Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Webové rozhraní pro sémantické vyhledávání v databázi závěrečných prací

Bc. Tomáš Šorejs

Vedoucí práce: Ing. Tomáš Bartoň

6. ledna 2015

Poděkování

V první řadě bych chtěl poděkovat svému vedoucímu Ing. Tomáši Bartoňovi za nekonečnou trpělivost a cenné rady při vedení mé práce. Dále bych chtěl poděkovat BlueberryApps za to, že mi dali šanci splnit si jeden z dalších snů a to živit se programováním. V neposlední řadě bych rád poděkoval mým učitelům ruby Bc. Tomáši Dundáčkovi, Ing. Martinovi Magnuskovi, Lukáši Vodovi a ostatním. Nakonec bych rád poděkoval rodinně, jmenovitě Mílovi, Dagmarce, Janě a Vášovi, a kamarádům za hlubokou podporu v dobách zlých i dobrých mého studia.

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 6. ledna 2015 .....

Odkaz na tuto práci

Šorejs, Tomáš. *Webové rozhraní pro sémantické vyhledávání v databázi závěrečných prací.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Tato práce pojednává o vyhledávání v databázi závěrečných pracích. V úvodu porovnává dostupná řešení vybraných škol. Na základě tohoto porovnání následuje návrh a implementace vlastní vyhledávácí aplikace za pomocí frameworku Ruby on Rails a vyhledávací technologie ElasticSearch. Tato aplikace klade důraz na rychlost a uživatelskou přívětivost vyhledávání. V závěru práce je popsáno testování aplikace, porovnání se současnými řešeními a finanční i nefinanční zhodnocení.

Klíčová slova vyhledávání, webová aplikace, ruby on rails, elasticsearch, dolování dat, prohledávání textu

Abstract

This work focuses on searching in the database of graduation theses. First it compares solutions that are used by selected universities. Based on the results of this comparison a search application is designed and implemented using Ruby on Rails framework and ElasticSearch technology. The application is designed to provide a quick and user-friendly search. Then this work describes testing of the application. At the end of this work, the application is compared

to the known solutions, and evaluated with respect to both financial and non-financial aspects.

Keywords searching, web application, ruby on rails, elasticsearch, data mining

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Všechny závěrečné práce na ČVUT i ostatních univerzitách, které v současnosti vznikají, mají kromě papírové podoby v archivu i podobu elektronickou. Tento fakt nám umožňuje nové práce elektronicky archivovat a pokud se univerzita rozhodne, tak i tento archív zpřístupnit veřejnosti. Jelikož prací vzniká velké množství, vyvstává otázka, jak se v tomto archívu orientovat a jak efektivně naleznout požadované informace. Právě touto otázkou a jejím řešením se zabývá tato diplomová práce.

Motivací pro vznik této práce bylo vytvoření nového archívu závěrečných prací pro ČVUT. Fakulta informačních technologií v současnosti nemá vlastní systém pro vyhledávání v závěrečných pracích, ani neexistuje podobné řešení v rámci ČVUT. Zastaralé řešení z Fakulty elektrotechnické má řadu nedostatků, které komplikují nalezení prací z předchozích ročníků a tudíž není jednoduché navázat na práci kolegů nebo vyhledat potencionálního vedoucího závěrečné práce.

Moderní systémy pro fulltextové vyhledávání umožňují jednoduché vyhledávání v databázi dokumentů. Díky znalosti struktury dokumentů a integraci meta informací můžeme tento systém vyhledávání ještě vylepšit.

Cíle práce

Cílem této práce je popsat problematiku vyhledávání v závěrečných pracích a následně navrhnout a implementovat systém, který bude odpovídat současným standardům pro vyhledávání.

Ke splnění tohoto cíle práce je potřeba vyřešit následující problémy:

- popsat současná řešení
- navrhnout vlastní řešení
- implementovat vlastní řešení
- importovat reálná testovací data

- otestovat funkcionalitu
- porovnat vlastní řešení se stávajícími
- zhodnotit finanční náročnost implementace, nákladů na údržbu a benefity řešení

Struktura dokumentu

V první kapitole 1 uvádím popis problému. Nejdříve popíšu a porovnám současná řešení pro archivaci a vyhledávání závěrečných prací a na základě jejich porovnání stanovím požadavky na výsledné řešení. Na závěr popisuji, které problémy budeme muset pro analýzu a vývoj aplikace vyřešit.

