ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ DIPLOMOVÉ PRÁCE

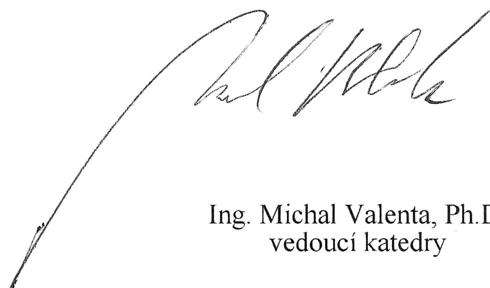
Název:	Aplikace pro analýzu textových zpráv pro potřeby vyšetřovatelů
Student:	Bc. Štěpán Škorpil
Vedoucí:	Ing. Pavel Kordík, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2015/16

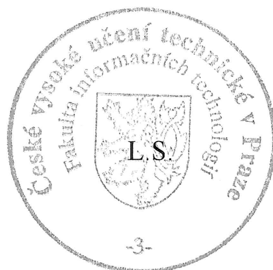
Pokyny pro vypracování

Prostudujte prostředí vyšetřovatelů závažné drogové kriminality a způsob jejich práce s textovými zprávami. Navrhněte a implementujte aplikaci, která usnadní a automatizuje zpracování a vyhodnocování textových zpráv v českém jazyce. Použijte vhodné open source nástroje pro analýzu a vyhledávání v textu, navrhněte a otestujte vhodné uživatelské rozhraní a způsob prezentace informací. Funkčnost aplikace demonstrujte na sadě českých textů podobných zpracovávaným zprávám a celou aplikaci pečlivě zdokumentujte.

Seznam odborné literatury

Dodá vedoucí práce.



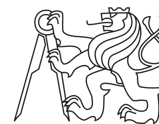


Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof.Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 12. ledna 2015

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra teoretické informatiky



Diplomová práce

Aplikace pro analýzu textových zpráv pro potřeby vyšetřovatelů

Bc. Štěpán Škorpil

Vedoucí práce: Ing. Pavel Kordík, Ph.D.

4. května 2015

Poděkování

Chtěl bych poděkovat všem, kteří mě podporovali během tvorby diplomové práce, ale i během celého studia. Děkuji především celé rodině, všem přátelům a zvlášť své přítelkyni za trpělivost, které se mi dostávalo.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 4. května 2015 ....

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2015 Štěpán Škorpil. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.

Odkaz na tuto práci

Škorpil, Štěpán. *Aplikace pro analýzu textových zpráv pro potřeby vyšetřovatelů.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Cílem práce je navrhnout a implementovat software pro zpracování textových dokumentů v českém jazyce, který by mohla Policie České republiky využít v boji se zločinem. Výsledkem je první část komplexního řešení, která se zabývá prohledáváním dokumentů, automatickým tříděním do shluků a hledáním podobných dokumentů. Práce se snaží dosáhnout cílů hledáním vhodných algoritmů pro stemizaci českého textu.

Klíčová slova Vyhledávání, Indexace, Stemizace českého textu, Shluková analýza

Abstract

This thesis describes the designing and implementation of a text file processing software in the Czech language which could be used by Police of the Czech Republic to fight crime. Of the complex solution, the first part has been developed here, designated for scanning text documents, clustering them and searching for similar ones. The aims of this thesis have been achieved through searching suitable algorithms for stemming the Czech text.

Keywords Searching, Indexing, Stemming of Czech text, Cluster analysis

Obsah

Úvod **1**

1 Analýza **3**
- 1.1 Analýza požadavků . . . 3
- 1.2 Aktéři systému . . . 5
- 1.3 Případy užití . . . 6
- 1.4 Technologie . . . 9

2 Návrh **15**
- 2.1 Architektura . . . 15
- 2.2 Crawler . . . 16
- 2.3 Uživatelské rozhraní . . . 20

3 Implementace **25**
- 3.1 Solr . . . 25
- 3.2 Crawler . . . 32
- 3.3 Frontend . . . 34
- 3.4 Jetty . . . 35

4 Testování **39**
- 4.1 Test stemmerů . . . 39
- 4.2 JUnit . . . 41
- 4.3 Test použitelnosti . . . 42
- 4.4 Akceptační test . . . 46

Závěr **49**

Literatura **51**

A Seznam použitých zkratek **53**

B Uživatelská příručka **55**
B.1 Požadavky 55
B.2 Instalace 55
B.3 Kompilace 55
B.4 Searcher 56
B.5 Crawler 57
B.6 Frontend 59

C Obsah přiloženého CD **61**

Seznam obrázků

1.1 Diagram případů užití 6

2.1 Diagram nasazení 16
2.2 Diagram procesorových tříd Crawleru 18
2.3 Diagram factory a filter tříd Crawleru 19
2.4 Návrh vyhledávacího uživatelského rozhraní 21
2.5 Návrh vyhledávacího uživatelského rozhraní s rozšířenou nabídkou 21
2.6 Návrh uživatelského rozhraní pro zobrazení dokumentu 23

3.1 Objektový model sestavených procesorových řetězů Crawleru . . . 33
3.2 Snímek UI (Vyhledávač) 36
3.3 Snímek UI (Detail dokumentu) 37

Úvod

Policie České republiky uchovává velké množství dokumentů. Jedná se o výpovědi, oznámení a stručné zprávy týkající se různých případů v čistě textové podobě, bez jakékoliv pevné struktury či atributů, psané v českém spisovném jazyce. Z těchto dokumentů je potřeba seskládat či vytěžit jakékoliv užitečné informace, které by vedly k dopadení pachatelů.

Nástrojů v oblasti získávání různých druhů dat z textu a jejich vizualizace existuje celá řada, mezi nejpoužívanější patří Rapid Miner, R či GATE. Tyto programy většinou disponují nástroji pro zpracování anglického textu, popřípadě pro zpracování textu v jiném světovém jazyce. Horší je situace s nástroji pro zpracování dokumentů v českém jazyce. Existuje několik komerčních řešení, avšak žádný jednoduše použitelný open source program.

Z těchto důvodů se zadavatel práce obrátil s problémem na Fakultu informačních technologií ČVUT. Na následujících stránkách této práce se pokusím navrhnout a implementovat nástroje pro zpracování českého textu a aplikaci, která pomůže vyšetřovatelům v jejich práci.

Kapitola 1

Analýza

1.1 Analýza požadavků

V první řadě jsem musel zjistit, co vyšetřovatelé od nové aplikace očekávají. Po konzultaci se zadavatelem práce jsem zjistil následující.

Policie uchovává data jako RTF dokumenty v MSSQL databázi. Zadavatel je schopen exportovat tyto dokumenty do plain textových souborů. Právě takto získané soubory chce zpracovávat. Vstupem pro software tedy bude jeden či více adresářů v souborovém systému s textovými soubory.

Soubory jsou čistě textové, bez jakékoliv pevné struktury či atributů, psané v přirozeném spisovném českém jazyce. Soudě podle anonymizovaných příkladů, které mi byly poskytnuty, jsou to výpovědi, oznámení, stručné zprávy a další úřední záznamy týkající se různých případů.

První požadavek se vztahuje k vyhledávání. V souborech je třeba najít ty texty, které se týkají daného tématu nebo případu. Bude tedy nutné nasadit či vytvořit fulltextový vyhledávací nástroj, který bude umět indexovat zdrojové texty v adresářové struktuře a poté texty vyhledat pomocí dostatečně robustního dotazovacího jazyka. Nástroj bude muset zvládnout provádět nad českými zdrojovými texty základní operace pro vytěžování dat, tj. tokenizaci a lematizaci nebo stemizaci.

Ve výsledcích hledání by měla být zvýrazněna hledaná slova. Dále by výsledky hledání mělo být možné automaticky seskupit podle shlukové analýzy. Systém by měl být také schopen nabídnout podobné dokumenty k již otevřenému dokumentu.

Aplikace by měla při indexaci vnitřně uchovat celý obsah dokumentu, protože zadavatel počítá s tím, že po zaindexování exportované soubory smaže ze souborového systému počítače.

Dále je třeba ze souborů získat ryzí informace. V první řadě by nástroj měl umět extrahovat z textu entity, kterých se text týká. Tyto entity by pak měly být také indexovány, aby podle nich mohly být texty vyhledány. K vyhledání entit v textu bude zapotřebí dalších, složitějších, nástrojů pro zpracování

textu. Bude k tomu nezbytné zjistit slovní druh jednotlivých slov a také vytvořit či získat rozsáhlé slovníky jmenných entit.

Nejnáročnější funkcí, kterou by software v tomto prostředí měl zvládat, je extrakce souvislostí a vztahů mezi entitami.

Zadavatel požaduje, aby byla aplikace postavena na klient-server architektuře. Složky s dokumenty totiž budou umístěny na jediném počítači a uživatelům by mělo být umožněno s daty pracovat na svých počítačích v síti nezávisle na sobě. Usoudil jsem, že nejlepší volbou bude vytvořit aplikaci webové rozhraní. Je to řešení moderní, které dostojí požadavku klient-server architektury a zároveň s ním odpadá jakákoliv správa aplikace na klientských počítačích.

Aplikace bude nasazena na běžném stolním počítači s operačním systémem Windows. Zadavatel se ale v případě nutnosti nebrání nasazení aplikace ve virtualizovaném linuxovém stroji pod Windows.

Požadavky, které jsou na aplikaci kladeny, by se tedy daly shrnout do následujících bodů.

1.1.1 Funkční požadavky

1. Systém bude indexovat textové soubory z adresářové struktury počítače.
2. Systém bude umožňovat rozšíření o další nezávislý index dokumentů.
3. Systém si bude ukládat celý obsah dokumentů.
4. Systém bude zpracovávat text v české jazyce.
5. Systém bude umožňovat fulltextové vyhledávání v indexovaných dokumentech.
    a) Systém bude k tomuto účelu zvládat tokenizaci českého textu.
    b) Systém bude k tomuto účelu zvládat stemizaci českého textu.
    c) Systém bude umožňovat zobrazit obsah nalezených dokumentů.
    d) Systém bude zvýrazňovat hledaná slova ve výsledcích hledání.
6. Systém bude umožňovat zobrazení podobných dokumentů.
    a) Systém bude k tomuto účelu vyhodnocovat podobnost dokumentů.
7. Systém bude umožňovat shlukovat výsledky hledání do automaticky generovaných kategorií.
    a) Systém bude k tomuto účelu provádět shlukovou analýzu.
    b) Systém bude umožňovat filtrovat výsledky podle zvolených shluků.
8. Systém bude umožňovat rozpoznávat jmenné entity.
    a) Systém bude k tomuto účelu umět rozpoznávat větné členy.

   b) Systém bude k tomuto účelu umět rozpoznávat slovní druhy.
   c) Systém bude k tomuto účelu obsahovat jmenné databáze.

9. Systém bude umožňovat extrakci vazeb mezi entitami.

1.1.2 Nefunkční požadavky

1. Systém bude postaven pouze na open source technologiích.
2. Systém bude postaven na architektuře klient-server jako webová služba.
3. Systém bude nativně nasaditelný pod operačním systémem Windows (není podmínkou).
4. Systém bude rozšiřitelný.

1.1.3 Stanovení první fáze

Definované požadavky jsou velmi rozsáhlé a z části na sebe navazují svými vstupy a výstupy. Je tedy třeba začít od základních požadavků a ty pak rozšiřovat o další možnosti zpracování. Celý definovaný rozsah je nad rámec jedné diplomové práce a tak bylo nutné projekt rozčlenit do několika fází.

První fáze sestává z tvorby vyhledávání v textu, a to z toho důvodu, že je jakýmsi základem dalších operací s textem. Pro fulltextové vyhledávání musí text projít tokenizací, stemizací a následně indexací do databáze. Na produkty těchto základních operací další operace navazují.

První fázi projektu jsem pak ještě doplnil o shlukovou analýzu výsledků vyhledávání a výpočet podobnosti dokumentů, protože s vyhledáváním úzce souvisí.

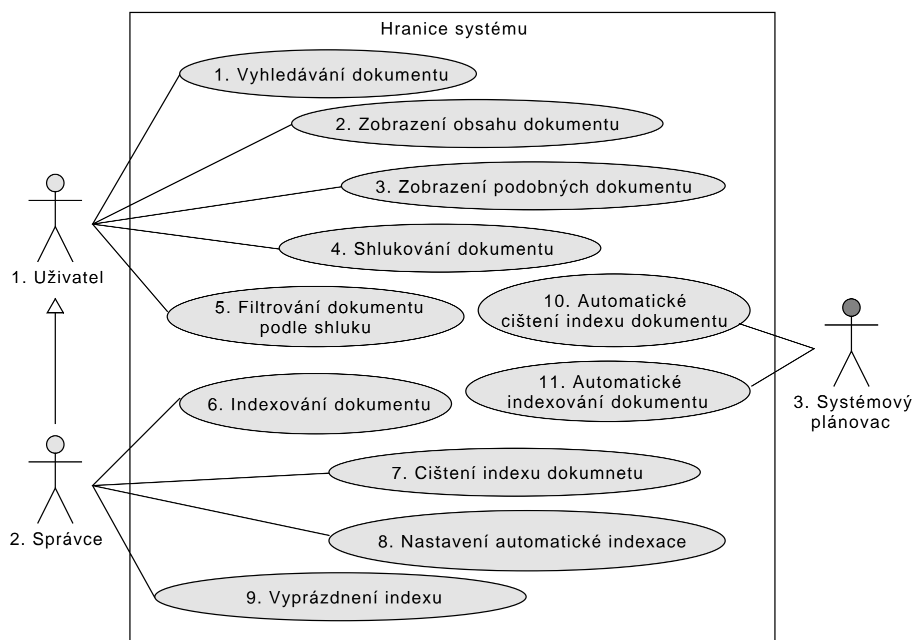
Další části této práce se už budou zabývat jen první fází projektu. V návrhu aplikace pak budu brát na zřetel budoucí možnost rozšíření o další definované požadavky.

1.2 Aktéři systému

Z definovaných požadavků vzešli aktéři a případy užití systému. Obojí je názorně zobrazeno v diagramu 1.1. V systému budou figurovat prakticky jen dvě uživatelské role a jeden neuživatelský aktér — systémový plánovač.

1. Uživatel: Aktér, který bude systém používat pro získávání informací z textu. Všechny scénáře tohoto aktéra mají jako prerekvizitu index již naplněný dokumenty k prohledávání.
2. Správce: Aktér bude moci provádět vše co uživatel, jen navíc bude spravovat index dokumentů. Aktér bude mít plný přístup k serverovému stroji.

3. Systémový plánovač: Některé scénáře mohou být plánovány a spouštěny automaticky systémem. Jsou řízeny nastaveným časem.



Obrázek 1.1: Diagram případů užití

1.3 Případy užití

1.3.1 Vyhledávání dokumentů

Aktéři: 1 Uživatel

Scénář:
1. Uživatel zadá dotaz.
2. Systém zobrazí výsledky dle dotazu.

Výstup: Dokumenty odpovídající dotazu

1.3.2 Zobrazení obsahu dokumentů

Aktéři: 1 Uživatel

Scénář:
1. Uživatel zvolí zdroj a zadá dotaz.
2. Systém zobrazí výsledky dle dotazu.

