čím výše je správný výsledek ve výsledkové sadě, tím spíše je uživatelem nalezen. Protože jsou výsledky ve výsledkové sadě technologie Elasticsearch řazeny podle hodnoty score, je hlavním optimalizačním cílem modifikace této hodnoty takovým způsobem, aby reflektovala relevantnost daného objektu ve výsledkové sadě.

6.4.1 Optimalizace vyhledávacího dotazu

K dotazování na data uložená v Elasticsearch indexech se používá flexibilní jazyk Query DSL, díky kterému je možné psát i velice komplikované a robustní složené dotazy, které jsou psané ve formátu JSON. Query DSL obsahuje dva typy klauzulí. Klauzule, jež hledají shody nebo částečné shody v jednotlivých polích. Mezi takové query se řadí například “match” nebo “term” query. Dalším typem klauzule jsou klauzule složených dotazů. Ty se dají použít k logické kombinaci více dotazů. Řadí se mezi ně například “bool” query, která rozhoduje, zda objekt z indexu odpovídá dotazu, nebo query jež ovlivňují chování ostatních dotazů, jako je například “functionScoreQuery”.

Některé dotazy jsou důležitější než ostatní, proto lze každému dotazu přiřadit jeho váhu a tím do značné míry ovlivnit výsledné score objektů z výsledkové sady.

6.4.1.1 Popisek stránky

V případě vyhledávání v rámci jedné aplikace může nastat situace, kdy uživatel přesně ví, co na stránce hledá, ale v kontextovém menu to není na první pohled zřejmé. V takovém případě je pravděpodobně do vyhledávače zadáno klíčové slovo, které definuje přesně tuto stránku.

Výsledky podle předešlého návrhu indexu (viz Code 2) mohou být ale zavádějící, jelikož dané klíčové slovo se na různých stránkách může také vyskytovat, mnohdy v hojnějším počtu. Z tohoto důvodu ve výsledkové sadě stránka, kterou uživatel doopravdy hledal, nemusí být v popředí. Je dobré proto k indexovanému objektu přiřadit i pole, jehož obsah definuje to, co je v daném objektu obsaženo.

V případě aplikace uuFulltextovéVyhledávání byl jako pole shrnující obsah stránky zvolen popisek (label) s dodatkem jazyku, pro který je daný popisek určen. Objektové struktury v Mongo databázi i Elasticsearch indexu byly o dané pole rozšířeny, jak je vidět na ukázkách datových struktur Mongo databáze (Code 4) a Elasticsearch indexu (Code 5).

Code 4: Struktura objektu v Mongo databázi. Přidání pole "label"

```
{
      "index": "exampleIndex",
      "uri": "http://www.example.com",
      "code": "example",
      "indexInfo": [
            {
                  "language": "cs",
                  "state": "IN_PROGRESS/COMPLETED/FAILED",
                  "label": "Lorem"
            }
      ]
}
```

Code 5: Struktura fulltextového indexu. Přidání pole "label"

```
{
      "_source": {
            "cs": "Lorem ipsum dolor sit amet",
            "label_cs": "Lorem",
            "code": "example",
            "uri": "http://www.example.com",
            "mts": "2018-04-13T09:21:44.835"
      }
}
```

Elasticsearch podporuje tzv. MultiMatchQuery, díky které je možné vyhledávat přes více polí a díky tomu dokáže zobrazit i objekty, které v případě obyčejného dotazu vůbec zobrazené nebyly, nebo je zobrazí s odlišnou relevancí.

Samotná relevance se může výrazně lišit i v závislosti na specifičnosti daného popisného pole. Čím specifičtější pole je, tím relevantnější daný objekt bude ve výsledkové sadě. Odlišnost relevance v závislostech na jednotlivých popiscích je vidět v Tab. 3, ze které vyplývá, že score je do značné míry ovlivněno celkovým počtem dokumentů v indexu, proto je potřeba dané dodatečné pole dále modifikovat a získat tak lepší kontrolu nad výsledkovou sadou. Více je popsáno v kapitole Rescore.

Tab. 3: Relevantnost výsledků v závislosti na specifičnosti popisku a celkového počtu dokumentů.

		Výsledek					
Počet dokumentů	Popisek	Počet stránek obsahujících popisek	Score	Počet stránek obsahujících popisek	Score	Počet stránek obsahujících popisek	Score
10	"Lorem"	1	2.356	2	1.812	1	1.816
	"Lorem Ipsum"	1	2.154	1	1.646	2	1.656
	"Example Label"	8	0.046	7	0.046	7	0.046
1000	"Lorem"	1	9.484	2	8.951	1	8.951
	"Lorem Ipsum"	1	8.613	1	8.129	2	8.129
	"Example Label"	998	4.90E-04	997	4.90E-04	997	4.90E-04

6.4.1.2 Boost hodnota

V situaci, kdy je potřeba objekt ve výsledkové sadě vždy upřednostnit před ostatními, lze jeho score dále přepočítat na základě definované hodnoty. Takovou hodnotou může být například popularita (inkrementována při každém výběru daného objektu z výběrové sady) nebo statická hodnota přiřazená objektu, která říká, že daný objekt je důležitější než ostatní a ve výsledkové sadě to musí být uváženo.

Pro případ modifikace score na základě číselné hodnoty zaindexované v poli objektu, Elasticsearch přináší tzv. “functionScoreQuery”. FunctionScoreQuery dokáže modifikovat score různými funkcemi. Každá funkce přijímá výsledkovou sadu z předchozí query a každému objektu z ní dokáže různými způsoby upravit score. Jedním ze způsobů je například tzv.

“field_value_factor”, který převezme indexovanou hodnotu daného pole a tou podle definice upraví původní score. Je možné definovat, jak velkou váhu bude mít hodnota v daném poli vzhledem k velikosti hodnoty. Elasticsearch přináší více způsobů modifikace score vedle již zmíněného “field_value_factor”. Dá se použít vlastní funkce, náhodné číslo nebo složitější funkce rozkladu jako jsou například Gaussova funkce nebo lineární funkce. Tyto funkce hodnotí dokumenty v závislosti na vzdálenosti hodnoty od původně uživatelem zadané hodnoty a hodí se například při udělování relevance objektům vzhledem ke vzdálenosti od uživatelem zadaného místa.