Další kapitola 2 se věnuje analýze a návrhu řešení. V této kapitole analyzuji požadavky, popisuji aktéry systémy a jejich případy užití. Chování složitějších případů užití popisuji detailněji v další části. Dále se věnuji analytickému modelu aplikace, jeho třídám, jejich atributům a společným vazbám. Nakonec popíšu možné technologie pro řešení stanovených problému a odůvodním výběru těch, které pro naší aplikaci použijeme.

V kapitole Implementace 3popisuji standardní rozvržení aplikace v Ruby on Rails, základní postup při vytváření mé aplikace a na závěr se detailněji věnuji řešení zajímavých problému, na které jsem narazil u implementace.

Následuje kapitola s vyhodnocením vlastního řešení 4 . V této kapitole porovnám implementované řešení se řešením stávajícím, navrhnu možnosti rozšíření aplikace, přidám ukázky aplikace a popíšu jaký měla aplikace finanční a nefinanční přínos.

Kapitola 1

Popis problému

V této kapitole popisuji problém řešený v této práci. Na začátku se věnuji již existujícím řešením. V druhé části popisuji problematiku vlastního řešení problému.

1.1 Současná řešení

1.1.1 Archív diplomových prací ČVUT

Jediným archívem pro závěrečné práce na ČVUT je v současné době systém Archív diplomových prací ČVUT.

Tento systém umožňuje vkládání jednotlivých prací, vyhledávání a stahování dokumentů. Systém by měl obsahovat všechny závěrečné práce z fakulty elektrotechnické a z fakulty informačních technologií od roku 2005 respektive 2012. Ostatní fakulty do tohoto systému práce nezadávají.

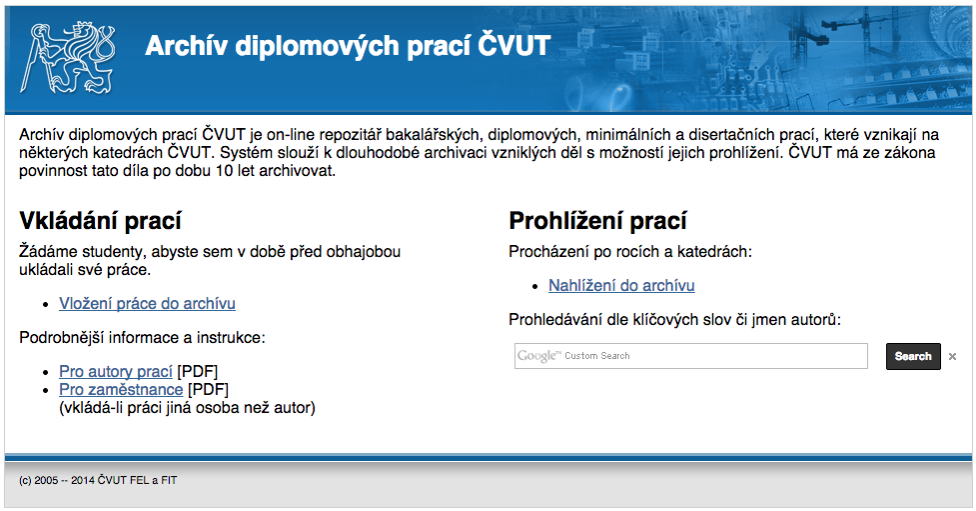
Náhled systému zobrazuje 1.1.

odkaz: `https://dip.felk.cvut.cz/`

1.1.1.1 Popis funkcionalit systému

Vkládání prací

1. Uživatel si vybere fakultu
2. Uživatel se přihlásí pomocí hesla ČVUT
3. Uživatel může vyplnit katedru, jméno autora, vedoucího práce, druh práce, český a anglický název a český a anglický abstrakt.
4. Po odeslání formuláře může uživatel nahrát práci ve formátu pdf a následně ji uložit.



Obrázek 1.1: Náhled Archívu diplomových prací ČVUT

Vyhledávání K vyhledávání lze použit integrovaný nástroj Google Custom Search. Vyhledávání probíhá nad výše uvedenými attributy a nad obsahem pdf dokumentu. Výsledky vyhledávání mohou odkazovat na detail práce nebo pdf dokument.