3. Uživatel ze zobrazených výsledků vybere požadovaný dokument.
4. Systém zobrazí obsah dokumentu.

Výstup: Zobrazený obsah dokumentu

1.3.3 Zobrazení podobných dokumentů

Aktéři: 1 Uživatel

Scénář:
1. Uživatel zvolí zdroj a zadá dotaz.
2. Systém zobrazí výsledky dle dotazu.
3. Uživatel ze zobrazených výsledků vybere požadovaný dokument.
4. Systém zobrazí seznam dalších dokumentů seřazených od nejpodobnějšího k nejméně podobnému

Výstup: Dokumenty, které jsou podobné zvolenému dokumentu, seřazené od nejpodobnějšího k nejméně podobnému

1.3.4 Shlukování dokumentů

Aktéři: 1 Uživatel

Scénář:
1. Uživatel zvolí zdroj a zadá dotaz.
2. Systém zobrazí jak výsledky dle dotazu, tak nalezené shluky výsledných dokumentů.

Výstup: Seznam shluků výsledných dokumentů

1.3.5 Filtrování dokumentů podle shluků

Aktéři: 1 Uživatel

Scénář:
1. Uživatel zvolí zdroj a zadá dotaz.
2. Systém zobrazí výsledky dle dotazu a nalezené shluky výsledných dokumentů.
3. Uživatel vybere jeden či více shluků.
4. Systém zobrazí jen dokumenty patřící do zvolených shluků.

Výstup: Seznam dokumentů v daných shlucích.

1.3.6 Indexování dokumentů

Aktéři: 2 Správce

Scénář:
1. Správce zvolí jeden či více adresářů.
2. Systém v adresářích najde textové soubory a přidá je do indexu.

Výstup: Zaindexované soubory ze zvolených adresářů připravené pro vyhledávání

1.3.7 Čištění indexu dokumentů

Aktéři: 2 Správce

Scénář:
1. Správce spustí čištění indexu.
2. Systém projde dokumenty a ty, které již neexistují v souborovém systému, odstraní z indexu.

Výstup: Index obsahující pouze dokumenty, které mají svou reprezentaci v souborovém systému počítače

1.3.8 Vyprázdnění indexu

Aktéři: 2 Správce

Scénář:
1. Správce spustí vyprázdnění indexu.
2. Systém odstraní všechny dokumenty z indexu.

Výstup: Prázdný index

1.3.9 Nastavení automatické indexace

Aktéři: 2 Správce

Scénář:
1. Správce zvolí interval, adresáře, zda se má indexovat, zda se má čistit index.
2. Systém ve zvolený čas spouští čištění a indexaci, viz aktér 3 Systémový plánovač.

Výstup: Nastavený systémový plánovač

1.3.10 Automatické indexování dokumentů

Aktéři: 3 Systémový plánovač

Scénář:
1. Plánovač nastaví konfiguraci dle nastavení uživatele.
2. Systém pokračuje v indexaci od bodu 2 scénáře 1.3.6 Indexování dokumentů.

1.3.11 Automatické čištění indexu dokumentů

Aktéři: 3 Systémový plánovač

Scénář:
1. Plánovač nastaví konfiguraci dle nastavení uživatele.
2. Systém pokračuje v indexaci od bodu 2 scénáře 1.3.7 Čištění indexu dokumentů.

1.4 Technologie

1.4.1 Surmon

Bylo mi navrženo, abych se pokusil implementovat práci pomocí nástroje Surmon. Prostudoval jsem si tedy dokumentaci aplikace, otestoval si její funkce a zjistil následující.

Surmon je balík modulů pro platformu OBBB[1] — Open Black Box Builder. Platforma OBBB umožňuje sestavit aplikaci pro zpracování dat pomocí propojování funkčních bloků. Blok je atomický kus algoritmu se vstupy a výstupy různých datových typů a s možností nastavení parametrů pro úpravu jeho chování. Každý blok načte data na svých vstupech, zpracuje je a výsledná data zapíše na své výstupy. Bloky pak lze propojovat spojnicemi z výstupu jednoho bloku na vstup jiného — samozřejmě jen tehdy, jsou-li kompatibilní datové typy vstupů a výstupů. Takovouto tvorbou orientovaných grafů bloků a spojnic lze definovat složité algoritmy. Na sestavování bloků má platforma již vytvořený intuitivní editor v podobě plátna, na které se bloky přidávají a kde se propojují. Každý blok má svou reprezentaci v uživatelském rozhraní, pomocí kterého lze nastavit parametry bloku. Vše je implementováno v jazyce Java a postaveno na Platformě NetBeans[2].

Samotný OBBB obsahuje základní sadu bloků pro řízení toku algoritmu, načítání a zápis souborů různých formátů, iteraci nad daty a další funkce. Surmon je pak sada dalších bloků pro operace zpracování textu a text mining. Pro účely této práce je podstatné zejména to, že Surmon obsahuje bloky pro text mining z českých textů.

Software tedy umožňuje sestavit algoritmy pomocí hotových celků bez programátorského zásahu. Sestavený alogritmus pak lze uzamknout heslem, vystavit některé vstupy či parametry nastavení jako uživatelské rozhraní a nabízet vzniklou aplikaci jako hotové řešení.

Platforma umožňuje poměrně snadnou tvorbu nových bloků. Má k tomu velmi detailně definované a dokumentované rozhraní. Lze tedy vytvořit bloky pro další dílčí práci s daty, které následně lze použít stejným způsobem jako bloky vestavěné.

Software Surmon mě velice zaujal, ale pro tuto práci jsem jej nakonec nevyužil. Hlavní problém je v požadavku klient-server architektury. Zkoumal jsem možnosti rozšířit software pomocí nových bloků tak, aby šel použít i v klient-

server architektuře, aby například blok začal poslouchat na síťovém portu na příchozí dotazy, nebo aby jen předzpracovával data do indexu, přičemž jiná aplikace by dodávala na základě tohoto indexu výsledky na vyhledávací dotazy. Všechny takové možnosti jsou příliš těžkopádné. Bloky jsou navrhovány jen na přijetí dat, zpracování a odeslání dat dál v řetězci. Vstupní jsou v programu všechny bloky které nemají vlastní vstupy a bloky se pak postupně zpracovávají v samostatných vláknech. Při takovém návrhu není snadné zařídit poslouchání na síťovém portu a odpovídání na požadavky.

Jinými slovy, Surmon je stavěn k ověřování konceptů různých algoritmů, nebo k tvorbě jednoúčelového text miningového softwaru jako tzv. standalone aplikace na jednom počítači. Nehodí se na finální řešení k poskytování služeb přes síťové rozhraní.

Kromě toho, aby mi byl software poskytnut pro implementaci projektu, muselo by být celé textové zpracování implementováno v Surmonu. Tvůrci Surmonu se totiž snaží bránit jakémukoliv vyvádění implementovaných knihoven ven ze softwaru. Mimo Surmon tedy nelze implementované knihovny využít.

1.4.2 Solr

Protože nástroj Surmon nelze pro můj projekt použít, začal jsem hledat open source software, který by pomohl s fulltextovým vyhledáváním, byl rozšiřitelný a zároveň by šel zakomponovat do složitější architektury. Objevil jsem Solr[3], platformu pro fulltextové vyhledávání vyvíjenou pod licencí Apache License nadací Apache Software Foundation ve spolupráci s projektem Lucene.

Solr je napsaný v jazyce Java, který mu zajišťuje multiplatformnost, takže může fungovat i pod systémem Windows, který je zadavatelem preferován.

Solr je vlastně vyhledávací server běžící v servlet containeru, jako jsou Jetty či Apache Tomcat. Poskytuje REST rozhraní pro integraci s jinými aplikacemi v různých jazycích. Solr v základní instalaci podporuje hned několik datových formátů pro REST komunikaci, například JSON a XML. Kromě REST rozhraní jsou k dispozici již hotové knihovny pro mnoho programovacích jazyků, které ještě více pomohou integrovat Solr s dalšími aplikacemi. Příkladem je Javová knihovna *SolrJ*, nebo JavaScriptové knihovny *AJAX Solr* nebo *Solr Search*.

Jako fulltextové vyhledávací jádro používá Solr knihovnu Lucene, jejíž funkce zprostředkovává pro samostatné serverové použití. Knihovna Lucene je také napsaná v jazyce Java, existují však i její portace do jiných jazyků. Knihovna slouží pro tvorbu a správu indexu a následné vyhledávání v indexu. Na svých stránkách[4] se tvůrci chlubí nízkou náročností na hardware počítače, širokou podporou různých způsobů dotazování, podporou simultálního aktualizace indexu a vyhledávání, podporou spojování výsledků z různých indexů a dalších pokročilých funkcí.

Solr sbírá (indexuje) a prohledává (dotazuje) index dokumentů. V jedné instanci Solru může být nakonfigurováno více nezávislých indexů — kolekcí.

Dokument je pro Solr jednotka informace. Sestává z atributů, polí daného datového typu, anglicky fields. Z jakých polí se dokument skládá charakterizuje tzv. schéma. To lze nastavit v konfiguračním souboru *schema.xml*. Všechny dokumenty mají v rámci kolekce stejnou strukturu polí danou schématem. Schéma určuje i to, jakým způsobem mají být pole interpretována, zpracovávána a dotazována. Tento proces tvůrci nazývají Field Analysis — analýza polí. Zde lze například nastavit, jaký tokenizer, stemmer, či filtr se má použít na dané pole. Tyto analyzéry nejsou ničím jiným než Java třídami, které implementují dané rozhraní. Poměrně jednoduše lze tedy vytvořit jakýkoliv vlastní analyzér, který předzpracuje text v poli.

Solr umožňuje použít k jednomu poli jeden tokenizer, tj. prvek, který text rozdělí na jednotlivé indexované úseky — nejčastěji slova. Dále můžeme přiřadit 0–n tzv. char filterů, které očistí text od nepotřebných znaků, jako jsou HTML značky, a 0–n filtrů, které odstraní z výstupu celá nechtěná slova. Na úryvku XML kódu můžeme pozorovat definice typu pole s analyzéry zvlášť pro proces indexace a pro dotazování a přiřazení tohoto typu pole. V ukázce si také můžeme všimnout nastavení pole *indexed* a *stored*. To znamená, že se pole má indexovat a zároveň se má v databázi uchovat i jeho původní plné znění, což je užitečné, jelikož podle indexu se rychle vyhledá daný dokument a pak se může pole zobrazit ve výsledcích v plném rozsahu.

V konfiguraci pole lze povolit i ukládání vektoru termů. Vektor lze potom využít pro další zpracování textu i mimo samotný Solr, například při shlukové analýze nebo v dalších fázích projektu.

Dalšími důležitými nastaveními v schema.xml jsou: které pole bude unikátní identifikátor dokumentu, ve kterém poli se bude ve výchozím stavu vyhledávat, jaká třída bude vyhodnocovat podobnost dokumentů. Analyzéry mívají ještě další možnosti nastavení.

```
<fieldType name="text_general" class="solr.TextField"
  positionIncrementGap="100">
 <analyzer type="index">
   <tokenizer class="solr.StandardTokenizerFactory"/>
   <filter class="solr.StopFilterFactory" ignoreCase="true"
       words="stopwords.txt" enablePositionIncrements="true" />
   <filter class="solr.SynonymFilterFactory"
           synonyms="index_synonyms.txt"
           ignoreCase="true" expand="false"/>
   <filter class="solr.LowerCaseFilterFactory"/>
 </analyzer>
 <analyzer type="query">
   <tokenizer class="solr.StandardTokenizerFactory"/>
   <filter class="solr.StopFilterFactory" ignoreCase="true"
           words="stopwords.txt"
           enablePositionIncrements="true" />
```

```
   <filter class="solr.SynonymFilterFactory"
           synonyms="synonyms.txt"
           ignoreCase="true" expand="true"/>
   <filter class="solr.LowerCaseFilterFactory"/>
 </analyzer>
</fieldType>

<field name="content" type="text_general"
       indexed="true" stored="true" termVectors="true" />
```

Zbývá se podívat na to, jak je na tom Solr s podporou českého jazyka. Podle informací na oficiálních stránkách[3] je Český jazyk nástrojem podporován již od verze 3.5. Lze prý použít standartní tokenizer *StandardTokenizer*, který dělí text na tokeny nejen podle definovaných prázdných znaků, ale má sadu pravidel, která správně nechají v celku zkratky, slova s apostrofy a podobně. K tomu jsou doporučovány dva běžné filtry: *LowerCaseFilter*, který převede token na malá písmen, a *StopFilter*, který umožní definovat stop word list, seznam slov, která se mají vynechat z indexu. Na závěr tvůrci doporučují pro český text použít *CzechStemFilter*. Je to stemmer vycházející z práce L. Dolamica a J. Savoye, kteří se na Švýcarské University of Neuchatel zabývali stemizačními algoritmy východoevropských jazyků. Stemmer funguje s tokeny již převedenými do malých písmen a umí pracovat s diakritikou. Funguje na principu postupného odebírání koncovek a některých předpon slov.

Výsledky analýzy nástroje Solr lze tedy shrnout takto: Solr se zdá být vhodným kandidátem pro nasazení v projektu. Neklade meze integraci s jinými aplikacemi a do jisté míry podporuje český jazyk. Nepochybně ale bude nutné otestovat kvalitu zpracování českých textů a popřípadě nástroj rozšířit o nové knihovny pro tento účel.

1.4.3 Jetty

Solr jako Java EE aplikace potřebuje servlet container ke svému běhu — Jetty. Jetty[5] je open source javový web server a java servlet container, vyvíjený společností Eclipse Foundation. Pro projekt jsem zvažoval i použití Apache Tomcat[6], se kterým už mám zkušenost z předchozích projektů, ale narazil jsem na několik chyb a nekompatibilit se Solrem. Zůstal jsem tedy nakonec u Jetty, se kterým se Solr oficiálně distribuuje a který oficiálně podporuje.

1.4.4 Carrot2

Jedním z požadavků na projekt je shlukování výsledků vyhledávání. Zjistil jsem, že existuje javový open source shlukovací engine Carrot2[7], který má podporu v jádře Solru. Engine je určený pro shlukování spíše menších kolekcí dat a hodí se tedy právě pro shlukování výsledků vyhledávání.

Carrot2 se distribuuje se třemi implementovanými algoritmy pro shlukování. Obsahuje algoritmus Bisekční K-means, který počítá vzdálenosti vektorů dokumentů nebo vektoru reprezentantů shluků a vždy do shluku spojí dva nejbližší sousedy až do požadovaného počtu shluků. Dále obsahuje algoritmus Lingo, který vytvořili S. Osiński, J. Stefanowski a D. Weiss, založený na algebraických transformacích matic termů a extrakci častých frází použitím sufixových polí. Posledním obsaženým algoritmem je STC, suffix tree clustering, který sestaví termy do sufixového stromu, aby nalezl prvotní zástupce shluků dokumentů.

Carrot2 obsahuje i J2EE webovou aplikaci s hotovým vyhledávacím rozhraním. Tato aplikace má možnost nakonfigurovat i několik informačních zdrojů, ve kterých se bude vyhledávat. V základu je několik zdrojů implementováno. Můžeme aplikaci napojit například na Wikipedii, Google, Bing i Solr. Aplikace již podporuje zvýraznění hledaných slov ve výsledcích, zobrazení shluků i filtrování dokumentů podle zvoleného shluku. Funguje tím způsobem, že dotaz přepošle na zdrojovou službu a nad výsledky, které jí služba vrátí provede shlukovou analýzu. Následně pak zobrazí výsledky vyhledávání i nalezené shluky. Shluky navíc umí zobrazit i interaktivně v grafické formě.