Funkce s modifikátorem typu “field_value_factor” je pro optimalizaci výsledkové sady, vzhledem k zaindexovaným hodnotám v polích “boost” a “popularity”, použita i v aplikaci uuFulltextovéVyhledávání. Hodnota “boost” obsahuje statickou hodnotu, která určuje důležitost objektu vzhledem k ostatním a hodnota v poli “popularity” říká, kolikrát byl daný objekt z výsledkové sady vybrán. Struktura objektu v Mongo databázi po přidání polí “boost” a “popularity” je zobrazena na ukázce Code 6, pro fulltextový index na ukázce Code 7. Jako způsob výpočtu váhy hodnoty “field_value_factor” modifikátoru je v případě pole “popularity” použita funkce Log1P. Díky této funkci, jak vyplývá z Tab. 4, má růst grafu výsledného score logaritmický průběh. To znamená, že změny při nižších hodnotách v poli “popularity” mají na výsledné score dokumentu větší vliv než při hodnotách vyšších. Další možností pro logaritmický průběh nárůstu výsledného score je použití Ln modifikátoru hodnoty. Tento způsob má však větší dopad na výsledné score, než je v případě aplikace uuFulltextovéVyhledávání žádoucí. Pole “boost” přímo určuje důležitost daného objektu, proto není žádoucí, aby váha této hodnoty byla nějakým způsobem upravována a žádná funkce není tedy modifikátoru typu “field_value_factor” přiřazena.

Code 6: Struktura objektu v mongo databázi. Přidání polí "boost" a "popularity"

```
{
      "index": "exampleIndex",
      "uri": "http://www.example.com",
      "code": "example",
      "boost": 1.0,
      "popularity": 1.0,
      "indexInfo": [
            {
                  "language": "cs",
                  "state": "IN_PROGRESS/COMPLETED/FAILED",
                  "label": "Lorem"
            }
      ]
}
```

Code 7: Struktura fulltextového indexu. Přidání pole "boost" a "popularity".

```
{
      "_source": {
            "cs": "Lorem ipsum dolor sit amet",
            "label_cs": "Lorem",
            "code": "example",
            "uri": "http://www.example.com",
            "mts": "2018-04-13T09:21:44.835",
            "boost": 1.0,
            "popularity": 1.0
      }
}
```

Tab. 4: Růst score podle hodnoty v závislosti na modifikátoru.

Hodnota	Modifikátor hodnoty		
	Žádný	Ln	Log1P
1	1.052	0.789	1.052
5	2.761	1.424	1.096
10	4.734	1.697	1.199
50	20.515	2.332	1.46
100	40.242	2.605	1.579
1000	395.32	3.514	1.972

6.4.1.3 Rescore

Díky tomu, že Elasticsearch používá dotazovací jazyk Query DSL, který podporuje skládání dotazů, existuje celá řada možností, díky kterým je možné kromě váhy jednotlivých dotazů výsledné score modifikovat. Pro dosažení pro danou aplikaci opravdu optimálních výsledků je možné zřetězit řadu dotazů uvnitř tzv. "rescore" query. Ta obsahuje pole dalších, obvykle více specifičtějších dotazů, jako je například "phrase" query, která vyhledává přímo zadanou frázi bez možnosti rozdělení fráze na jednotlivá slova nebo "phrase" query s definovaným "slop" atributem, jenž se projevuje na relevantnosti podle toho, jak jsou dané tokeny z vyhledávané fráze v textu od sebe navzájem vzdálené.

Tato optimalizační metoda nemá vliv na počet nalezených objektů ve výsledkové sadě, ale dokáže přepočítat přiřazené score objektů vrácených jako odpověď na prvotní dotaz a tím tak dosáhnout požadovaného setřídění dané výsledkové sady.

6.4.2 Analýza textu

Analýza textu je z pohledu optimalizace podstatným krokem. Reálné dokumenty se stanou dohledatelnými po svém zaindexování, což je proces ukládání dat do invertovaného indexu, kterému předchází právě proces analýzy.

Veškerá snaha o optimalizaci relevantnosti objektů takovým způsobem, aby byl hledaný výsledek co možná nejvýše ve výsledkové sadě, by mohla být zbytečná, pokud se hledané objekty do výsledkové sady ani nedostanou.

Hlavní myšlenkou optimalizace fáze analýzy je tedy zajištění zpracování textu takovým způsobem, aby byl invertovaný index naplněn co možná nejobecnějšími tokeny.

6.4.2.1 Optimalizace analyzátorů

Analyzátor je objekt, který je přiřazen každému jednotlivému poli dokumentu (je použit defaultní, přenastavený analyzátor, pokud v mapovacím dokumentu indexu není definováno jinak) a obsahuje v sobě definici procesu, který se stará o předpřípravu textu.

Ta spočívá v rozdělení daného textu na pole tokenů. Pole tokenů je dále možné na základě definovaných filtrů upravit tak, aby obsahovalo tokeny pouze v takovém tvaru, díky kterému budou co nejsnáze dohledatelné. Ideálně by měly být v základním tvaru a bez interpunkce (lze použít matematické algoritmy nebo slovníková pravidla pro hledání kořenů slova) nebo navíc ještě rozděleny po částech, tzv. “n-gramech”. To umožňuje nalezení slova i v případě, že je zadána jen jeho část. Pole tokenů by mělo dále obsahovat co možná nejméně tzv. “stopwords”, což jsou slova pro daný jazyk hojně se vyskytující a pro vyhledávání nepodstatná. Může se jednat například v českém jazyce o předložky, spojky nebo v angličtině členy apod. Tato slova by mohla mít z důvodu své četnosti jednak vliv na výkon, ale zároveň by mohla mít podstatný vliv na relevantnost výsledků.