Další možností vyhledávání je průchod archívu:

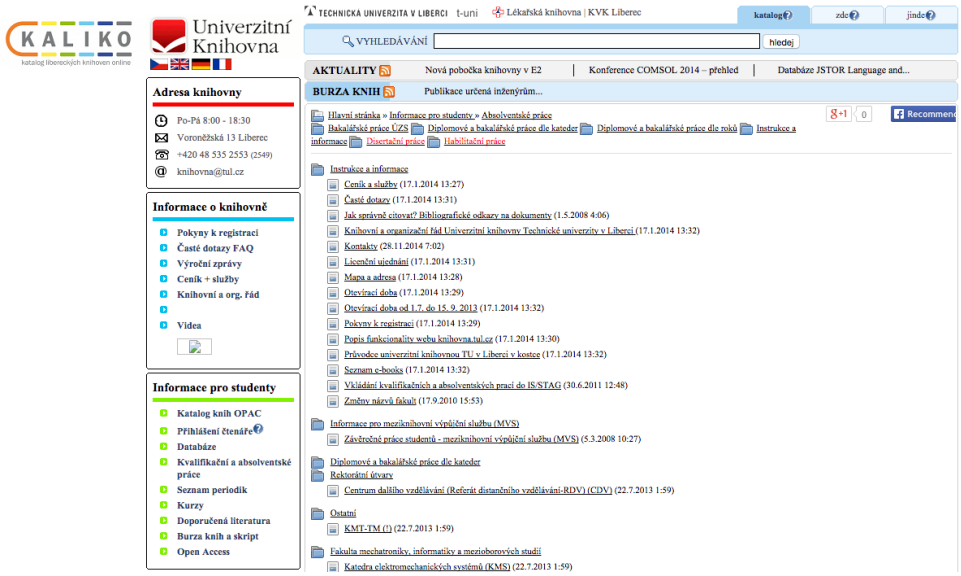
1. Uživatel vybere fakultu
2. Uživatel vybere katedru
3. Uživatel vybere rok
4. Uživateli je zobrazen seznam prací se jménem autora, názvem a typem práce. Uživatel má možnost zobrazit detail práce.
5. Na stránce detailu práce jsou všechny atributy práce a možnost stažení pdf dokumentu.

1.1.2 Řešení jiných škol

1.1.2.1 Technická univerzita Liberec

Tento sytém obsahuje nefukční vyhledávací pole, které ovšem nefunguje. Vyhledávání funguje tedy pouze průchodem dle kategorií. Na hlavní stránce si uživatel vybere fakultu a následně katedru. Po volbě katedry se mu zobrazí všechny závěrečné práce pro danou fakultu seřazené dle roku přidání.

Detail práce obsahuje informaci o počtu stran, ostatní autory a abstrakt. Nepřihlášený uživatel nemůže práci zobrazit. Kdokoliv může přidat komentář k práci.



Obrázek 1.2: Náhled vyhledávání Technické univerzity Liberec

Náhled systému zobrazuje 1.2.
odkaz: `http://knihovna.tul.cz/cat/Absolventske-prace-45.php`

1.1.2.2 Repozitář závěrečných prací Karlovy univerzity

Systém Karlovy univerzity obsahuje fulltextové vyhledávání nad názvem, abstraktem a obsahem práce. Vyhledávání je možno dále specifikovat použitím filtrů dle typu práce(bakalářská, diplomová, rigorózní, disertační), fakulty, roku obhajoby, typu dokumentu a jazyku.

Detail práce obsahuje informace u autorech, id práce, informace o studiu autora, hodnocení a datum obhajoby. Dále je možné stáhnout dokument práce i posudky oponenta a vedoucího.

Systém obsahuje 59994 prací a 154723 dokumentů.

Náhled systému zobrazuje 1.3.
odkaz: `https://is.cuni.cz/webapps/zzp/search/`

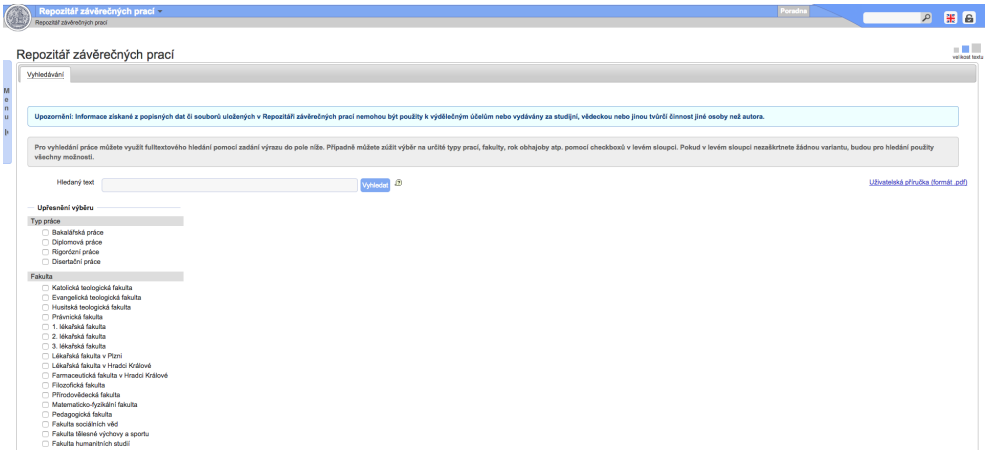
1.1.2.3 Informační systém Masarykovy univerzity