Carrot2 lze tedy k Solru připojit dvěma způsoby. Jednou možností je nasadit Carrot2 jako samostatnou webovou aplikaci, která bude zprostředkovávat vyhledávací dotazy na Solr server a výsledky vyhledávání před vrácením ještě podrobí shlukové analýze. V tomto případě by ale musel Carrot2 být zároveň i frontendem systému a tak využiji druhé možnosti připojení: Carrot2 bude do Solru nainstalován a nakonfigurován jako rozšiřující knihovna pro shlukování výsledků, která bude obohacovat výsledky vyhledávání o výsledky shlukové analýzy. Tím se zachová možnost výběru frontendu.

Kapitola 2

Návrh

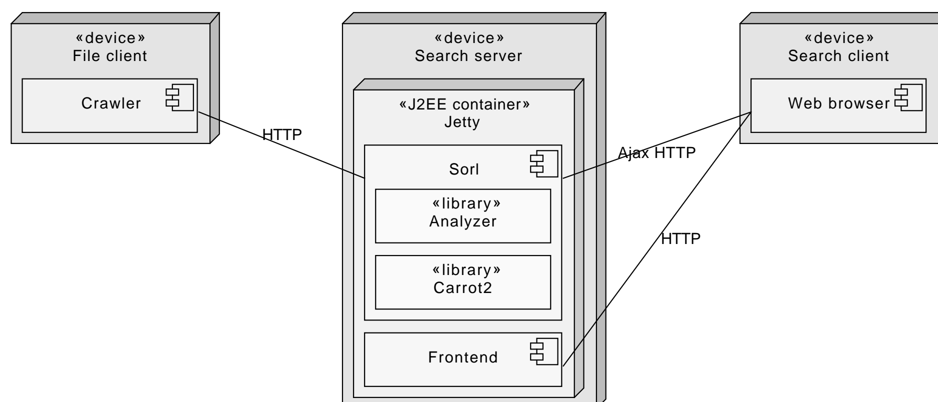
2.1 Architektura

V této kapitole se pokusím navrhnout strukturu jednotlivých komponent výsledného řešení. Celá architektura systému je znázorněna na obrázku 2.1. Středobodem systému je server s nainstalovaným servlet containerem Jetty. V containeru bude nainstalován Solr.

Solr bude tvořit backend aplikace. Bude do něj připojena knihovna *Carrot2*, která rozšíří jeho možnosti o shlukování výsledků, a má vlastní knihovna s implementovanými nástroji pro zpracování českého textu pojmenovaná *Analyzery*. Solr bude tedy uchovávat index i samotný obsah dokumentů, bude zpracovávat vyhledávací dotazy uživatelů a vracet nalezené výsledky včetně informací o shlucích a o podobnosti dokumentů.

Kromě Solru poběží v Jetty ještě frontend aplikace. Frontend vytvořím vlastní tak, aby vyhovoval požadavkům. Solr tvoří solidní backend, který komunikuje pomocí HTTP rozhraní a umožňuje vracet data ve snadno zpracovatelném JSON formátu. Rozhodl jsem se proto, že frontend bude tvořen několika statickými stránkami a veškerá dynamika bude dodána pomocí JavaScriptu. Stránka se bude pomocí AJAX požadavků dotazovat přímo backendu aplikace — Solru. Servlety tedy budou jen posílat statické HTML stránky, JavaScriptové knihovny, CSS styly a podobné soubory. Výhoda řešení spočívá především v jeho jednoduchosti. Celý vyhledávací server bude fungovat v jednom servlet containeru, vše se tedy zprovozní jen spuštěním nakonfigurovaného Jetty. Zároveň nebudu muset vytvářet stejné rozhraní dvakrát (jednou pro dotazování JavaScriptu webového serveru a jednou pro dotazování webového serveru Solru).

Poslední komponentou v architektuře je tzv. Crawler. To bude samostatná aplikace, jejímž úkolem bude indexovat soubory nebo naopak mazat chybějící soubory z indexu. Připojovat se bude přímo na rozhraní Solru pomocí HTTP. Komunikace se Solrem bude síťová, díky čemuž může aplikace fungovat nejen na stejném stroji s vyhledávacím serverem, ale i na jakémkoliv počítači v síti.



Obrázek 2.1: Diagram nasazení

2.2 Crawler

Jedná se o aplikaci, která dle nastavení projde rekurzivně složky file systému, vyhledá všechny soubory, sestaví z nich Solr dokumenty a pošle je k indexaci do Solru. Dále bude procházet dokumenty v Solru, kontrolovat, zda daný dokument ještě existuje ve filesystému, a pokud ne, tak jej z indexu odstraní. Jejím posledním úkolem je kompletní vyprázdnění indexu Solru.

Vzhledem k tomu, že aplikace bude komunikovat se Solr serverem, rozhodl jsem se, že tuto aplikaci naprogramuji v jazyce Java, jelikož pro Javu existuje oficiální Java knihovna *Apache SolrJ* pro napojení na Solr server. Kromě této knihovny využiji ještě knihovnu *Apache Log4J* pro zaznamenávání nastalých událostí, která je již závislostí *SolrJ*, a protože bude součástí aplikace, nebránil jsem se jejímu využití. Konfiguraci programu si usnadním využitím knihovny *Apache commons-configuration*, která mi umožní definovat v *resources* adresáři výchozí konfigurační hodnoty, které ovšem bude možné přetížit hodnotami jinými pomocí externího konfiguračního souboru nebo pomocí argumentů programu. Díky tomu bude možné měnit chování programu podle aktuálních potřeb.

Aplikaci mám v plánu navrhnout tak, aby byla rozšiřitelná o další specializace výše sepsaných úkolů. Zadavetel totiž již během tvorby aplikace předložil několik nových návrhů k jejímu rozšíření — například aby aplikace podporovala načítání nejen textových souborů. Existuje také možnost (jak jsem psal v kapitole 1), že v budoucnu se bude aplikace napojovat přímo na databázi policejních dokumentů.

2.2.1 Návrh tříd

Během analýzy požadavků jsem nalezl několik společných rysů v jednotlivých úkolech. Úkol vždy začíná procházením určitého média (např. indexu dokumentů či file systemu). Jednotlivé nalezené prvky (např. Solr dokumenty či soubory) se poté validují (kontroluje se existence na filesystému). V dalším kroku se prvky parsují či konvertují do jiného datového formátu (soubor na Solr dokument), prvky se pak sbírají do kolekce a jako kolekce se pak finálně zpracují (pošlou se na server k zaindexování nebo se smažou z indexu). Tyto shodné rysy jsem tedy postupně ještě více generalizoval, až jsem nakonec došel k návrhu, který popisuji níže.

2.2.1.1 Procesory

Jednotlivé úkoly budou prováděny pomocí řetězce procesorů, tříd implementujících rozhraní *IProcessor*. Procesorové třídy jsou znázorněny v diagramu 2.2. Řetězec vždy bude začínat *Crawlerem*, tj. třídou, která prochází médium a předává jednotlivé nalezené prvky prvnímu procesoru v řetězci. Většina procesorů bude dědit od *AbstractPassThroughProcessor*. Tyto procesory vždy vykonají určitou dílčí činnost a produkt své činnosti pošlou dalšímu procesoru v řetězci. Každý procesor bude mít jedinou funkci.

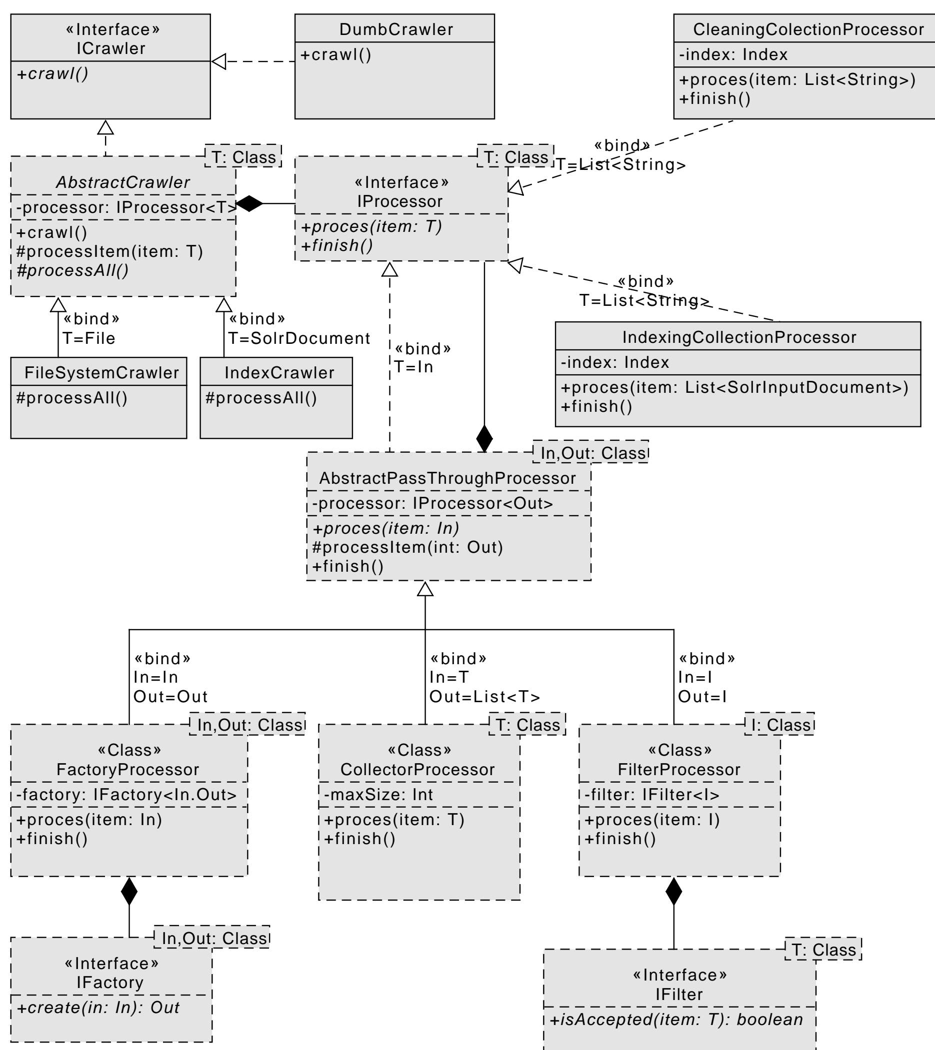
FactoryProcessor bude konvertovat data z jednoho formátu do druhého. Bude tak činit pomocí třídy implementující *IFactory* rozhraní dle návrhového vzoru strategy. Implementovat budu dva typy těchto factory strategií. Jeden typ bude získávat *SolrInputDokument* z objektu *File* (ten bude použit při indexování dokumentů). Vytvořím dvě implementace: jedna bude umět pouze načítat textové soubory, druhá bude určena pro načítání jakýchkoli textových dokumentů na disku za použití knihovny *Tika*. Druhý typ factory strategie bude z objektu *SolrDocument* získávat id dokumentu pro kontrolu a případné smazání z indexu.

FilterProcessor dokáže ze vstupních dat odfiltrovat některé prvky tím, že je nepředá na výstup. Strategie filtrování bude opět záviset na vložené třídě implementující rozhraní, v tomto případě rozhraní *IFilter*. Návrh *Factory* a *Filter* tříd je znázorněn na diagramu 2.3. Filtry budou použity jen při čištění indexu, všechny tedy souvisí s tímto úkolem. Do budoucna však nepředstavuje problém rozšířit aplikaci o filtry souborů.

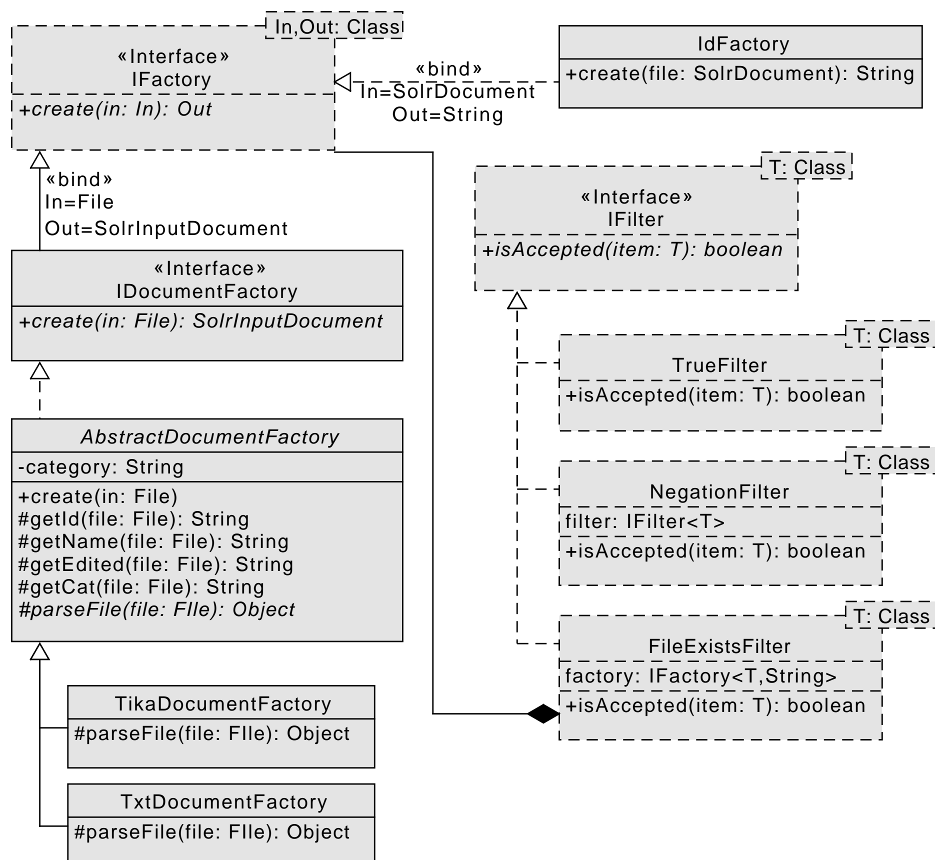
Budu implementovat *TrueFilter*, který přijme všechny dokumenty, *NegationFilter*, který bude negovat informaci o přijetí souboru dalšího vloženého filtru, a *FileExistsFilter*, který bude kontrolovat existenci souboru. Poslední zmiňovaný filtr bude využívat dříve definované *Factory* pro získání cesty k souboru ze Sorl dokumentu.

Nakonec bude nutné implementovat *CollectorProcessor*, který sesbírá prvky do kolekce a nechá je naráz zpracovat dalším procesorem.

Řetězce procesorů budou končit procesorem *IndexingCollectionProcessor* nebo *CleaningCollectionProcessor*. Tyto procesory přijmou kolekci dokumentů k odstranění či zaindexování a provedou kýžené změny v indexu. Komunikace těchto procesorů se Solr serverem bude odstíněna vyčleněním funkcí do samostatné třídy *Index*. *Index* bude pro procesory jednotným modelem pro komunikaci se Solrem a jeho rozhraní tedy budou procesory používat k aplikování změn v indexu.



Obrázek 2.2: Diagram procesorových tříd Crawleru



Obrázek 2.3: Diagram factory a filter tříd Crawleru

2.2.1.2 Tovární třídy

Řetězce procesorů budou sestavovány na základě konfigurace, s využitím návrhového vzoru factory. Sestavování bude mít vždy na starosti daná tovární třída. Tovární třídy se budou řídit parametry přijatými z konfigurace.

V budoucnosti by bylo možné tyto tovární třídy upravit, aby spolupracovaly s načítáním tříd a konfigurací. V konfiguraci by byl definován řetězec tříd s případnými konfiguračními parametry. Tovární třída by našla definované třídy (například i v externích jar knihovnách v adresáři *lib*), načetla je a sestavila do řetězce. Tím by byla zaručena absolutní rozšiřitelnost řešení. Podobným způsobem funguje i konfigurace Solru. Tato verze Crawleru však toto rozšíření zatím obsahovat nebude.