Jak již bylo zmíněno, pro každé pole je v základním nastavení použit defaultní analyzátor, který ve svém nastavení používá standartní tokenizer, který dělí text na tokeny pokaždé, když narazí na mezeru, pomlčku, podtržítko nebo jiný speciální znak. Na pole tokenů je potom v základním nastavení aplikován pouze filtr, který všechny tokeny převede na malá písmena.

Optimalizace analýzy textu tedy spočívá v přenastavení základního chování analyzátorů, jejich tokenizerů a použití a posloupnosti jednotlivých filtrů.

6.4.2.2 Optimalizace tokenizerů

Tokenizer je objekt, který předepisuje chování analyzátoru při prvotním zpracování textu, a jeho rozdělení na pole tokenů, se kterým je dále pracováno. Jeho optimalizace spočívá v zajištění správné identifikace jednotlivých tokenů, čehož lze dosáhnout několika způsoby. Lze vybrat z řady již předdefinovaných tokenizerů nebo lze vytvořit tokenizer s vlastním chováním a využít tak různé metody pro rozdělení textu na tokeny. Například je možné využít regulárních výrazů pro rozdělení frází psaných ve formátu “CamelCase”.

Další metodou, jak optimalizovat pomocí tokenizerů, je pomocí rozdělení tokenů po částech, tzv. “n-gramech”, čehož se využívá pro odhad kompletní fráze po jejím neúplném zadání.

Po použití n-gramu s minimální velikostí tokenu 3 a maximální velikostí 5 se do pole tokenů pro frázi například “index” uloží tokeny “ind”, “inde”, “index”. Po následném vyhledání nekompletní fráze “ind” index dokáže zareagovat a vrátit tak shodu s dokumentem, který v textu obsahuje “index” nebo jiné fráze začínající na danou posloupnost znaků.

6.4.2.3 Optimalizace filtrů

Analyzátor může obsahovat pole filtrů. Na všechny tokeny vytvořené pomocí tokenizeru, jsou aplikována pravidla v posloupnosti, jak jsou jednotlivé filtry uvedeny. Hlavním účelem filtrů je upravit tokeny na co možná nejobecnější tvar, a umožnit tak jejich vyhledání i po zadání nepřesné fráze.

Hlavním optimalizačním cílem je vybrat správné token filtry a z důvodu zajištění optimálního chování a výkonu dané filtry využít ve správném pořadí.

Na aplikaci uuFulltextovéVyhledávání byly otestovány filtry v pořadích, jež jsou vidět na ukázkách Code 8 a Code 9. Výsledky pro indexování a vyhledávání pro jednotlivé analyzátory jsou zaneseny do Tab. 5, která znázorňuje čas, za který byly stejné dotazy zpracovány, a počet výsledků, které byly navráceny. Pro daný příklad bylo zaindexováno 37 stránek reálné aplikace a následné vyhledání frází podle kategorií.

Z výsledků jsou patrné malé odlišnosti ve výkonu ve prospěch algoritmického zpracování. Hlavním rozdílem jsou ale navrácené výsledky, kde po vyhledání netypické fráze “zaindexování” byl v případě slovníkového zpracování navrácen pouze jeden výsledek. To je z důvodu, že český slovník slovo “zaindexovat” nezná a není tedy možné ho převést na základní tvar. Proto není možné nalézt v invertovaném indexu ani slova “zaindexovat”, “zaindexováno”, apod. Jednak z důvodu výkonu, ale hlavně z důvodu, že služba předpokládá obsah, kde se budou

hojně vyskytovat cizí slova, která český slovník nezná, je do aplikace implementováno nastavení podle Code 8 a tedy za použití algoritmického hledání kmenů slov.

V případě analyzátoru používajícího algoritmické zpracování jsou postupně použity následující filtry. Filtr “lowercase”, který všechny tokeny převede na malá písmena. Filtr “czech_stop”, který má za cíl z pole tokenů odstranit všechna česká “stopwords”, což jsou slova, která jsou pro daný jazyk nepodstatná a není třeba mezi nimi vyhledávat nebo je dále zpracovávat. Pro hledání základního tvaru slova je použit obecný “czech_stemmer” token filtr, který využívá proces stemantizace s algoritmem stemmer. Ten se využívá pro nalezení kmene slova. Obvykle ale kmen slova není správným kmenem, proto se ve výsledkové sadě jako odpověď na dotaz mohou objevit i shody, které nebyly hledány. Další nevýhodou gramaticky špatného kmene slova je to, že není možné použít slovník synonym, protože ten hledá synonyma podle základního správného tvaru slova. Stemantizace ale na rozdíl od lemantizace (nalezení základního tvaru slova podle slovníkem daných pravidel pro daný jazyk) zpracuje i slova, která nejsou gramaticky správně nebo v daném jazyce neexistují a to bylo v případě aplikace uuFulltextovéVyhledávání vyhodnoceno jako podstatnější faktor než správný tvar s možností hledání synonym, proto byl do aplikace implementován právě tento přístup. Dalším použitým filtrem je “asciifolding”, což je v Elasticsearch předdefinovaný filtr, který má za cíl převést písmenné, číselné nebo speciální znaky, které nejsou mezi prvním 127 znaky ascii tabulky, na jejich ASCII equivalent. V poslední řadě je použit filtr “unique_on_same_position”, jehož cílem je odstranit duplicitní tokeny, které během analýzy textu vzniknou na stejné pozici.