Systém má fulltextové vyhledávání, které vyhledává nad názvem, klíčovými slovy a abstraktem.

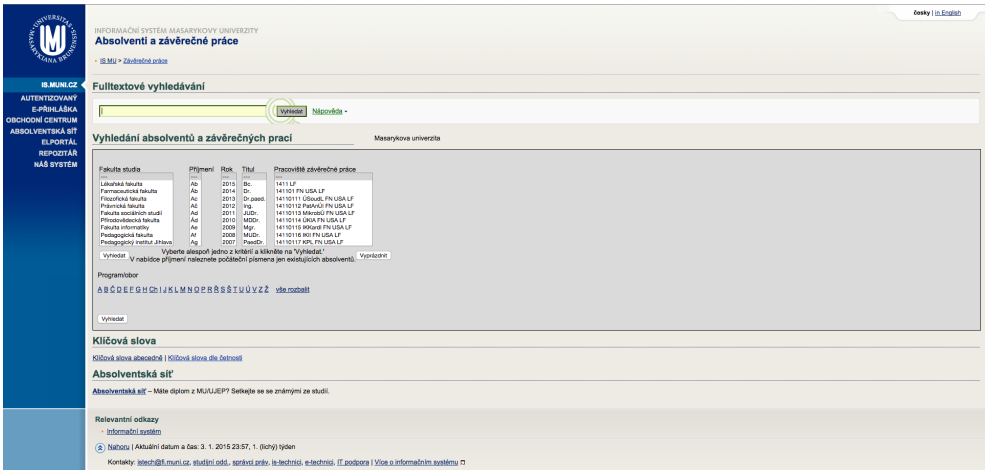
Dále systém vyhledává za použití filtrů dle fakulty, prvních dvou písmen příjmení autora, roku obhajoby, titulu autora a oboru studia autora.

Další možností vyhledávání jsou odkazy na klíčová slova prací seřazené dle četnosti použití.

Výše uvedené možnosti vyhledávání mezi sebou nelze kominovat.



Obrázek 1.3: Náhled Repozitáře závěrečných prací Karlovy univerzity



Obrázek 1.4: Náhled Informačního systému Masarykovy univerzity

Náhled systému zobrazuje 1.4.
odkaz: `http://is.muni.cz/thesis/`

1.1.3 Porovnání jednolivých řešení

Pro porovnání jednotlivých řešení jsem se zaměřil pouze na vyhledávání. Možnosti vyhledávání porovnávám v tabulce 1.1. K porovnání jsem použil čtyři vybrané vlastnosti vyhledávání, které nejčastěji popisuji výše. A to vyhledávání dle průchodu kategoriemi, fulltextové vyhledávání nad atributy práce, další filtry(rok, katedra, typ práce..) a zda je možné použít kombinaci předchozích vlastností.

Tabulka 1.1: Srovnání jednotlivých řešení pro vyhledávání

škola	**kategorie**	**fulltext**	**filtry**	**kombinace předchozích**
ČVUT	ano	ano	ne	ne
TUL	ano	ne	ne	ne
CUNI	ne	ano	ano	ano
MUNI	ne	ano	ano	ne

1.1.4 Závěr

V této části jsem chtěl uvést problematiku vyhledávání, kterou se zabývá tato práce a představit různá řešení tohoto problému. Všechny popisované systémy nedosahují kvalit dnešních běžných vyhledávacích systému. Nejvíce se jim přibližuje systém Karlovy univerzity, který má velice zastaralé uživatelské rozhraní a vyhledávání je značně pomalé. Co se týče možností a způsobu vyhledávání, je v porovnání s ostatními systémy jasně nejlepší. Díky porovnání s ostatními řešeními jsem chtěl poukázat na potřebu vylepšení řešení pro ČVUT, jenž budu popisovat v dalších částech této práce.