2.3 Uživatelské rozhraní

Na návrhu UI jsem úzce spolupracoval se zadavatelem, abych co nejvíce vyhověl jeho požadavkům. Při návrhu rozhraní jsem se snažil držet se zvyklostí zažitých ve webovém prostředí. Zvláště pak vycházím ze zvyklostí z internetových vyhledávačů, jako je Google. Další inspirací byl webový frontend aplikace Carrot2. Jelikož jde o webovou aplikaci, počítá návrh s proměnnou velikostí okna a celý layout bude plně responzivní a přizpůsobitelný pro jakoukoliv velikost okna, včetně mobilní.

Všechny funkční požadavky jde rozdělit do dvou částí: na požadavky týkající se výsledků vyhledávání jako celku (vyhledávání, shlukování) a požadavky týkající se jednoho vybraného dokumentu (podobnostní vyhledávání, zobrazení obsahu). Uživatelské rozhraní tedy bude sestávat také ze dvou layoutů, z nichž každý bude odpovídat jedné skupině požadavků. Mezi layouty se pak bude volně procházet proklikem.

Layouty budou mít společný jednotící prvek, a to hlavičku stránky s logem aplikace, které bude umístěné tradičně v levém horním rohu stránky. Zbytek hlavičky bude odpovídat kontextu daného layoutu, aby se neplýtvalo místem. Hlavička bude i nositelem informace o kontextu aktuální stránky a bude napovídat, v jaké části se uživatel nachází.

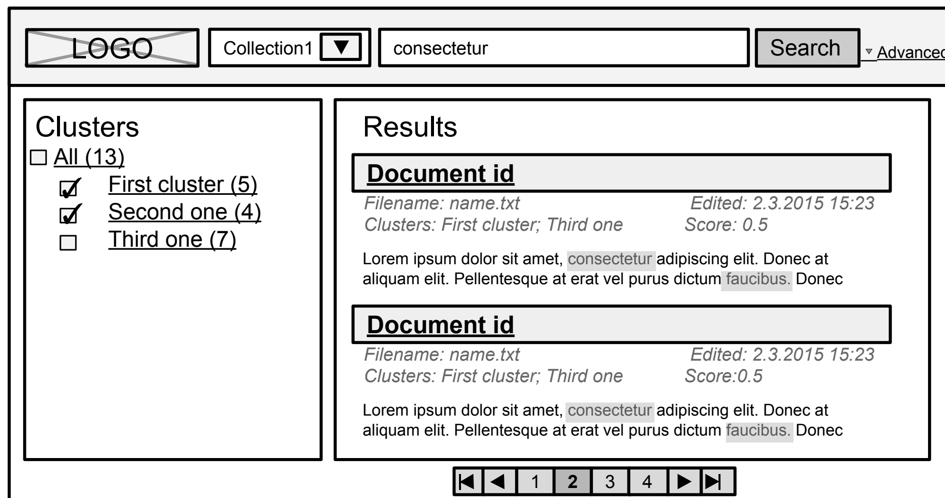
2.3.1 Vyhledávání dokumentů

Výchozí stránka bude mít v hlavičce vyhledávací formulář. Layout je znázorněn na obrázku 2.4. Formulář bude obsahovat vstupní pole pro výběr zdroje v podobě roletky, kde uživatel zvolí, který index dokumentů se má pro vyhledávání použít. V roletce by měl být automaticky vybrán výchozí index nastavený v Solru. Vpravo od roletky umístím samotné vyhledávací pole a potvrzovací tlačítko pro vyvolání akce vyhledávání. Vyhledávací pole by mělo být rozměrné a mělo by vyplnit celou zbývající šířku okna.

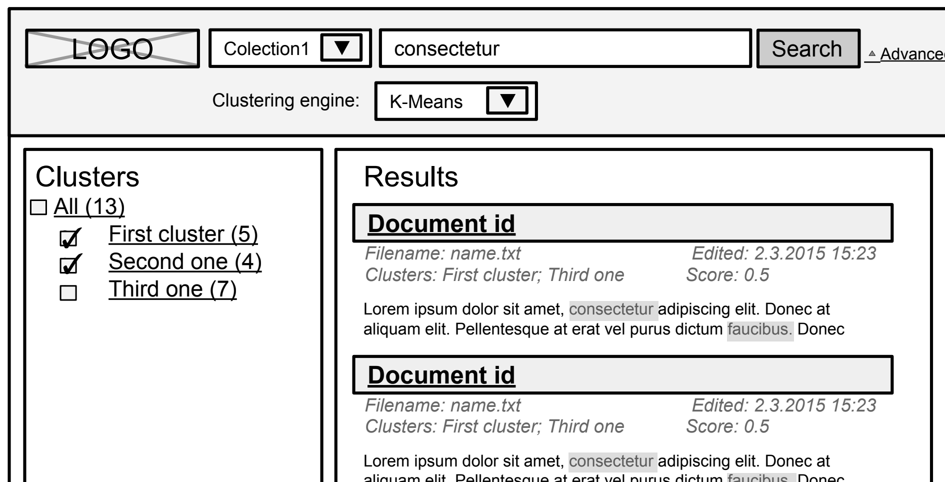
Vpravo od potvrzovacího tlačítka bude dále umístěn malý odkaz se šipkou, který bude přepínat zobrazení rozšířené nabídky nastavení vyhledávání. Podobu rozšířené nabídky můžeme pozorovat na obrázku 2.5. Nabídka bude stále ještě součástí hlavičky stránky (hlavička se o ní vlastně rozšíří). Měla by obsahovat jen roletku pro přepínání shlukovacího algoritmu, v budoucnu bude možné přidat další volby.

Pod hlavičkou se zobrazí samotný vyhledaný obsah. Všechen vyhledaný obsah projde rovnou shlukovací analýzou. Shlukování nebude nutné zapínat. Vlevo bude umístěn sloupec o šířce rovné čtvrtině šířky okna s nalezenými shluky. Vpravo se pak ve zbývající šířce zobrazí nalezené výsledky. Pokud bude okno příliš úzké, zobrazí se výsledky až pod seznamem shluků a shluky i výsledky se roztáhnou na plnou šíři okna.

Sloupec se shluky bude mít formu odrážkového seznamu. Nejvyšší prvek seznamu je shluk „All“, který bude obsahovat všechny nalezené výsledky. Pod



Obrázek 2.4: Návrh vyhledávacího uživatelského rozhraní



Obrázek 2.5: Návrh vyhledávacího uživatelského rozhraní s rozšířenou nabídkou

ním teprve budou jednotlivé nalezené shluky. U každého shluku se zobrazí číslo odpovídající počtu obsažených dokumentů. Seznam shluků bude provádět i filtrování dokumentů. Kliknutím na shluk se shluk označí (v obrázku 2.4 je označen zatržítkem) a ve výsledcích se zobrazí jen dokumenty patřící do vybraného shluku. Vybrat takto půjde libovolný počet shluků. Při každém novém hledání se vždy jako výchozí označí shluk „All“, aby se zobrazily

ve výsledcích všechny nalezené dokumenty a teprve sám uživatel si provedl případnou filtraci.

Sloupec s výsledky bude obsahovat nalezené dokumenty ve vybraných shlucích. Každý dokument bude reprezentován svým ID v podobě odkazu vedoucího na zobrazení obsahu dokumentu. Pod ID budou zobrazena různá metadata, jako jméno souboru, kdy byl dokument naposledy upraven, skóre (tj. číslo vyjadřující míru shody hledaného výsledku s hledaným výrazem) či seznam clusterů, do kterých dokument patří. Pod metadaty bude umístěn úryvek obsahu dokumentu se zvýrazněnými hledanými výrazy. Výsledky budou řazeny podle skóre.

Na konci stránky pod výsledky bude umístěno ovládání stránkování: tlačítka pro posun na předchozí či další stránku, na první či poslední stránku a seznam několika stránek v okolí aktuálně zobrazené stránky, která bude na tomto panelu zvýrazněna.

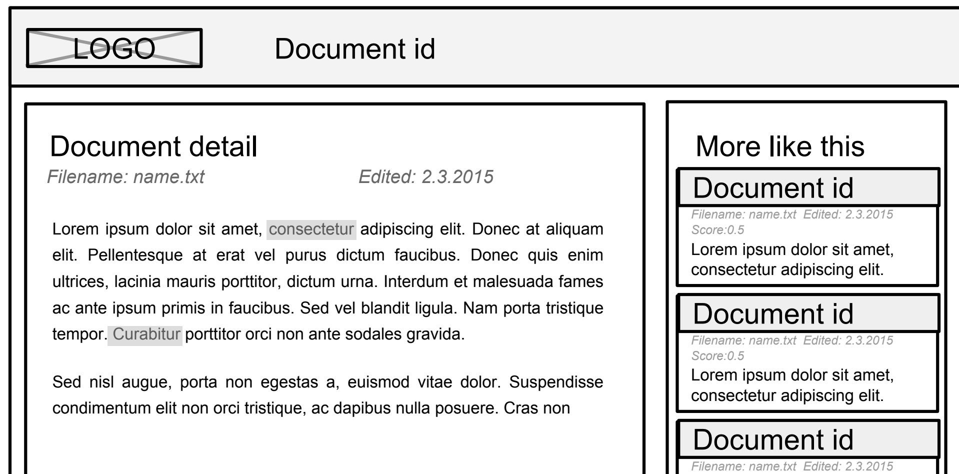
Pořadí sloupců se shluky a s výsledky jsem volil s ohledem na posloupnost pracovního postupu a na zvyklosti. Lze předpokládat, že uživatel po zadání dotazu nejdříve prohlédne shluky a vybere, z kterých shluků chce dokumenty vidět. Seznam shluků má zároveň funkčně nejblíže k menu, a to je zvykem na internetových stránkách umisťovat po levé straně od obsahu.

2.3.2 Detail dokumentu

Kliknutím na ID dokumentu ve výsledcích vyhledávání se dostaneme do zobrazení detailu dokumentu. Návrh stránky je znázorněn na obrázku 2.6. Layout této stránky bude mít v hlavičce kromě loga už jen vypsané ID otevřeného dokumentu a bude opět dvousloupcový. Vlevo bude umístěn obsah samotného dokumentu, vpravo sloupec o šířce jedné třetiny šířky okna obsahující seznam podobných dokumentů k aktuálně zobrazenému. Bude-li okno příliš malé, sloupce se zobrazí pod sebou na plné šíři okna, stejně jako u vyhledávacího layoutu. Pořadí sloupců je opět dáno zvyklostmi na internetových stránkách. Sloupec s podobnými dokumenty se obvykle spolu s dalšími kontextovými informacemi umisťuje do pravého sloupce vedle hlavního obsahu, případně pod obsah.

Sloupec s obsahem bude ve vrchní části zobrazovat metadata dokumentu a následovat bude celý text dokumentu. Vyhledávané fráze budou v textu zvýrazněny.

Pravý sloupec bude obsahovat seznam dokumentů, které jsou podobné dokumentu právě otevřenému. Seznam bude strukturován podobně jako seznam výsledků ve vyhledávacím layoutu: bude obsahovat ID dokumentu, které bude zároveň odkazem na detail příslušného dokumentu, poté budou následovat metadata dokumentu a úryvek obsahu. Dokumenty budou v tomto sloupci řazeny podle skóre, které vyjadřuje míru podobnosti dokumentu s otevřeným dokumentem.



Obrázek 2.6: Návrh uživatelského rozhraní pro zobrazení dokumentu

KAPITOLA 3

Implementace

3.1 Solr

Základní komponentou, kterou bylo třeba implementovat, byl Solr. Potřebuji aby software zvládl indexovat text v českém jazyce. Solr obsahuje předinstalovanou a částečně předkonfigurovanou českou sadu analyzérů, která zahrnuje *StandardTokenizer*, *LowerCaseFilter*, *StopFilter* a *CzechStemFilter*. V tomto pořadí.

StandardTokenizer dělí text na tokeny. Po zběžné zkoušce jsem usoudil, že jeho algoritmus je funkční i pro český jazyk. *LowerCaseFilter* je určen k převodu textu na malá písmena. I tento filtr jsem otestoval a opět se neprojevily žádné problémy — převádí na malá písmena i speciální české znaky s interpunkcí.

StopFilter odebírá tokeny odpovídající slovům v seznamu. Jeho fungování tedy závisí na kvalitě seznamu. V Solru je předinstalován seznam *czech_stopwords.txt*, jenž obsahuje slova, která jsou v českém jazyce natolik frekventovaná, že nevypoví o vyhledávaném textu žádnou zásadní informaci. Jedná se především o nejrůznější předložky, spojky a pomocná slovesa. Tento seznam jsem shledal dostačujícím, už proto, že kdykoliv v případě potřeby půjde seznam rozšířit o další slova, která se vyskytují příliš často v policejním prostředí, a nepomáhají tudíž specifikovat hledaný text.

Poslední *CzechStemFilter* je klíčový analyzér. Jak jsem již psal v kapitole 1 Analýza, měl by být výsledkem práce švýcarských programátorů. O jeho kvalitách nejsem zcela přesvědčen, implementoval jsem proto do Solru další české stemmery.

3.1.1 Stemmery

K tomu, aby mohl být implementovaný *StemFilter* použit v Solru, je nutno vytvořit JAR knihovnu obsahující třídu implementující rozhraní *TokenFilter* a zároveň její tovární třídu, nejlépe potomka *TokenFilterFactory*. Sestavená

JAR knihovna se pak umístí do adresáře *Lib*, kde z ní bude moci *ClassLoader* Solru číst. V konfiguraci Solru je pak již možné použít nový filtr definovaný absolutním jménem tovární třídy. Parametry definované v konfiguraci se předávají tovární třídě, je tedy možné konfigurací ovlivnit funkci stemmeru.

3.1.1.1 Light a Agressive stemmer

Vnitřní *CzechStemFilter* Solru je dle dokumentace[8] jedním ze stemizačních algoritmů autorů L. Dolamica a J. Savoye. Výsledkem jejich práce[9] je stemizační algoritmus, který je založen na technice postupného hledání a odtrhávání koncovek slov. Vytvořili dvě verze svého algoritmu, takzvanou light a agressive verzi, které se liší počtem odebíraných koncovek slov (agresivní verze jich odebírá více). Ani jedna z těchto verzí ale prostým porovnáním kódu neodpovídá českému filtru implementovanému v Solru. Rozhodl jsem se proto, že doimplementuji light i agressive stemmer.

Zdrojový kód obou stemmerů je k dispozici na internetu v jazyce Java pod BSD licencí. Jde o dvě třídy s implementací. Vytvořil jsem tedy vlastní třídu *CzechStemFilter*, která bude tvořit adaptér implementačním třídám. Dále jsem vytvořil tovární třídu *CzechStemFilterFactory*, která inicializuje *CzechStemFilter* a dle parametrů v konfiguraci zvolí „agresivní“ či „odlehčenou“ implementaci stemmeru. Při tvorbě těchto dvou tříd jsem se inspiroval původní implementací *CzechStemFilter* v aplikaci Solr.

Výslednou implementaci jsem zkompiloval do knihovny *Analyzery.jar* a podle výše popsaného postupu integroval do Solru.