Code 8: Algoritmické hledání kmenů slov

```
"custom_czech_stemmer_analyzer": {
      "type": "custom",
      "tokenizer": "standard"
      "filter": [
            "lowercase",
            "czech_stop",
            "czech_stemmer",
            "asciifolding",
            "unique_on_the_same_position"
      ]
}
```

V případě použití analyzátoru s definicí podle Code 9, je použit “czech_hunspell” token filtr. Czech_hunspell je označení pro filtr, který k nalezení základního tvaru slova využívá slovník

všech slov daného jazyka a definici pravidel pro jejich skloňování a časování. Při využití tohoto způsobu je zajištěna gramatická správnost a přesnost vyhledávání. Nezpracují se ale slova, která nejsou definována ve slovníku. V poli filtrů je definován dvakrát “czech_stop” filtr. To je z důvodu, že “stopwords” daného jazyka nemusí být definována ve všech tvarech, proto se v prvním pokusu odstraní “stopwords”, která již v základním tvaru jsou, poté jsou tokeny za pomoci “czech_hunspell” filtru převedeny na základní tvar a následně jsou tokeny opět promazány. Místo “asciifolding” filtru je v případě daného analyzátoru použit filtr “icu_folding”, který funguje na podobném principu jako “asciifolding” filtr, ale zahrnuje také specifická pravidla pro daný jazyk. Pro češtinu například rozpozná, že sekvence znaků “c” a “h” znamená “ch” a díky tomu dokáže například výsledky správně setřídit.

Code 9: Slovníkové hledání základního tvaru slov.

```
"custom_czech_hunspell_analyzer": {
      "type": "custom",
      "tokenizer": "standard"
      "filter": [
            "lowercase",
            "czech_stop",
            "czech_hunspell",
            "czech_stop",
            "icu_folding",
            "unique_on_the_same_position"
      ]
}
```

Tab. 5: Porovnání testovaných analyzátorů.

Typ fráze	Fráze	Stemmer (Code 8) analyzer		Hunspell (Code 9) Analyzer	
		Čas (ms)	Nalezené shody	Čas (ms)	Nalezené shody
Popisek	Obsah	92	37	12	37
Titulky na více stárnkách	Vychozi hodnoty	65	19	16	19
Často používané slovo	Nazev	88	22	84	22
Několik používaných slov	nazev hodnoty informace atribut init workspace inicializace model business	85	37	15	37
Netypické slovo	Zaindexování	15	1	9	4

7 Výsledky

Na základě návrhu řešení byly navržené metody optimalizace implementovány a následně otestovány. Pro testování byla zaindexována reálná webová aplikace obsahující softwarovou dokumentaci čítající 50 stránek, na které byly následně kladené dotazy. Tyto dotazy byly rozděleny do různých kategorií podle pravděpodobného užití. Pro testování byl použit předpoklad, že pouze jedna stránka přesně odpovídá danému dotazu. Z toho důvodu měl každý jednotlivý dotaz také definovanou přesnou stránku, která měla být navrácena. Pokud se hledaná stránka ve výsledkové sadě nevyskytovala na prvním místě, byl bez ohledu na pořadí ostatních položek výsledkové sady daný test považován za neúspěšný. Jednotlivé optimalizační metody měly za cíl zajistit, aby stránka byla opravdu nalezena a zároveň aby hledaná stránka ve výsledkové sadě byla zobrazena na prvním místě. Ostatní výsledky byly brány jako nežádoucí, a tedy celý test daného dotazu jako neúspěšný.

7.1 Kategorie dotazů

Dotazy byly rozděleny do jednotlivých kategorií podle možného případu užití. Tyto kategorie a jednotlivé dotazy v nich s přiřazenými správnými stránkami, které měly být aplikací navráceny, byly vybrány na základě analýzy zaindexované webové aplikace z pohledu vývojáře.

7.1.1 Přesné fráze a termíny

Do dané kategorie patří dotazy, které obsahují frázi přesně odpovídající textu nebo části textu na některé ze stránek. Daná stránka by měla být také ve výsledkové sadě navrácena a ohodnocena jako nejlepší možná.

S danou kategorií by měla každá technologie pro fulltextové vyhledávání umět pracovat a bez jakékoli optimalizace by měla vrátit požadovaný výsledek.

7.1.2 Fráze bez diakritiky a/nebo s odlišným slovosledem

Daná kategorie stejně jako v případě kategorie “Přesné fráze a termíny” obsahuje frázi, která odpovídá textu nebo části textu na některé ze stránek. Fráze ovšem neobsahuje přesný text, ale může být zadán bez diakritiky nebo s přeházenými slovy.

Hlavním cílem dané kategorie bylo otestovat, že i přestože nebyla zadána přesná fráze, dokáže aplikace na základě použitých slov ve frázi správně identifikovat, že se jedná právě o tu danou část textu z hledané stránky.

7.1.3 Vyhledávání stránek podle jejich obsahu

Dotazy patřící do dané kategorie bývají zpravidla jednoslovné termíny, maximálně dvou nebo tříslovná terminologická sousloví, jež přesně definují obsah některé z hledaných stránek. Tento obsah odpovídá právě jedné stránce, která má být ve výsledkové sadě navrácena na prvním místě.

Cílem pro aplikaci je identifikovat, že obsah definovaný ve frázi je obsažen právě na jedné stránce, kterou vrátí. Termíny definující obsah stránky, ale mohou být poměrně často používaným termínem na více stránkách, proto je potřeba využít některé optimalizační metody.

7.1.4 Vyhledávání podle obecně používaných termínů

Termíny a soubory termínů patřící do dané kategorie jsou zastoupeny na více stránkách ve větším počtu.

Z pohledu aplikace zde hraje velkou roli řada faktorů. Na jedné straně to může být poměr absolutní četnosti výskytu daných termínů ve všech stránkách ku četnosti daných termínů na jedné konkrétní stránce. Ale ovlivňujícím faktorem může být také rozestup jednotlivých termínů na stránce, v případě zadání fráze s více termíny. Pro aplikaci je obtížné obecně definovat jednu stránku náležící dotazu z této kategorie, ale jednotlivé optimalizační metody se i tohoto cíle snaží dosáhnout.

7.1.5 Fráze s překlepy nebo částečně zadané fráze

Kategorie obsahuje fráze, ve kterých jednotlivá slova nejsou napsána správně. Obsahují v některé části slova překlep nebo nejsou uživatelem, ať už s úmyslem nebo bez, napsána celá.