3.1.1.2 Helebrand stemmer

V rámci dalšího výzkumu jsem nalezl diplomovou práci D. Helebranda z VUT v Brně[10], který implementoval stemmer pro český jazyk v prostředí Snowball. Snowball je programovací jazyk speciálně navržený pro zpracovávání textových řetězců, především právě pro tvorbu stemizačních algoritmů. Výsledný algoritmus lze pomocí nástrojů Snowballu přímo přeložit do jazyka C či Java. Hotový stemizační algoritmus pana Helebranda je k dispozici pod licencí GNU General Public License.

Získal jsem tedy algoritmus z webových stránek brněnské technické univerzity a pomocí výše zmíněných nástrojů jsem jej nechal přeložit do Javy. Solr v sobě již má implementovány stemmery pro několik jazyků pomocí prostředí Snowball, nebylo proto příliš složité výsledný Java algoritmus integrovat. Obsažená tovární třída *SnowballPorterFilterFactory* je příhodně navržená tak, že jméno portované třídy s algoritmem hledá v balíčku *org.tartarus.snowball.ext*. Stačilo tedy jen vygenerovanou třídu s algoritmem vhodně pojmenovat a umístit do správného balíčku. *SnowballPorterFilterFactory* přijímá jméno třídy, kterou má hledat, jako parametr z konfigurace.

Výslednou sestavu jsem přibalil do JAR knihovny s předchozími implementacemi stemmerů, bude tedy umístěna v adresáři *Lib*, kde ji *ClassLoader* Solru nalezne.

3.1.1.3 Hunspell stemmer

Po implementaci tří stemmerů založených na postupném odtrhávání koncovek jsem začal hledat řešení, které by používalo jinou metodu získávání kořenů. Hledal jsem algoritmus založený buď na strojovém učení, nebo na porovnávání se slovníkem a zjistil jsem, že existuje algoritmus, který využívá Hunspell k odebrání koncovek s využitím slovníku, a že je již implementovaný v Solru.

Hunspell je open source knihovna sloužící ke kontrole pravopisu. Je využívaný například v projektech Apache OpenOffice či Google Chrome. Slovníky, včetně českého, lze pro Hunspell volně stáhnout přímo z internetových stránek OpenOffice[11]. České slovníky jsou licencovány pod GNU/GPL licencí. Byly tvořeny pro starší knihovnu Ispell, Hunspell je s ní však zpětně kompatibilní.

Ze sady souborů českého slovníku v projektu využiji jen dva: soubor s koncovkou *.dic*, který obsahuje seznam slov daného jazyka, ideálně v kořenovém tvaru, a soubor *.aff*, který obsahuje pravila přidávání koncovek[12]. Soubor *cs_CZ.aff* českého slovníku obsahuje syntaktickou chybu, kterou jsem musel před použitím slovníku opravit, protože jinak se Solr při jeho „parsování“ ukončí kritickou chybou.

Nevýhodu této metody vidím především v nedostatečné kvalitě slovníku. Poslední verze pochází z roku 2008 a nezdá se, že by byl nadále aktivně rozšiřován.

3.1.2 Konfigurace

3.1.2.1 Datová pole

K již existujícímu textovému datovému typu *text_cz*, který používá vestavěný český stemmer, jsem v konfiguračním souboru *schema.xml* vytvořil datové typy, jenž využívají nově vytvořené stemmery. Toho jsem docílil následující konfigurací.

```
<!-- Czech original -->
<fieldType name="text_cz"
           class="solr.TextField"
           positionIncrementGap="100">
  <analyzer>
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.StopFilterFactory"
            ignoreCase="true"
            words="lang/stopwords_cz.txt" />
```

```
    <filter class="solr.CzechStemFilterFactory"/>
  </analyzer>
</fieldType>

<!-- Czech Helebrand -->
<fieldType name="text_cz_helebrand"
           class="solr.TextField"
           positionIncrementGap="100">
  <analyzer>
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.StopFilterFactory"
            ignoreCase="true"
            words="lang/stopwords_cz.txt" />
    <filter class="solr.SnowballPorterFilterFactory"
            language="CzechHelebrand" />
  </analyzer>
</fieldType>

<!-- Czech Dolamic light-->
<fieldType name="text_cz_light"
           class="solr.TextField"
           positionIncrementGap="100">
  <analyzer>
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.StopFilterFactory"
            ignoreCase="true"
            words="lang/stopwords_cz.txt" />
    <filter class=
"cz.cvut.skorpste.dip.stemmer.dolamicsavoy.CzechStemFilterFactory"
            implementation="Light" />
  </analyzer>
</fieldType>

<!-- Czech Dolamic agressive-->
<fieldType name="text_cz_agressive"
           class="solr.TextField"
           positionIncrementGap="100">
  <analyzer>
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.StopFilterFactory"
            ignoreCase="true"
```

```
            words="lang/stopwords_cz.txt" />
    <filter class=
"cz.cvut.skorpste.dip.stemmer.dolamicsavoy.CzechStemFilterFactory"
           implementation="Agressive" />
  </analyzer>
</fieldType>

<!-- Czech Hunspell -->
<fieldType name="text_cz_hunspell"
           class="solr.TextField"
           positionIncrementGap="100">
  <analyzer>
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.StopFilterFactory"
            ignoreCase="true"
            words="lang/stopwords_cz.txt" />
    <filter class="solr.HunspellStemFilterFactory"
            dictionary="cs_CZ.dic" affix="cs_CZ.aff" />
  </analyzer>
</fieldType>
```

Datové typy jsou pojmenované *text_cz_light*, *text_cz_agressive*, *text_cz_helebrand* a *text_cz_hunspell*. Všechny tyto typy mají shodný analyzér pro dotazování i pro indexování. V ukázce konfigurace můžeme pozorovat parametry stemmerů, o kterých jsem se zmiňoval v kapitolách o implementaci jednotlivých stemmerů.

Poté jsem definoval datová pole indexu. Pole se definují také v konfiguračním souboru *schema.xml*. Pole ID je povinné, protože (podobně jako v databázích) jednoznačně identifikuje dokument. Datový typ tohoto pole jsem nastavil na *string* a budu do něj ukládat absolutní cestu k souboru ve filesystému zdrojového počítače. Pole *name* bude uchovávat jméno souboru, ze kterého dokument pochází, pole *edited* pak čas poslední editce souboru. Pole *cat* je spíše přípravou pro budoucí možnost rozšíření o nastavení kategorie, do které soubor spadá. Aktuálně nemá žádnou funkci.

Posledním definovaným polem je *text*, do kterého se bude ukládat samotný text dokumentu. Jeho datový typ je aktuálně nastavený na *text_cz* a lze jej nastavit na kterýkoliv z implementovaných českých datových typů definovaných výše. O tom, který datový typ bude zvolen ve výchozím nastavení, rozhodnu na základě výsledků testu stemmerů v kapitole 4 Testování.

```
<field name="id" type="string" indexed="true"
       stored="true" required="true"
       multiValued="false" />
```

```
<field name="name" type="text_general" indexed="true"
       stored="true"/>
<field name="edited" type="date" indexed="true"
       stored="true" />
<field name="cat" type="string" indexed="true"
       stored="true" required="true" multiValued="false"/>
<field name="text" type="text_cz" indexed="true"
       stored="true" termVectors="true" />
```

3.1.2.2 Request handlery

Konfigurace tzv. request handlerů se nalézá v konfiguračním souboru *solrconfig.xml.* Mapuje URL adresu na komponenty, které dle nastavení zpracují požadavek. Definoval jsem dva request handlery, z nichž každý odpovídá jedné stránce uživatelského rozhraní, protože pro každou stránku bude třeba jiné zpracování.

Request handler */searcher* je určen pro stránku vyhledávání a bude tedy dodávat výsledky vyhledávání obohacené o shlukování a zvýraznění hledaných slov. Dále se v konfiguraci nastavuje výchozí velikost stránky, výchozí dotaz a pole, která má server vrátit v odpovědi. Druhý request handler, */detail*, je určen pro stránku s detailem dokumentu a bude tedy vracet obsah dokumentu se zvýrazněním hledaných slov a podobné dokumenty.

```
<requestHandler name="/searcher"
                startup="lazy"
                enable="${solr.clustering.enabled:false}"
                class="solr.SearchHandler">
  <lst name="defaults">

    <!-- Clustering -->
    <bool name="clustering">true</bool>
    <bool name="clustering.results">true</bool>
    <bool name="clustering.collection">true</bool>
    <str  name="clustering.engine">lingo</str>
    <str name="carrot.title">text</str>
    <str name="carrot.url">id</str>
    <bool name="carrot.produceSummary">true</bool>
    <bool name="carrot.outputSubClusters">true</bool>

    <!-- Search -->
    <str name="defType">edismax</str>
    <str name="qf">text^10.0</str>
    <str name="df">text</str>
    <str name="mm">100%</str>
```

```
    <str name="q.alt">*:*</str>
    <str name="rows">50</str>
    <str name="fl">*,score</str>

    <!-- More like this -->
    <str name="mlt.qf">text^10.0</str>
    <str name="mlt.fl">text</str>
    <int name="mlt.count">0</int>

    <!-- Highlighting -->
    <str name="hl">on</str>
    <str name="hl.fl">text</str>
    <str name="hl.preserveMulti">true</str>
    <str name="hl.encoder">html</str>
    <str name="hl.simple.pre">&lt;mark&gt;</str>
    <str name="hl.simple.post">&lt;/mark&gt;</str>
    <str name="hl.simple.ellipsis">...</str>
    <str name="f.text.hl.snippets">3</str>
    <str name="f.text.hl.fragsize">200</str>
    <str name="f.text.hl.alternateField">text</str>
    <str name="f.text.hl.maxAlternateFieldLength">750</str>
  </lst>
  <arr name="last-components">
    <str>clustering</str>
  </arr>
</requestHandler>

<requestHandler name="/detail"
                startup="lazy"
                class="solr.SearchHandler">
  <lst name="defaults">

    <!-- Search -->
    <str name="defType">edismax</str>
    <str name="qf">text^10.0</str>
    <str name="df">text</str>
    <str name="mm">100%</str>
    <str name="q.alt">*:*</str>
    <str name="rows">1</str>
    <str name="fl">*,score</str>


    <!-- More like this -->
    <str name="mlt">true</str>
```

```
    <str name="mlt.qf">text^10.0</str>
    <str name="mlt.fl">text</str>
    <int name="mlt.count">10</int>

    <!-- Highlighting -->
    <str name="hl">on</str>
    <str name="hl.fl">text</str>
    <str name="hl.preserveMulti">true</str>
    <str name="hl.encoder">html</str>
    <str name="hl.simple.pre">&lt;mark&gt;</str>
    <str name="hl.simple.post">&lt;/mark&gt;</str>
    <str name="hl.simple.ellipsis">...</str>
    <str name="f.text.hl.snippets">1</str>
    <str name="f.text.hl.fragsize">0</str>
    <str name="f.text.hl.alternateField">text</str>
    <str name="f.text.hl.maxAlternateFieldLength">0</str>
  </lst>
</requestHandler>
```

3.2 Crawler

Crawler jsem implementoval přesně podle návrhu popsaného v kapitole 2.2. Třída *DocumentFactory* tvoří výsledné dokumenty odpovídající schématu Solru a aplikace počítá při ověřování existence souboru s tím, že v poli *id* je uložena absolutní cesta k souboru v místním filesystému.

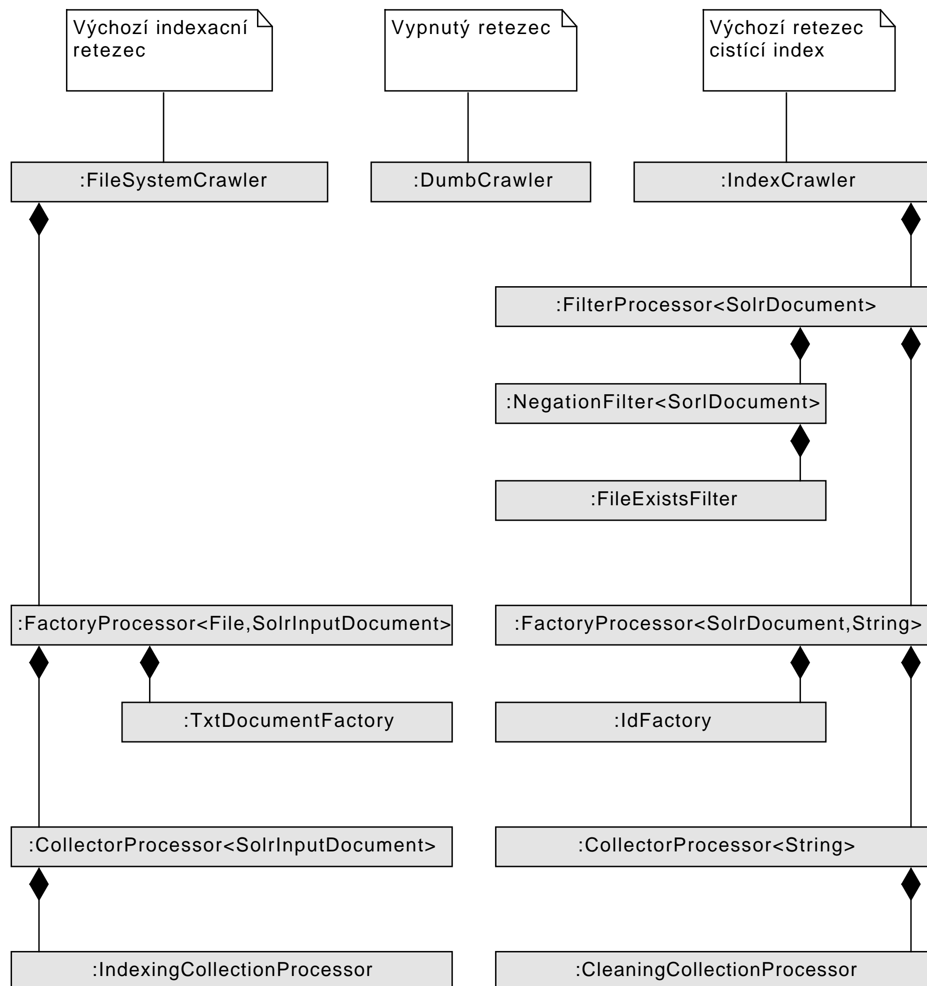
3.2.1 Konfigurace

Veškerá konfigurace se realizuje pomocí Apache commons-config knihovny. Knihovna umožňuje načíst a zparsovat hned několik zdrojů nastavení a pak v místě potřeby jen získat hodnotu pomocí definovaného klíče. Za sestavení konfigurace je zodpovědná třída *ConfigFactory*. Ta vrací singleton objekt *Config* s nastavením.

Crawler nastavení načítá z parametrů Java VM, z konfiguračního souboru a z vnitřního konfiguračního souboru. Tyto soubory jsou ve formátu Java properties. Hodnoty nastavení lze přetěžovat s tím, že nejvyšší prioritu mají parametry VM, poté externí konfigurační soubor a nejnižší prioritu má vnitřní konfigurační soubor. Cestu k souboru s nastavením lze dodat programu pomocí VM parametru. Bez tohoto parametru se aplikace pokusí načíst konfiguraci ze souboru *crawler.properties*. Pokud soubor nenalezne, nastavení ze souboru nenačte vůbec a chování se bude řídit pouze pomocí parametrů předaných Java VM a vnitřním souborem s nastavením.

Kromě adresy Solr serveru a umístění konfiguračního souboru řídí konfigurace také sestavování řetězců procesorů a obsahuje tedy informace pro tovární

třídy, které řetězce sestavují. Řetězce jsou dva, proto jsou nastaveny ve dvou prefixech `indexer` a `cleaner`. V každém prefixu jsou pak uloženy informace pro jednotlivé části řetězce. Příklady sestavených řetězců jsou znázorněny na diagramu3.1. Zobrazené řetězce „indexing“ a „cleaning“ se sestavují ve výchozí konfiguraci, jejich tvorba je tedy definovaná ve vnitřním souboru *config.properties*.