Aplikace by měla být schopna nalézt podobnost nepřesných nebo nekompletních frází se slovy v textech zaindexovaných stránek, pokud se alespoň částečně podobají, a navrátit tak hledanou stránku.

7.1.6 Tvarosloví a synonyma

Mezi dotazy náležícími této kategorii patří, jak napovídá název kapitoly, fráze, které obsahují slova v nejrůznějších, ale správných tvarech nebo je namísto hledaného slova použito některé z jeho synonym.

Ačkoli existuje řada přístupů, které dokáží efektivně pracovat s tvaroslovím, ne všechny z těchto přístupů jsou zcela kompatibilní s přístupy, které se používají k nahrazování synonymy. Tato myšlenka je detailněji rozvedena v návrhu řešení v kapitole “Optimalizace filtrů”.

7.1.7 Obecné dotazy

Do dané kategorie patří dotazy, které uživatel položí, pokud přesně neví, co hledat. Může sem patřit otázka nebo soubor termínů, které společně dohromady nemusí tvořit smysluplnou frázi, ale které uživatele zrovna napadnou v souvislosti s hledaným předmětem.

7.2 Použité optimalizace

Podle kapitoly “Návrh řešení” byly použité optimalizace rozdělené do 4 kategorií. Jednotlivé optimalizace na sebe navzájem navazují, a tedy každá další optimalizace obsahuje také optimalizaci předešlou.

7.2.1 Bez optimalizace

Tento stav aplikace slouží jako funkční základ, vhodný pro budoucí porovnání výsledků po implementování jednotlivých optimalizačních metod. Aplikace v této fázi používá základní nastavení indexu a základní vyhledávací dotaz bez jakékoli optimalizace.

7.2.2 Optimalizace analýzy textu

Daná metoda optimalizace vychází z návrhu optimalizace navržené v návrhu řešení v kapitole “Analýza textu”. Je zde tedy implementován vlastní analyzátor pro přizpůsobení zpracování a zaindexování textu.

7.2.3 Popisek stránky

Daná kategorie obsahuje implementaci optimalizace na základě definovaného obsahu stránky. Relevance výsledků je tedy modifikována na základě shody hledané fráze nebo části hledané fráze s indexovaným obsahem dané stránky.

7.2.4 Boost hodnoty

Podstata této metody optimalizace, podle kapitoly "Boost hodnota" v návrhu řešení, je v přepočítání relevance výsledků s ohledem na zaindexované hodnoty každé stránky a tím dosažení lepších výsledků. Implementace tedy modifikuje dotaz odesílaný na server za použití různých modifikátorů relevance.

7.2.5 Rescore

Rescore metoda optimalizace slouží, podobně jako boost hodnoty, pouze k rekalkulaci relevance. Na rozdíl od předešlých metod optimalizace ale jeho přínos není z hlediska pořadí výsledkové sady tak výrazný, jelikož nedokáže pořadí výrazněji ovlivnit. Jeho hlavní přínos spočívá v utvrzení daného pořadí, které je vráceno po implementaci předchozích optimalizačních metod a tím zamezení nežádoucímu přeskupení výsledkové sady. Je zde přiřazena váha jednotlivým optimalizačním krokům, podle které je relevance dodatečně upravena.

7.3 Výsledky navržených optimalizačních metod

Cílem bylo sledovat chování a výskyt hledané stránky ve výsledkové sadě po zadání všech dotazů všech kategorií do aplikace po implementování jednotlivých, navzájem na sebe navazujících optimalizačních metod.

Výsledky byly následně zaneseny do grafů pro lepší vizualizaci a byly na nich provedeny různé statistické analýzy. Na základně výsledků těchto analýz vyplynulo, že navržené metody optimalizace splňují svůj účel a rozdíly ve výsledcích mezi některými optimalizačními kroky jsou statisticky významné na hladině $\alpha = 0.05$.

7.3.1 Kruskal-Wallisův test

Pro otestování, zda optimalizační kroky fungují, a tedy požadovaný výsledek jednotlivých dotazů se po provedení optimalizací dostal ve výsledkové sadě na vyšší místo, byly výsledky zaneseny do krabicového grafu na Obr. 4, na kterém je vidět, že jednotlivé po sobě jdoucí optimalizační kroky plní svůj účel a optimalizacemi 1 - 3 se skutečně upravuje výsledková sada a hledaná stránka se přibližuje prvnímu místu.

Na první pohled se může zdát, že mezi optimalizačními kroky 0 a 1 nedochází k vylepšení výsledků, spíše naopak. To je zapříčiněno počtem navrácených výsledků, který je díky analýze

textu v případě “1 Optimalizace analýzy textu” daleko větší než v případě “0 Bez optimalizace”. K vylepšení výsledků z hlediska pozice hledané stránky ale v tomto případě skutečně dochází jen nepatrně, jelikož tato optimalizace má za cíl pouze vrátit i výsledky pro nepřesně specifikovaný dotaz.

K výraznějšímu vylepšení výsledků dochází až při optimalizaci “2 Popisek stránky”, která snižuje rozptyl mezi prvním a třetím kvartilem souboru pozic hledaných stránek a medián posouvá až na hranici 1. Čili absolutní většina výsledků se ve výsledkové sadě opravdu vyskytuje na první pozici.

Optimalizací “3 Boost hodnoty” je dosaženo opět nepatrného zlepšení, kdy se třetí kvartil souboru pozic hledaných stránek zmenšil, a je tak zřejmé, že další hledané stránky se posunuly blíže k pozici 1 ve výsledkové sadě.