Obrázek 3.1: Objektový model sestavených procesorových řetězů Crawleru

3.2.2 Sestavení

Kompilace aplikace je nastavena tak, aby se vytvořil JAR soubor obsahující všechny knihovny, na kterých aplikace závisí. K aplikaci jsem vytvořil skript

pro prostředí Windows i Linux, který zjednoduší její spouštění.

3.3 Frontend

Frontend podle pospaného návrhu sestává ze dvou statických HTML stránek oživených pomocí JavaScriptu. Pro usnadnění tvorby layoutu jsem využil CSS knihovny Bootstrap[13], která přináší jednotný a přehledný vizuální styl a definuje vizuální komponenty nad rámec prostého HTML.

Stránku jsem oživil pomocí JavaScriptové knihovny Angular.js[14], která mi umožnila provázat backend v podobě Solru s HTML frontendem za použití služeb pro JSON komunikaci se serverem a tzv. two way bindingu, mechanismu k provázání HTML DOM s JavaScriptovým modelem. Angular.js zastoupil i původní JavaScriptovou knihovnu Bootstrapu, protože umožňuje definovat mnohem jednodušším způsobem skrývání komponent a prvků v návaznosti na události.

Oproti popsanému návrhu v kapitole 2.3 jsem opatřil prvky ikonami ze sady volně použitelných univerzálních ikon FontAwesome[15], které usnadní vizuální navigaci uživatele po stránce. Ikonami jsem nahradil i původně zamýšlené zaškrtávací vstupní pole u jednotlivých shluků. Použil jsem ikonu adresáře, kterou uživatelé znají a která napoví, že položky znázorňují jisté skupiny dokumentů. Vybrané shluky se znázorní ikonou otevřeného adresáře. Číselné vyjádření počtu dokumentů ve shluku jsem umístil do komponenty *badge*, kterou je zvykem na internetu používat k tomuto účelu. Seznam shluků, do kterých patří dokumenty ve výsledcích, jsou umístěné ve vizuální komponentě *label*, kterou je zvykem používat například pro zobrazení štítků u položek. Výsledný layout lze vidět na obrázcích 3.2 a 3.3.

Aplikace počítá s nasazením aplikace Solr na stejném webovém serveru. Služby načítající data se přímo odkazují na request handlery */solr/searcher* a */solr/detail* definované v kapitole 3.1.2, a kromě toho získávají data o dostupných indexech pomocí vnitřního handleru */solr/admin/cores*.

Při implementaci jsem dbal na to, aby web odpovídal běžným zvyklostem i přesto, že je implementovaný v JavaScriptu. Všechny změny stavu aplikace jsou proto promítány i do adresního řádku. Díky tomu lze například zkopírováním adresy odkazovat na konkrétní výsledky vyhledávání, využívat funkcí zpět a vpřed v historii prohlížeče, či otevírat odkazy v novém okně pomocí běžných prostředků prohlížeče. Využívám k tomu standardní *Angular.js* službu *Route*, která zprostředkovává parsování i změny informací v adresním řádku.

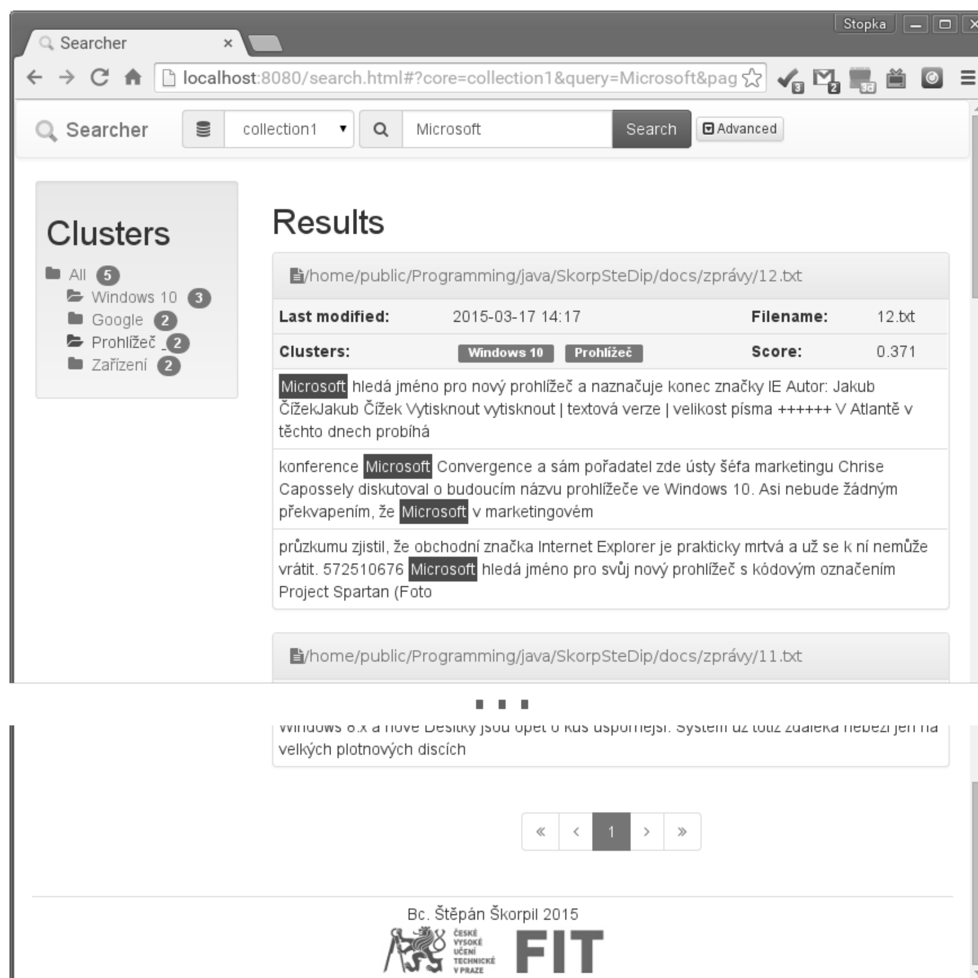
Aby mohl být frontend nasazen do servlet containeru, umístil jsem všechny statické soubory do adresáře *webapp* webové aplikace a vytvořil *HttpServlet* pojmenovaný *ViewFile*, jehož jedinou funkcí je odeslat jako odpověď na požadavek obsah souboru. V konfiguračním souboru *WEB-INF/web.xml* jsem servlet zaregistroval a namapoval na něj všechny url adresy.

Výsledná aplikace se kompiluje jako WAR soubor, který je možné nasadit do servlet containeru.

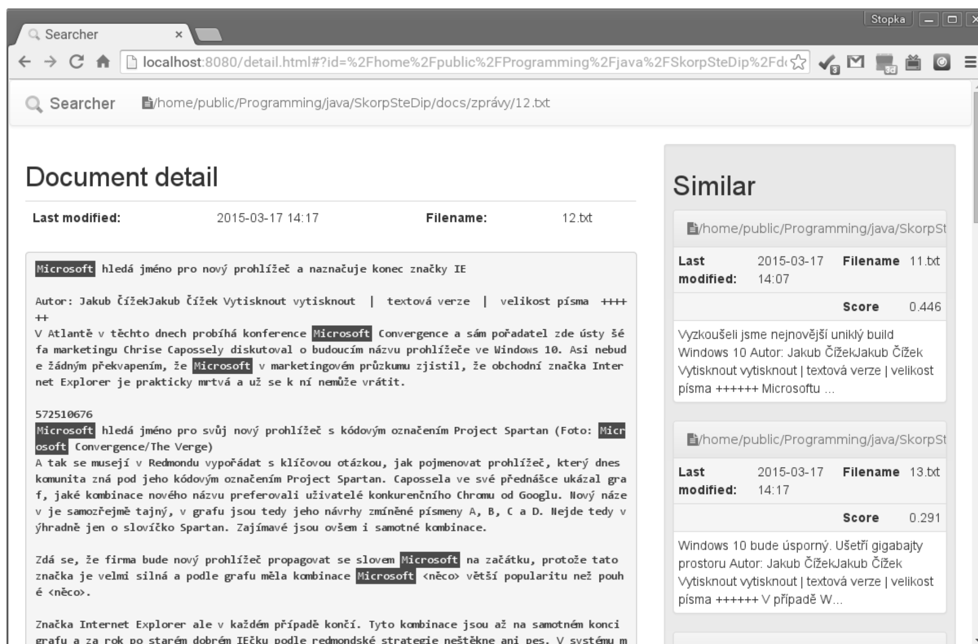
```
<servlet>
  <servlet-name>ViewFile</servlet-name>
  <servlet-class>
    cz.cvut.skorpste.dip.frontend.ViewFile
  </servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>ViewFile</servlet-name>
  <url-pattern>/*</url-pattern>
</servlet-mapping>
```

3.4 Jetty

Posledním krokem v rámci implementace řešení byla konfigurace serveru Jetty. Použil jsem instalaci distribuovanou se Solrem. Port serveru jsem nastavil na 8080 místo standardního 80, abych minimalizoval kolizi s jiným webovým serverem, a aby pro běh aplikace nebyla zapotřebí práva správce. Do aplikace jsem přidal webovou aplikaci *frontend* a nastavil *contextPath* na kořenový adresář serveru. Pro usnadnění spouštění aplikace jsem napsal krátké scripty pro prostředí Linux i Windows.



Obrázek 3.2: Snímek UI (Vyhledávač)



Obrázek 3.3: Snímek UI (Detail dokumentu)

Kapitola 4

Testování

4.1 Test stemmerů

4.1.1 Správnost kořenů

Tato část je věnována testování kvality jednotlivých implementovaných stemmerů. Helebrand pro účely testování svého stemmeru vytvořil seznamy slov, ve kterých ručně označil správné kořeny. V seznamu jsou u slov uvedeny dva kořeny, jeden lingvisticky naprosto správný, druhý méně správný, ale uznatelný tvar.

Seznamy jsou připraveny dva. Jeden obsahuje slova vybraná z reálného článku a umožní otestovat nasazení algoritmu v reálném prostředí. Druhý seznam slov je připravený speciálně pro otestování správné funkce stemmerů pro různé tvary slov včetně méně obvyklých tvarů a vyjímek. Obsažená slova jsou uvedena ve všech pádech a číslech. Slova jsou v seznamech tříděna do souborů podle slovních druhů, z výsledků tedy bude možné odvodit úspěšnost získávání kořene slova jednotlivých slovních druhů.

V jazyce Ruby jsem naprogramoval krátký jednoúčelový skript, který na základě parametrů načte příslušný seznam slov a zparsuje výchozí tvary slov a jejich označené kořeny. Jednotlivá slova pak nechá převést na jejich kořeny analyzérem s příslušným stemmerem tak, že je přímo pošle na REST rozhraní Solru. Získaný kořen pak porovná s označenými kořeny v seznamu. Průběžné výsledky zapíše do výstupního souboru, aby bylo možné provádět hlubší analýzy. Nakonec skript zobrazí a do výstupního souboru zapíše číselné hodnoty, které odpovídají celkovému počtu slov a počtu slov, která byla stemována na žádaný tvar. Tímto způsobem lze vyjádřit procentuální úspěšnost algoritmu.

Pomocí tohoto skriptu jsem otestoval všechny implementované české stemmery. Výsledky jsou uvedeny v tabulce 4.1. Z ní také můžeme vyčíst, že v tomto testu si vede nejhůře původní stemmer. Nejlépe pak dopadl Helebrand stemmer. Výsledky testu rovněž prokázaly, že slovníkový stemmer Hunspell není celkově příliš účinný, ale má velmi dobré výsledky v kategorii sloves — prav-

(%)	Podst. jm.		Příd. jm.		Slovesa		Vše	
Stemmery	**Čl.**	**Tst.**	**Čl.**	**Tst.**	**Čl.**	**Tst.**	**Čl.**	**Tst.**
cz	51,68	3,07	39,69	4,15	7,58	2,44	38,50	3,20
cz_light	60,40	3,80	46,56	2,63	11,36	2,64	45,63	3,40
cz_agressive	55,03	48,53	52,67	23,37	3,03	0,00	42,25	36,61
cz_helebrand	53,69	58,47	49,62	47,16	31,82	38,62	47,59	53,39
cz_hunspell	35,91	0,00	1,53	0,00	70,45	90,04	38,04	12,55
cz_hh	56,04	74,42	54,20	66,39	39,39	61,59	51,69	70,98

Tabulka 4.1: Výsledek testu správnosti kořenů

děpodobně proto, že jde spíše o lematizér, který vrací slova v základním tvaru, takže se utvořený tvar neshoduje s označným kořenem. Malá účinnost je zároveň důsledkem nízké kvality českého slovníku, neboť je třeba brát v potaz, že v době jejich vzniku byl určen ke kontrole pravopisu.

V rámci další analýzy výsledků algoritmů jsem procházel výstupní soubory s vytvořenými kořeny a zjistil jsem, že nenalezne-li Hunspell slovo ve slovníku, vrací původní tvar slova. Napadlo mě tedy zkusit spojit dva stemmery za sebe. Slovo by se nejdříve předložilo Hunspell stemmeru. Když nebude úspěšný, slovo zůstane v původním tvaru a bude možné jej předložit stemmeru dalšímu. Když bude úspěšný, vrátí slovo v základním tvaru podle slovníku a další stemmer v pořadí bude mít jednodušší práci s ořezáním slova na jeho kořen. V obou případech by měl být výsledkem kořen slova použitelný pro indexaci. V konfiguraci Solru jsem tedy vytvořil další analyzér obsahující dva stemmery. Jako druhý stemmer po slovníkovém jsem zvolil ten, který měl v testu nejlepší výsledek — Helebrandův.

Výsledný analyzér jsem otestoval stejným způsobem jako předešlé a zanesl výsledek do tabulky 4.1 pod názvem *text_cz_hh*. Analyzér dosáhl rekordní úspěšnosti 70,98%. Obě metody se dle očekávání doplňují. Slovník přispívá vysokou přesností, druhý stemmer možností univerzálního použití pro kterékoliv slovo.

4.1.2 Shodnost kořenů

Po další důkladné analýze výsledků předchozího testu jsem zjistil, že test nevypovídá dostatečně o vhodnosti nasazení stemmeru pro indexaci a vyhledávání. Pro nejlepší výsledky je nutné, aby stemmer tvořil pokud možno stejný kořen pro všechny tvary jednoho slova, což se v mnoha případech nedělo.