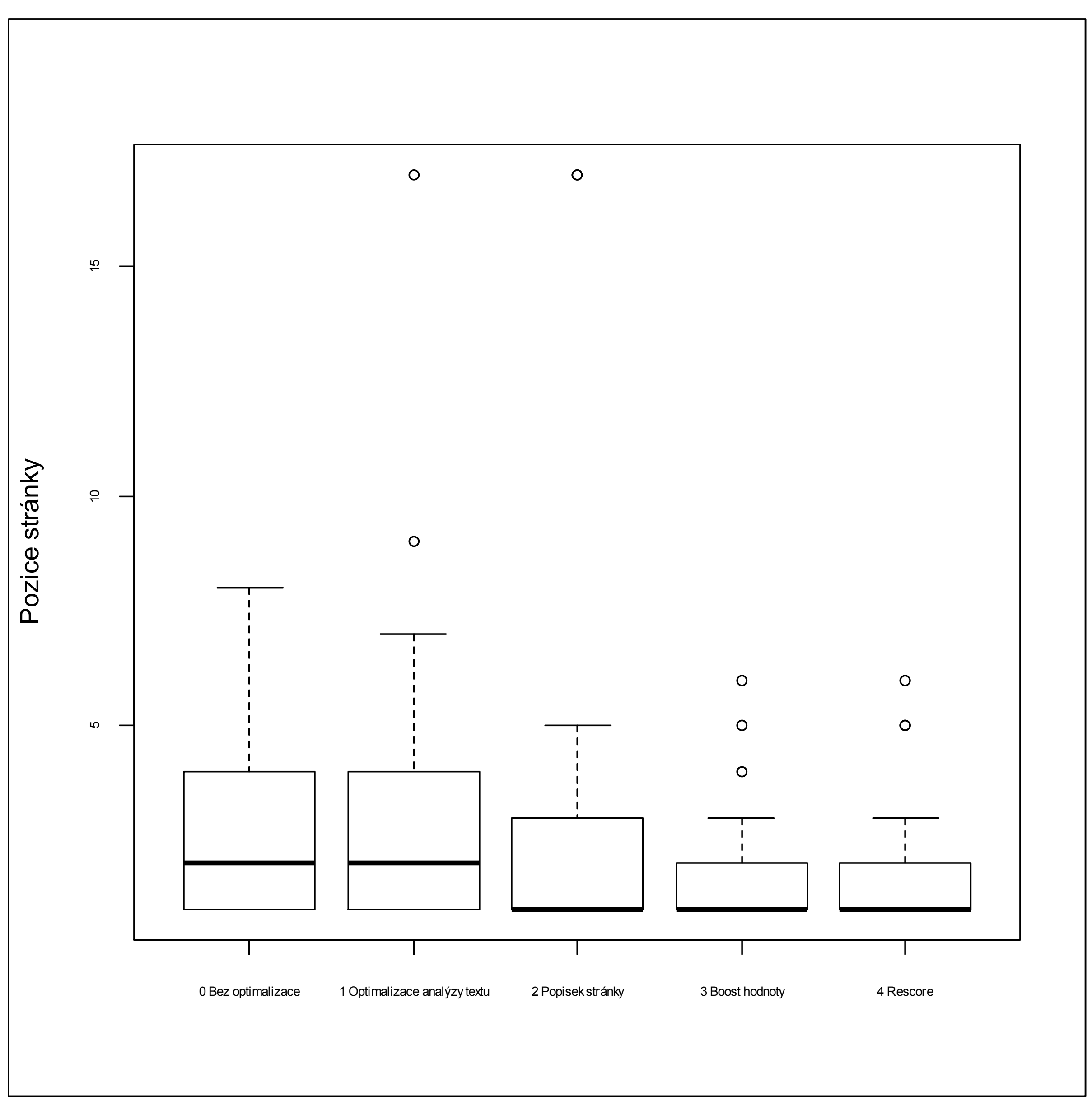
Pro vyhodnocení statistické významnosti výsledků byl na data použit Kruskal-Wallisův test, podle kterého jsou výsledky optimalizací statisticky významné s $p = 0.0003$.

Kruskal-Wallisův test říká, zda je alespoň jedna použitá optimalizace z hlediska výsledků významná. Neplyne z něj ale, která z použitých optimalizací je tou méně významnou nebo právě tou nejvýznamnější.

Ke zjištění statistické významnosti jednotlivých optimalizačních kroků byly použity Dunnovy post-hoc testy ke Kruskal-Wallisově testu. Jejich výsledky jsou zaneseny do Tab. 6 a symbolem * jsou označeny p-hodnoty, které jsou brány jako statisticky významné. Je potřeba brát v úvahu, že, jak již bylo zmíněno, jednotlivé optimalizační kroky na sebe navazují a každá následující optimalizační metoda obsahuje také všechny předešlé. Z výsledků je tedy patrné, že první významnější optimalizací z hlediska pozice hledané stránky ve výsledkové sadě je optimalizace “2 Popisek stránky”. Další optimalizace, jako jsou “3 Boost hodnoty” a “4 Rescore” nemají z hlediska pozice hledané stránky takový význam.

Tab. 6: Dunnovy post-hoc testy ke Kruskal-Wallisově testu

Optimalizace	**0 Bez optimalizace**	**1 Analýza textu**	**2 Popisek stránky**	**3 Boost hodnoty**
1 Analýza textu	**0.4209**			
2 Popisek stránky	**0.0070***	**0.0088***		
3 Boost hodnoty	**0.0012***	**0.0014***	**0.2695**	
4 Rescore	**0.0004***	**0.0004***	**0.161**	**0.3535**



Obr. 4: Krabicový graf pozice hledané stránky po implementaci optimalizačních kroků.