Připravil jsem tedy test nový. Odvodil jsem nový seznam slov, kde na řádku jsou vždy uvedeny různé tvary téhož slova. Upravil jsem také testovací algoritmus, který tentokrát postupně nechá všechny tvary jednoho slova zpracovat analyzérem Solru a porovná vytvořené kořeny mezi sebou. Poté spočítá, kolik

(%)	Podst. jm.		Příd. jm.		Slovesa		Vše	
Stemmery	**Pr.**	**Úsp.**	**Pr.**	**Úsp.**	**Pr.**	**Úsp.**	**Pr.**	**Úsp.**
cz	98,38	92,00	94,60	40,00	24,95	1,82	85,09	62,22
cz_light	87,40	38,00	81,38	3,64	23,50	1,52	75,08	21,11
cz_agressive	89,57	54,00	91,75	31,64	39,35	1,82	81,81	37,78
cz_helebrand	77,91	33,27	59,92	3,27	52,27	5,15	68,64	18,48
cz_hunspell	86,66	44,18	84,10	32,00	94,39	66,67	87,24	44,55
cz_hh	90,44	51,09	90,00	32,00	94,39	66,67	90,97	48,38
cz_hx	97,05	89,82	94,60	40,00	94,39	66,67	95,92	72,12

Tabulka 4.2: Výsledek testu shody kořenů

procent kořenů se shoduje. Čím více slov se tedy zpracuje do stejného kořene, tím vyšší číslo a tím lepší podal algoritmus výsledek na daném slově. Celkovou úspěšnost algoritmu odvozuji ze dvou čísel a to z průměrné úspěšnosti, získané aritmetickým průměrem úspěšnosti na jednotlivých slovech a úspěšnosti, počítané jako procento slov, u nichž se všechny tvary převedly na shodný tvar (jinými slovy na kolika procentech slov měl algoritmus stoprocentní úspěšnost).

Výsledky testu můžeme pozorovat v tabulce 4.2. První sloupec udává pro daný slovní druh průměrný počet nalezených shodných kořenů, druhý pro kolik slov byl nalezen shodný kořen pro všechny jejich tvary. Tento test ukazuje, že nejlepších výsledků dosahuje slovníkový Hunspell stemmer. Druhým nejlepším je původní stemmer zabudovaný v Solru.

Oba stemmery předstihuje pouze kombinace Hunspell a Helebrnd stemmeru, vytvořená po minulém testu, což potvrzuje funkčnost navrženého principu kombinování. Protože jsou výsledky tohoto testu odlišné od výsledků testu předchozího, vytvořil jsem nový analyzér s použitím nejúspěšnějších stemmerů v novém testu — Hunspell a původní stemmer. Nový kombinovaný analyzér jsem také podrobil testu a výsledky zanesl do tabulky pod jménem *text_cz_hx*. Nově vytvořený stemmer dosahuje nejlepších výsledků a zdá se tak být pro vyhledávání nejvhodnějším kandidátem.

V konfiguraci Solru jsem na základě výsledků testu nastavil pro pole *text* datový typ *text_cz_hx* sestávající z po sobě jdoucích *solr.Hunspell* stemmeru a *solr.CzechStemmer*.

4.2 JUnit

Všechny zásadní celky kódu jsou pokryty testy, které kontrolují integritu komponent pomocí *JUnit*. Testy budou užitečné při případném dalším rozšiřování. Pokud dojde budoucí úpravou k nechtěné změně funkčnosti existujících komponent, měly by to testy odhalit.

V aplikaci Crawler jsou takto pokryty všechny procesorové komponenty a crawlery. V rozšiřující Solr knihovně *Analyzery.jar* jsou pokryty všechny tři nové stemmery i s jejich továrními třídami.

Všechny napsané testy aktuálně hlásí stoprocentní úspěšnost.

4.3 Test použitelnosti

V tomto testu byla aplikace předložena skutečným uživatelům, aby otestovali navržené uživatelské rozhraní. Ověřím tak intuitivnost a funkčnost návrhu i implementace.

4.3.1 Výběr uživatelů

Testovací uživatele jsem volil především podle kritéria zkušenosti s prací s internetovými vyhledávači. Vybral jsem uživatele, kteří se běžně pohybují na internetu, umějí ovládat internetový prohlížeč, mají zažité zvyklosti z tohoto prostředí a běžně vyhledávají informace pomocí internetových vyhledávačů. Toto kritérium by mělo přesně vymezit skupinu uživatelů, pro kterou bylo uživatelské rozhraní určeno.

Uživatele jsem se rozhodl vybírat z okruhu svých přátel a to z několika důvodů. Jednak u nich mám představu o úrovni jejich znalostí a schopností, tudíž si mohu zvolit takové testovací uživatele, kteří spadají do výše zmíněné cílové skupiny. Takoví uživatelé ke mně budou během provádění testu také více otevření, tudíž se dozvím s menším úsilím více detailů o problémech, se kterými se potýkali.

4.3.2 Charakteristika uživatelů

Uživatelka Karolína je studentkou Filozofické fakulty Univerzity Karlovy. Počítač využívá k tvorbě školních prací, k práci s internetem, komunikaci i sledování multimédií. Je schopná na počítači provést i některé složitější konfigurační úkony, její uživatelské schopnosti tak hodnotím jako vysoké. Na internetu se pohybuje každodenně, často hledá informace skrze internetové služby, sleduje video portály, navštěvuje sociální sítě. Taktéž její zkušenost s internetem hodnotím jako vysokou.

Uživatelka Petra používá počítač pro prohlížení a zálohu fotografií, práci s dokumenty a práci na internetu. Konfigurační úkony jsou pro ni velkou neznámou. Na internetu využívá sociální sítě, e-mailové služby, zpravodajské weby a vyhledávače. Web používá i k prodeji a propagaci zboží. Její počítačové schopnosti jsem tedy vyhodnotil spíše jako průměrné, internetové jako vysoké.

Uživatel Miroslav je absolventem Strojní fakulty ČVUT, jeho počítačové zkušenosti považuji za průměrné. Počítač používá pouze pro psaní dokumentů a k práci na internetu. Internet používá převážně k e-mailové komunikaci.

Jméno	Věk	Pohlaví	Vzdělání	PC gram.	WWW gram.
Karolína	21	Žena	Maturita	Vysoká	Vysoká
Petra	46	Žena	Maturita	Průměrná	Vysoká
Ing. Miroslav	54	Muž	Vysokoškolské	Průměrná	Průměrná
Anna	21	Žena	Maturita	Vysoká	Vysoká
Vladimír	20	Muž	Maturita	Expertní	Expertní

Tabulka 4.3: Testovací uživatelé

Informace na internetu vyhledává v porovnání s předchozími uživatelkami méně často.

Anna je studentkou Ústavu translatologie na Filozofické fakultě Univerzity Karlovy. Počítač používá pro plnění školních povinností i pro zábavu. Na internetu používá sociální sítě, e-mailové služby, zpravodajské weby i vyhledávání. Ve vyhledávání na internetu je velmi pokročilá, denně při překladu vyhledává správná užití frází pomocí pokročilých technik dotazování internetových vyhledávačů.

Posledním uživatelem, který se účastnil testu, je Vladimír. Vladimír je studentem Technické univerzity v Liberci. V testu je nejpokročilejším uživatelem. Ve volném čase se zabývá programováním webových stránek i složitějších aplikací.

Základní orientační charakteristiku vybraných uživatelů znázorňuje tabulka 4.3.

4.3.3 Průběh testu

Testovacím uživatelům jsem stručně představil aplikaci, seznámil jsem je s účelem aplikace, v bodech jsem vyjmenoval možnosti, které nabízí, a vysvětlil jsem jim základní terminologii. Z celého průběh testu jsem se svolením uživatelů pořídil videonahrávku, abych mohl test zpětně analyzovat a vyvodit závěry. Test jsem prováděl s každým uživatelem zvlášť.

Do systému jsem pro účely testu nahrál data — do jednoho indexu dokumenty s uloženými články z technického zpravodajského webu, do druhého anonymizované příklady policejních dokumentů. Uživatele jsem nechal několik minut zkoumat prostředí, aby se sami zběžně seznámili s funkcemi. Poté jsem uživatelům zadal následující úkoly a instruoval je, aby své kroky, nahlas komentovali. Během plnění úkolu jsem nezasahoval, pouze jsem pasivně pozoroval jejich počínání.

1. Vyhledejte dokumenty týkající se firmy „Microsoft“.
2. Zjistěte, jaké shluky se ve výsledcích vyskytují.
3. Zjistěte, v jakých shlucích se nachází první výsledek.

4. Odfiltrujte pouze výsledky ve shlucích „Windows 10“ a „Google“.
5. Zobrazte znovu všechny výsledky hledání.
6. Otevřete první dokument ve výsledcích.
7. Otevřete dokument, který je otevřenému dokumentu nejpodobnější.
8. Vyhledejte v druhé kolekci dokumenty, které obsahují spojení „úřední záznam“.
9. Změňte mechanismus shlukování.
10. Zjistěte, kdy byly nalezené dokumenty naposledy upraveny.

Úkoly byly navrženy tak, aby prověřily především klíčové aspekty rozhraní, jako například zda uživatel bude vědět, jak se vrátit z detailu dokumentu, zda uživatel pochopí funkci filtrování podle shluků a zda pochopí možnost přepínání kolekcí či shlukovacích algoritmů.

Po dokončení testu jsem s každým uživatelem ještě diskutoval o návrzích na vylepšení a dotázal jsem se na jejich porozumění funkcím jednotlivých detailů prvků rozhraní. Zaměřil jsem se na následující body.

- Jestli vědí, co vyjadřuje číslo u každého shluku.
- Jestli bezpečně rozeznají otevřené shluky.
- Jestli dokáží zjistit, do jakých shluků dokument patří.
- Jestli vědí, jaké výsledky se zobrazují ve výchozím stavu.
- Jestli rozumí metadatům dokumentu.

Po každém testu jsem vždy rovnou navrhl a implementoval řešení nalezených kritických problémů, abych v dalších testech ihned viděl, zda má opatření účinek.

4.3.4 Výsledky testu

4.3.4.1 Karolína

Jako první jsem test provedl s uživatelkou Karolínou. Uživatelka neměla problém s rozeznáním funkcí uživatelského rozhraní. Největší problém nastal v bodě 4, kde měla vybrat výsledky jen z daných shluků. Uživatelku nenapadlo kliknout na více shluků a tím jich více vybrat. V následné diskusi jsem zjistil, že se tak stalo kvůli nedostatečnému zvýraznění otevřených shluků. Otevřený shluk nyní změní pouze přidruženou ikonu složky z běžné na otevřenou, což

uživatelka nepostřehla, nebo nebrala v úvahu. Zároveň jí zmátlo barevné označení odkazu po kliknutí. Prohlížeč Chrome totiž neodebírá správně *hover* efekt odkazu po odjetí myši z odkazu.

Na základě tohoto poznatku jsem dodatečně provedl drobnou úpravu frontendu. Řádek s otevřeným shlukem se označí nejen ikonou otevřeného adresáře, ale i změnou barvy. Tím by mělo dojít k dostatečnému zvýraznění otevřených shluků.

4.3.4.2 Petra

Následně jsem test provedl s uživatelkou Petrou. Ta se v prostředí také rychle zorientovala. Problém nastal opět u úkolu 4. Aktuální vizualizace otevřeného shluku stále dostatečně nenapovídá, že lze filtrovat výsledky podle více shluků.

Další problém nastal v bodě 9. Možnost přepnout shlukovací algoritmus hledala v prostoru bloku s nalezenými shluky a vůbec ji nenapadlo otevřít rozšířenou nabídku vyhledávání. Nabídku jsem dle zvyklostí z jiných vyhledávačů umístil vpravo od vyhledávacího pole. Rozšířenou nabídku uživatelka nepoužívá ani v běžných internetových vyhledávačích. Tento úkon naštěstí není nutný pro běžné vyhledávání, proto je nabídka běžně schovaná a proto také není velkým problémem, když ji uživatel hned nenajde.

Diskuzí jsme nakonec dospěli ještě k jedné nesrovnalosti. Při vyhledávání je každý výsledek zobrazen se třemi úryvky textu, ve kterých jsou zvýrazněna vyhledávaná slova. Každý úryvek je oddělen vizuálně novým řádkem tabulky a to se zdálo být matoucí.

Na základě výsledků tohoto testu jsem opět vyvodil drobné změny v návrhu UI. Znovu jsem opravil zvýraznění otevřených shluků v seznamu a nahradil původní ikonu otevřeného či zavřeného adresáře ikonou zaškrtnutého či nezaškrtnutého formulářového políčka *checkbox*. Uživatelé jsou zvyklí z formulářů na internetu, že zaškrtávací pole ve tvaru čtverce znamenají možnost vybrat více položek, tato úprava by jim tedy měla napovědět i v tomto případě.

Dále jsem se pokusil řešit problém s úryvky textu ve výsledcích. Existují dvě možnosti. Buď se zobrazí jen jeden delší úryvek, nebo se změní způsob zobrazení úryvků tak, aby byly méně matoucí. Rozhodl jsem se pro druhou možnost. Nadále se budou zobrazovat tři úryvky, neboť mi připadá výhodné zobrazit více relevantního textu s hledanými slovy, což pak usnadní hledání požadovaného dokumentu. Změní se však způsob zobrazení úryvků. Úryvky se nebudou zobrazovat ve třech samostatných řádcích, ale budou v jednom bloku odděleny obyčejnou textovou výpustkou.

4.3.5 Ing. Miroslav

Miroslav měl s použitím softwaru největší potíže. Největší problém představovalo najít schovanou nabídku přepínání shlukovacího algoritmu. Hledal ji

spíše v místě nabídky kolekcí, ale nakonec ji úspěšně našel. Navíc očekával, že změna algoritmu proběhne ihned při změně ve vstupním poli, a byl zmaten, když se tak nestalo.

Miroslav si také nebyl jistý funkcí filtrování podle shluků. Ihned ale pochopil, že může zaškrtnout i více položek, takže předchozí opravná opatření se zdají být úspěšná. Nebyl si však jistý jak se výsledek filtrování projevil. Nepostřehl, že se mezi výsledky zobrazují jen některé dokumenty. Rozhodl jsem se tedy provést ještě jednu drobnou změnu: u každé položky se zvýrazní barevně shluky, podle kterých se filtruje.

4.3.6 Anna

Anna zvládla všechny úkoly bez zjevného zaváhání. V následné diskuzi o proběhlém testu se pouze přiznala, že hned nevěděla, jak otevřít detail dokumentu. Zkoušela kliknout nejdříve na text dokumentu ve výsledcích, než si všimla, že je v hlavičce odkaz.

Zvažoval jsem, že bych jako odkaz na detail dokumentu označil celý box patřící položce výsledku. Tuto možnost jsem však nakonec zavrhl. Uživatelé by například mohli mít potíže s kopírováním textů a podobně.

4.3.7 Vladimír

Poslední test jsem provedl s uživatelem Vladimírem, přičemž s jeho zkušenostmi by se tento test dal označit spíše za expertní test. Uživatel neměl s použitím aplikace žádný větší problém. Jeho jedinou drobnou připomínkou bylo, že na stránce s detailem dokumentu hledal tlačítko pro návrat, než si uvědomil, že může použít funkce prohlížeče.

4.4 Akceptační test

Posledním provedeným testem byl akceptační test. Prošel jsem jednotlivé definované požadavky na systém a ověřil jejich splnění. Všechny požadavky, které jsem zařadil do první fáze projektu a kterým se věnuje tato práce jsou splněny. Ostatní požadavky jsou odloženy, jelikož nebylo v mých silách implementovat celé tak rozsáhlé zadání. Detailní rozpis je uveden v tabulce 4.4.