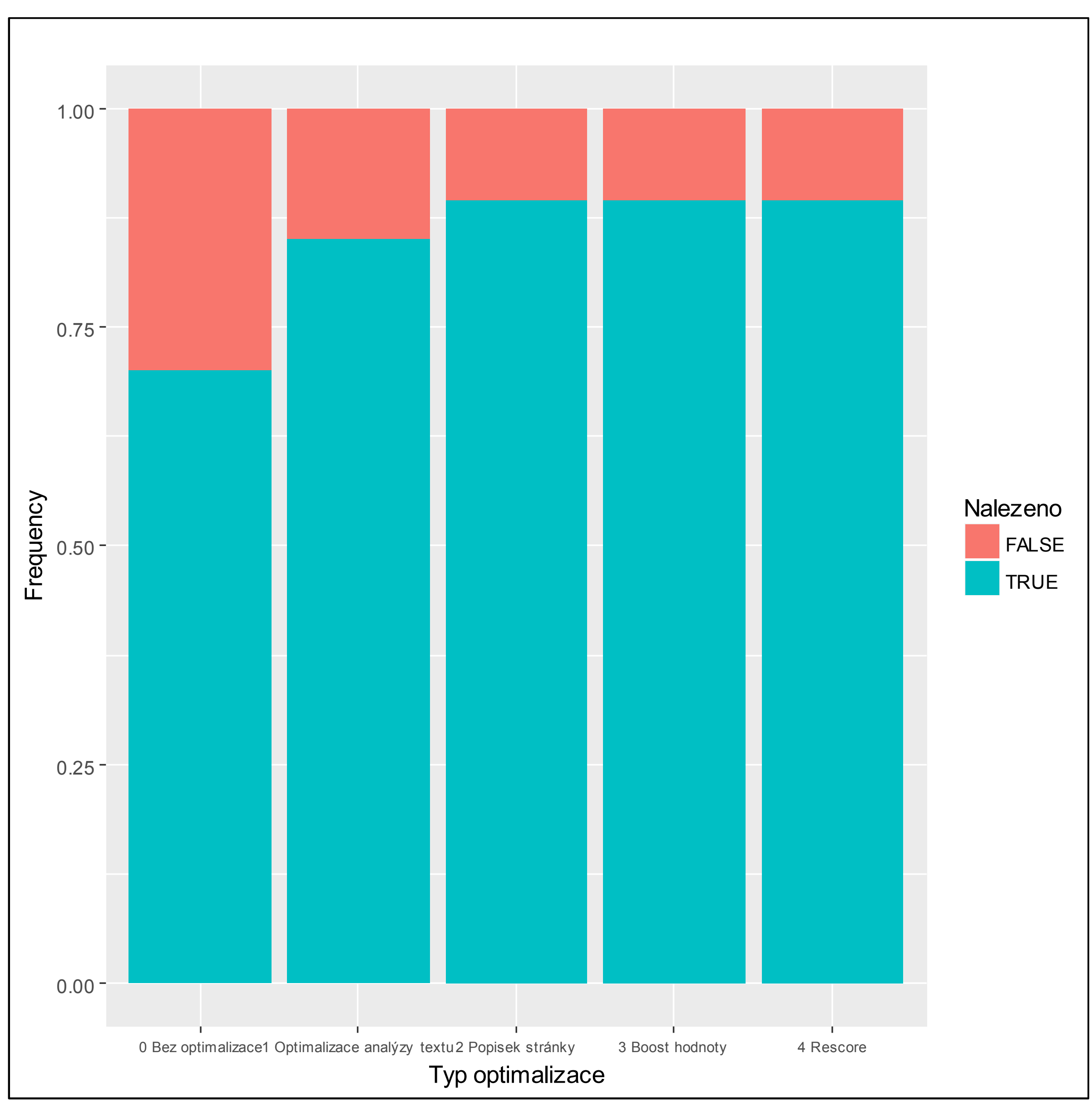
7.3.2 Fisherův exaktní test

Dalším sledovaným kritériem, vedle pozice hledané stránky ve výsledkové sadě, byl údaj, zda byla hledaná stránka vůbec nalezena. Výsledky jsou znázorněny v grafu na Obr. 5. Metoda "1 Optimalizace analýzy textu" je v tomto směru nejpodstatnější optimalizací. K nepatrnému zlepšení podílu nalezených stránek dochází také v případě přidání optimalizace "Popisek stránky". V dalších optimalizačních metodách se více stránek ani vyhledat nemůže, jelikož tyto optimalizace mají jako primární účel pouze rekalkulaci relevantnosti výsledků.

Přes všechny optimalizace existuje ale několik stránek, které, jak plyne z grafu, přesto nebyly nalezeny. To je z důvodu nedokonalé analýzy textu, pro kterou bylo rozhodnuto podle kapitoly

“Analýza textu” v návrhu řešení, a proto pro některé dotazy z kategorie “Tvarosloví a Synonyma” není možné nalézt správný výsledek.

Pro vyhodnocení statistické významnosti optimalizace z hlediska počtu navrácených výsledků byl použit Fisherův exaktní test, podle kterého optimalizace s nalezením stránky významně souvisí s p = 0.0122.



Obr. 5: Graf podílu nalezených stránek.

8 Závěr

Cílem práce bylo seznámení se s celkovou problematikou fulltextového vyhledávání, zvolení správné technologie, a nakonec její implementace a optimalizace pro vyhledávání informací v rámci webové aplikace.

Z počátku práce je část textu věnována seznámení se s obecnou problematikou fulltextového vyhledávání. Je zde zmíněno, jak se vyhledávalo v minulosti a jaké přístupy a technologie se využívají dnes. Řada dnešních technologií je následně porovnána a na základě tohoto porovnání je vybrána technologie Elasticsearch, které je věnován zbytek práce. Jednak je zde popsáno, na jakých přístupech tato technologie funguje, ale jsou na ní také demonstrovány jednotlivé navržené optimalizační metody, které jsou také hlavním výstupem práce. Tyto obecné metody optimalizace ovlivňují pozici hledané stránky ve výsledkové sadě takovým způsobem, že hledané stránky se po jejich aplikaci dostanou ve výsledkové sadě do popředí. Ve stavu, kdy je hledaná stránka na prvním místě, je výsledková sada brána za optimální, a tohoto cíle měla práce dosáhnout. Výsledky testů ukázaly, že optimalizační metody mají na pozici hledané stránky ve výsledkové sadě podstatný vliv. Na základě tohoto zjištění lze tedy říci, že navržené metody jsou správným přístupem, který lze převzít a aplikovat i pro širší spektrum aplikací.