1.1.1 Funkční požadavky		
Požadavek		**Stav**
1	Indexace textových souborů	Splněno
2	Rozšíření o další index dokumentů	Splněno
3	Ukládání celého obsahu dokumentů	Splněno
4	Zpracovávání textu v českém jazyce	Splněno
5	Fulltextové vyhledávání v dokumentech	Splněno
5a	Tokenizace českého textu	Splněno
5b	Stemizace českého textu	Splněno
5c	Zobrazení obsahu nalezených dokumentů	Splněno
5d	Zvýraznění hledaných slov	Splněno
6	Zobrazení podobných dokumentů	Splněno
6a	Počítání podobnosti dokumentů	Splněno
7	Shlukování výsledků hledání	Splněno
7a	Shlukování do automatických kategorií	Splněno
7b	Filtrování výsledků podle shluků	Splněno
8	Rozpoznávání jmenných entit	Odloženo do dalších fází
8a	Rozpoznávání větných členů	Odloženo do dalších fází
8b	Rozpoznávání slovních druhů	Odloženo do dalších fází
8c	Porovnávání s jmennými databázemi	Odloženo do dalších fází
9	Extrakce vazeb mezi entitami	Odloženo do dalších fází
1.1.2 Nefunkční požadavky		
Požadavek		**Stav**
1	Čistě open source technologie	Splněno
2	Architektura klient-server jako webová služba	Splněno
3	Nativně nasaditelný pod Windows	Splněno
4	Rozšiřitelnost	Splněno

Tabulka 4.4: Výsledek akceptačního testu

Závěr

Podařilo se mi navrhnout a implementovat systém, který dokáže spolehlivě vyhledávat a shlukovat české textové dokumenty vyšetřovatelů. Jedná se zatím pouze o první část navrženého komplexního řešení, které by vyšetřovatelé potřebovali. K dalším částem bude zapotřebí integrovat další, ještě pokročilejší metody zpracování textu.

V práci jsem využil pouze open source technologie, především aplikace Solr a Carrot2, do kterých jsem implementoval nové nástroje pro zpracování českých textů. Speciálně jsem se zaměřil na nástroje pro získání kořenů slov, takzvané stemmery. Kvality těchto nástrojů jsem otestoval a vybral ty nejvhodnější pro účely vyhledávání v textu.

Mnou navržený český stemmer (respektive kombinace stemmerů) je dle mých testů o deset procent úspěšnější než původní stemmer obsažený v Solru. K tomu jsem navíc implementoval i další stemmery, z nichž jeden má vysokou úspěšnost tvorby správných lingvistických kořenů, což může být využito v dalších fázích implementace budoucího řešení zpracování textu.

K dosažení všech definovaných cílů jsem vytvořil několik přidružených aplikací, například *Crawler*, aplikace, která se k Solru připojuje a která má za úkol přidávání a odebírání dokumentů z indexu. Aplikaci se mi podařilo navrhnout plně modulární a tedy i do budoucna rozšiřitelnou.

Pro Solr jsem také vytvořil webový frontend pro uživatelské vyhledávání v dokumentech, postavený na moderních technologiích jako HTML5, CSS3 či Angular.js. Frontend je navržený s důrazem na intuitivnost, což bylo ověřeno uživatelským testováním.

Výsledný systém jsem se snažil koncipovat tak, aby mohl být dále rozšiřován o další moduly či aplikace tak, aby v dalších fázích projektu mohlo být na tuto implementovanou část navázáno. Tomu jsem podřídil výběr technologií i dalších prostředků vývoje.

Po celou dobu vývoje jsem své kroky konzultoval se zadavatelem projektu, aby výsledek co nejvíce vyhovoval jeho potřebám. I tak už nyní vyvstávají nové návrhy, jak aplikaci pro vyhledávání ještě dále vylepšit. Zadavatel by napří-

klad uvítal možnost vyhledat dokumenty i z několika kolekcí (myšleno indexů) naráz. Technicky by nový požadavek mohl být splněn například úpravou frontendu, který by dotaz rozeslal do vybraných kolekcí a výsledky sloučil. Tato možnost poměrně jednoduchého řešení dodatečných požadavků pouze úspěšně dokazuje rozšiřitelnost návrhu řešení.

Literatura

[1] Hanousek, J.: *Průvodce OBBB*. 2013. Dostupné z: http://sourceforge.net/projects/obbb/files/PruvodceOBBB.zip

[2] *NetBeans Platform Quick Start*. 2014. Dostupné z: https://platform.netbeans.org/tutorials/nbm-quick-start.html

[3] Foundation, A. S.: *Solr Wiki*. 2014. Dostupné z: http://wiki.apache.org/solr

[4] Foundation, A. S.: *Lucene-java Wiki*. 2014. Dostupné z: http://wiki.apache.org/lucene-java

[5] Foundation, E.: *Jetty : The Definitive Reference*. 2014. Dostupné z: http://www.eclipse.org/jetty/documentation/9.2.2.v20140723/

[6] Foundation, A. S.: *Apache Tomcat 7: Documentation*. 2014. Dostupné z: http://tomcat.apache.org/tomcat-7.0-doc/

[7] Osiński, S.; Weiss, D.: *Carrot2: User and Developer Manual*. 2014. Dostupné z: http://download.carrot2.org/stable/manual/

[8] Foundation, A. S.: *Apache Solr Reference Guide*. 2014. Dostupné z: https://archive.apache.org/dist/lucene/solr/ref-guide/

[9] Dolamic, L.; Savoy, J.: Indexing and Stemming Approaches for the Czech Language. *Inf. Process. Manage.*, ročník 45, č. 6, Listopad 2009: s. 714–720, ISSN 0306-4573, doi:10.1016/j.ipm.2009.06.001. Dostupné z: http://dx.doi.org/10.1016/j.ipm.2009.06.001

[10] Hellebrand, D.: Nalezení slovních kořenů v češtině. 2010. Dostupné z: http://www.fit.vutbr.cz/research/prod/index.php.cs?id=133

[11] Kolář, P.: *Czech dictionary pack*. 2008. Dostupné z: http://extensions.services.openoffice.org/en/project/czech-dictionary-pack-české-slovníky

[12] Vlček, L.: Elasticsearch: Vyhledáváme hezky česky (a taky slovensky). 2013. Dostupné z: http://www.zdrojak.cz/clanky/elasticsearch-vyhledavame-hezky-cesky-ii-a-taky-slovensky/

[13] *Bootstrap*. Dostupné z: http://getbootstrap.com

[14] *AngularJS*. Dostupné z: https://angularjs.org/

[15] *Font Awesome*. Dostupné z: http://fortawesome.github.io/Font-Awesome/

Příloha **A**

Seznam použitých zkratek

AJAX Asynchronous JavaScript and XML

API Application programming interface

CSS Cascading Style Sheets

ČVUT České Vysoké Učení Technické V Praze

DOM Document object model

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

ID Identifikátor

JAR Java archive

Java EE Java Platform, Enterprise Edition

JSON JavaScript Object Notation

OBBB Open Black Box Builder

PC Personal Computer

REST Representational State Transfer

STC Suffix Tree Clustering

UI User interface

URL Uniform Resource Locator

XML Extensible Markup Language

Java VM Java Virtual Machine

WAR Web application archive

WWW World Wide Web

Příloha B

Uživatelská příručka

B.1 Požadavky

Systém ke svému fungování potřebuje následující softwarové prerekvizity. Aplikace může fungovat i na jiných verzích, na těchto zde napsaných byla aplikace testována.

- Windows 7/8; Linux
- Java runtime 1.8 (na počítači *search server* a *file client*, viz kapitola2.1)
- Internetový prohlížeč, např. Chrome 42/Firefox 37 (na počítači *search client*, viz kapitola 2.1)
- Apache Maven 3 (pouze pro kompilaci)
- IntelliJ Idea (doporučeno, pouze pro kompilaci)

B.2 Instalace

1. Zkopírujte adresář s aplikací *searcher* do adresáře cílového počítače (*search server* - viz kapitola 2.1).
2. Zkopírujte adresář s aplikací *crawler* do adresáře cílového počítače (*file client* - viz kapitola 2.1)

B.3 Kompilace

Kompilaci mají všechny součásti velmi podobnou, proto zde uvedu jednotný návod.

1. V adresáři zdrojových souborů spusťte program Maven příkazem `maven clean build`.

2. Maven automaticky stáhne potřebné knihovny a sestaví knihovnu, webový archiv či java aplikaci. Hotový archiv naleznete v adresáři */target/* příslušné aplikace.

B.3.1 Sestavení

- Aplikace *crawler* je samostatnou aplikací běžící v Java runtime. Všechny závislosti se při kompilaci přibalí do výsledného aplikačního archivu *crawler-jar-with-dependencies.jar*
- Knihovnu *Analyzery* nainstalujete do solru zkopírováním výsledného souboru*Analyzery.jar* do adresáře */searcher/solr/lib/* serverové aplikace.
- *Frontend* nainstalujete do solru zkopírováním výsledného souboru *frontend.war* do adresáře */searcher/webapps/*.

B.4 Searcher

B.4.1 Spuštění

Aplikaci spustíte zavoláním skriptu *run.bat* na Windows, respektive *run.sh* na Linuxu. Po každé úpravě konfigurace je nutné aplikaci restartovat.

B.4.2 Konfigurace

B.4.2.1 Změna portu

Síťový port nastavíte v souboru */searcher/etc/jetty.xml* změnou hodnoty označené textem *[PORT]* v následujícím úryvku. Ve výchozím stavu je port nastaven na *8080*.

```
<Call name="addConnector">
  <Arg>
    <New class="org.eclipse.jetty.server.bio.SocketConnector">
      <Set name="port">
        <SystemProperty name="jetty.port" default="[PORT]"/>
      </Set>
    </New>
  </Arg>
</Call>
```

B.4.2.2 Přidání indexu

Vytvořte kopii adresáře */searcher/solr/collection1* pro český index, nebo */searcher/solr/en_collection1* pro anglický. Nový adresář vhodně pojmenujte, například */searcher/solr/collection2*. Upravte jméno kolekce změnou hodnoty

`name=collection2` v souboru */searcher/solr/collection2/core.properties*. Jméno kolekce musí být unikátní.

B.4.2.3 Změna stemmeru

Nahraďte *[ANALYZER]* v úryvku souboru
/searcher/solr/[KOLEKCE]/conf/schema.xml jménem analyzéru s požadovaným stemmerem.

```
<field name="text" type="[ANALYZER]" indexed="true"
       stored="true" termVectors="true" />
```

Přípustné hodnoty jsou *text_cz*, *text_cz_light*, *text_cz_agressive*, *text_cz_helebrand*, *text_cz_hunspell*, *text_cz_hh*, *text_cz_hx*. Výchozí hodnota je *text_cz_hx*. Po změně je nutné přeindexovat všechny soubory.

B.4.2.4 Změna výchozího shlukovacího algoritmu

Nahraďte *[ENGINE]* v úryvku souboru
/searcher/solr/[KOLEKCE]/conf/solrconfig.xml jménem požadovaného shlukovacího enginu. Slovo *[KOLEKCE]* v cestě nahraďte jménem adresáře s kolekcí, jejíž chování chcete upravit.

```
<requestHandler name="/searcher" startup="lazy"
                enable="${solr.clustering.enabled:false}"
                class="solr.SearchHandler">
  <lst name="defaults">
    <str  name="clustering.engine">[ENGINE]</str>
  </lst>
</requestHandler>
```

Přípustné hodnoty jsou *lingo* (výchozí), *stc*, *kmeans*.

B.4.2.5 Přidání stop slov

Do seznamu stop slov lze přidat nové slovo vytvořením nového řádku se slovem v konfiguračním souboru
/searcher/solr/[KOLEKCE]/conf/lang/stopwords_[JAZYK].xml. Slovo *[KOLEKCE]* v cestě nahraďte jménem adresáře s kolekcí, jejíž chování chcete upravit. Slovo *[JAZYK]* pak mezinárodním dvoupísmeným kódem jazyka kolekce.

B.5 Crawler

Crawler se spuští zavoláním skriptu *run.bat* na Windows, respektive *run.sh* na Linuxu. V rámci jednoho běhu se provedou dvě fáze: *cleaning* a *indexing.*

Indexing fáze prochází adresáře, hledá nové soubory a všechny soubory zanese do indexu. Existuje-li již soubor v indexu, aktualizuje jej. *Cleaning* fáze projde všechny dokumenty v indexu a pokud již soubor neexistuje v adresáři, z indexu jej smaže.

Veškerá konfigurace se provádí skrze konfigurační soubor či vstupní parametry. Ve výchozím stavu se hledá konfigurační soubor *crawler.properties* v adresáři spuštění aplikace. Cesta ke konfiguračnímu souboru se může změnit přidáním parametru *config*. Výsledný příkaz pro spuštění pak vypadá například následovně:
`run.sh -Dconfig=cesta/k/souboru/nastavení.properties`

Konfigurační soubor má formu textového souboru, kde každý parametr je na novém řádku. Parametry jsou ve tvaru `proměnná=hodnota`. Chcete-li měnit chování programu pomocí vstupních parametrů, musíte je vložit za jméno skriptu *run.sh*, každý uvozený znaky `-D`, a jednotlivé parametry oddělit mezerou. Příklad najdete níže.

Chcete-li provádět indexování adresářů v pravidelných intervalech, musíte si pomocí nástrojů vašeho operačního systému nastavit spuštění skriptu i s konfiguračními parametry.

Základní spuštění pak vypadá takto:

```
run.sh -Dsolr.host=http://localhost:8080/solr/collection1
 -Dindexer.crawler.file_system.paths=~/documents,~/files
```

Parametry určují adresu solr serveru a kolekce, ke které se má připojit a seznam adresářů, které má indexovat. Další volby lze nastavit přidáním dalších parametrů.

B.5.1 Vypnutí fáze

Pro vypnutí *cleaning* fáze aktivujte parametr `cleaner.crawler=none`. Pro vypnutí *indexing* fáze parametr `indexer.crawler=none`.

B.5.2 Parsování dokumentů

Lze specifikovat, jakým způsobem se budou získávat data z nalezených souborů. Jsou dvě možnosti. Výchozí je použití *txt*, které načítá všechny soubory jako UTF-8 textové soubory. Přidáním parametru
`indexer.factory.txt.charset=windows-1250` lze nastavit kódování *windows1250* pro načítání souborů. Obdobným způsobem lze zvolit i jiné kódování.

Druhou možností je použití *Tika* knihovny, která umí získat text i ze složitějších formátů dokumentů, jako jsou *DOC*, *RTF* a podobně. Knihovna sama rozpozná typ a kódování souborů. Tuto možnost aktivujete pomocí parametru `indexer.factory=tika`.

B.5.3 Úplné vyčištění indexu

Přidáním parametru `cleaner.filter=all` se ve fázi čištění nebude kontrolovat existence souborů, ale odstraní se všechny dokumenty z indexu.

B.6 Frontend

Frontend je k dispozici hned po spuštění searcher serveru. Připojíte se k němu pomocí internetového prohlížeče na nastavené adrese
`http://<ip adresa serveru>:<port>/`. Ve výchozím stavu je port nastaven na *8080*.

Příloha C

Obsah přiloženého CD

```
SkorpSte_cd
├── README.txt .................. Stručný popis obsahu CD
├── applications ................ Adresář se spustitelnou formou implementace
│   ├── crawler ................. Klientská aplikace pro správu indexu
│   └── searcher ................ Serverová aplikace
├── documentation ............... Dokumentace aplikace
│   ├── analyzery ............... Vygenerovaný JavaDoc Sorl knihovny Analyzery
│   ├── crawler ................. Vygenerovaný JavaDoc aplikace Crawler
│   └── thesis .................. Text diplomové práce
└── sources ..................... Zdrojové kódy
    ├── analyzery ............... Zdrojový kód Solr knihovny Analyzery
    ├── crawler ................. Zdrojový kód aplikace Crawler
    ├── frontend ................ Zdrojový kód webové aplikace FrontEnd
    ├── stemmer_tester .......... Aplikace pro testování kvality stemmerů
    ├── thesis .................. Zdrojový text diplomové práce ve formátu LaTeX
    └── uml ..................... Zdrojové soubory UML diagramů pro aplikaci UMLet
```