Navržené optimalizační metody jsou již použity v reálné aplikaci, kde až doposud fulltextové vyhledání zcela chybělo a jeho nasazení tak přineslo značné zkrácení času stráveného hledáním informací.

Přestože je takto navržená aplikace funkční a v rámci možností také optimalizovaná, existují stále oblasti, ve kterých je možné na vývoji pokračovat. Jedním z námětů na vylepšení mohou být například synonyma. Tato oblast je sice v práci zmíněna a je v ní také vysvětleno proč v danou chvíli tuto funkčnost aplikace nepodporuje, ale je to jednoznačně návrh na zlepšení. Po dostatečném sběru dat by také bylo možné data zpracovat a vytvořit tak sadu dotazů obsahujících cizí slova, která český slovník doposud neznal, a doplnit jej. Tímto přístupem by bylo možné i na cizí slova aplikovat lemantizaci a tím tak umožnit následné porovnání se slovníkem synonym.

9 Zdroje:

[1] HUBERMAN, Bernardo A.; ADAMIC, Lada A. Internet: growth dynamics of the world-wide web. *Nature*, 1999, 401.6749: 131.

[2] comScore Releases February 2015 U.S. Desktop Search Engine Rankings - comScore, Inc. *comScore helps clients measure what matters to make cross-platform audiences and advertising more valuable* (online). Copyright © 2018 comScore, Inc. (citace duben, 23., 2018). Přístup z internetu: URL:https://www.comscore.com/Insights/Rankings/comScore-Releases-February-2015-US-Desktop-Search-Engine-Rankings

[3] ARLINGTON, M. *Google's misleading blog post: the size of the web and the size of their index are very different*, 2008 (citace listopad, 11., 2017). Přístup z internetu: URL:http://techcrunch.com.

[4] JANANI, R.; VIJAYARANI, S. An Efficient Text Pattern Matching Algorithm for Retrieving Information from Desktop. *Indian Journal of Science and Technology*, 2016, 9.43.

[5] GORMLEY, Clinton a Zachary TONG. *Elasticsearch: the definitive guide*. Sebastopol, CA: O'Reilly, 2015. ISBN 978-1-449-35854-9.

[6] BALIPA, Mamatha; BALASUBRAMANI, R. Search Engine using Apache Lucene. *International Journal of Computer Applications*, 2015, 127.9: 27-30.

[7] Goyvaerts Jan, *Regular Expressions*, 2017. (on-line) (citace březen, 10., 2018) Přístup z internetu: URL:http://www.regular-expressions.info/tutorial.html.

[8] KAPITANOVA, Krasimira; SON, Sang H. Machine learning basics. *Intelligent Sensor Networks: The Integration of Sensor Networks, Signal Processing and Machine Learning*, CRC Press,2012, 13.

[9] *Algolia technical documentation*. (on-line) 2016 (citace květen, 10., 2018). Přístup z internetu: URL:https://www.algolia.com/doc.

[10] *Database-Engines Ranking*. (on-line) 2018 (citace květen, 10., 2018). Přístup z internetu: URL:https://db-engines.com/en/ranking/search+engine.

[11] SEDGEWICK, Robert. Algorithms. *Series in Computer Science*. 1984.

[12] PEYMANDOUST, Armita; SIMUNIC, Tajana; DE MICHELI, Giovanni. Low power embedded software optimization using symbolic algebra. In: *Proceedings of the conference on Design, automation and test in Europe*. IEEE Computer Society, 2002. p. 1052.

[13] ELLIOTT, Alan C.; HYNAN, Linda S. A SAS® macro implementation of a multiple comparison post hoc test for a Kruskal–Wallis analysis. *Computer methods and programs in biomedicine*, 2011, 102.1: 75-80.

[14] Dunn's test: Definition . *Statistics How To: Elementary Statistics for the rest of us!* (on-line). Copyright © 2018 (citace duben, 23., 2018). Přístup z internetu: URL:http://www.statisticshowto.com/tukey-test-honest-significant-difference/

[15] Matematická biologie učebnice: Fisherův exaktní test. *Matematická biologie učebnice: Úvod* (on-line) (citace duben, 23., 2018). Přístup z internetu: URL:http://portal.matematickabiologie.cz/index.php?pg=aplikovana-analyza-klinickych-a-biologickych-dat--analyza-a-management-dat-pro-zdravotnicke-obory--testovani-hypotez-o-kvalitativnich-promennych--fisheruv-exaktni-test

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2017/2018

Studijní program: Aplikovaná informatika
Forma: Prezenční
Obor/komb.: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Kratochvíl Lukáš	Vrcha 62, Sobětuchy - Vrcha	I1500383

TÉMA ČESKY:

Fulltextové vyhledávání

TÉMA ANGLICKY:

Fulltext Search

VEDOUCÍ PRÁCE:

Mgr. Jan Vaněk, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cíl práce:
Prostudovat algoritmy vyhledávání řetězců, zanalyzovat konkrétní oblast a zvolit či navrhnout systém pro fulltextové vyhledávání ve zvolené oblasti, systém implementovat a otestovat.

Postup prací:
Rešerše literatury
Návrh systému
Implementace
Testování

SEZNAM DOPORUČENÉ LITERATURY:

JANANI, R. a S. VIJAYARANI, 2016. An Efficient Text Pattern Matching Algorithm for Retrieving Information from Desktop. Indian Journal of Science and Technology. 9(43).
HE, Hao, Haixun WANG, Jun YANG a Philip S. YU, 2007. BLINKS: ranked keyword searches on graphs. In: Proceedings of the 2007 ACM SIGMOD international conference on Management of data. B.m.: ACM, s. 305316. ISBN 1-59593-686-6.

Podpis studenta: .... **Datum:** ....

Podpis vedoucího práce: .... **Datum:** ....