1.2 Požadavky na vyhledávání v závěrečných pracích

Při porovnání jednotlivých výše popsaných řešení vyhledávání jsem stanovil základní vlastnosti na náš vyhledávací systém:

- fulltext vyhledávání nad zvolenými parametry
- fulltext vyhledávání nad obsahem dokumentu
- vyhledávací filtry (rok, typ práce, katedra, autor..), které lze použít dohromady s fulltextovým vyhledáváním
- automatické i manuální označení práce předem definovanými textovými značkami – tagy
- možnost stažení dokumentu a zpětná úprava některých atributů práce autorem i po vložení práce
- uživatelsky přívětivé UI
- rychlost

1.3 Popis řešených problémů

V této části uvedu problémy, které budeme muset vyřešit při návrhu a vývoji našeho vyhledávače.

1.3.1 Architektura aplikace

Nejprve musíme navrhnout základní scénáře průchodu aplikací a s nimi navrhnout uživatelské role. Problémem zde bude vazba mezi uživateli aplikace a autory prací a s nimi spojená práva na jednotlivé scénáře například editaci vložených prací. Dále také budeme musíme vytvořit administrátorské rozhraní pro přidávání a editaci atributů, které nebudou mít možné spravovat všechny uživatelské role. V neposlední řadě budeme muset navrhnout prvky vyhledávání a jejich propojení s vyhledávacím nástrojem a databázovým systémem.

1.3.2 Testovací data

Při návrhu této práce nemáme přístup k žádné databázi závěrečných prací. Testovací data proto budeme muset získat z jediného dostupného archívu ČVUT, který jsem popsal výše.

1.3.3 Výběr technologií

Ze zadání vyplývá, že musíme použít open-source fulltextový vyhledávací systém, další požadavky na technologie nejsou jasně dané.

Je potřeba zvolit technologie a systémy pro:

- databázový systém
- backend rozhraní
- frontend rozhraní
- vyhledávání
- parsování dat z pdf
- přihlašování a autorizaci uživatelů
- nástroj pro dolování dat z archívu ČVUT

Kapitola 2

Analýza a návrh řešení

Tato kapitola obsahuje analýzu a návrh systému na vkládání a vyhledávání v závěrečných pracích. Návrh systému jsem provedl na základě srovnání existujících řešení a popisu požadavků na vyhledávací systém v sekci 1.2.

2.1 Analýza požadavků

2.1.1 Funkční požadavky

- systém umožní uživatelům vyhledávat v databázi závěrečných prací
- systém umožní uživatelům přidávat a editovat jejich vlastní práce
- systém bude na základě dokumentu práce automaticky generovat tagy
- systém umožní uživatelům stahovat dokument
- systém umožní registraci, přihlášení a odhlášení uživatelů
- systém bude stahovat všechny práce pro Fakultu informatiky z Archívu diplomových prací ČVUT 1.1.1

2.1.2 Nefunkční požadavky

- systém bude používat open-source technologie pro vyhledávání
- systém bude odpovídat v reálném čase[1] i pro více uživatelů
- systém bude mít responzivní webdesign[2] design
- systém bude dostupný jako webová aplikace

[1] cca 400ms

[2] způsob stylování HTML dokumentu, které zaručí, že zobrazení stránky bude optimalizováno pro všechny druhy mobilních zařízení

- systém bude napsán tak, aby by byl snadno rozšiřitelný
- systém bude navržen hlavně pro užití na ČVUT

2.1.3 Aktéři systému a případy užití

2.1.3.1 Aktéři systému

Aktéry systému jsme rozdělili do čtyř skupin:

- nepřihlášený uživatel
- přihlášený uživatel
- administrátor
- správce aplikace

2.1.3.2 Případy užití

Nepřihlášený uživatel Nepřihlášený uživatel, tedy kdokoliv kdo přijde na stránky našeho systému, bude moci provádět tyto činnosti:

- UC01 vyhledávat v databázi prací
- UC02 zobrazit detail jednotlivých prací
- UC03 stáhnout dokument práce
- UC04 registrovat se do systému
- UC05 přihlásit se do systému

Přihlášený uživatel Přihlášený uživatel je uživatel, který se přihlásil svým uživatelským jménem, jenž jsou v tomto případě email a heslo. Tyto údaje získal buď registrací nebo mu je vytvořil administrátor. Tento uživatel má dostupné tyto činnosti:

- UC01 vyhledávat v databázi prací
- UC02 zobrazit detail jednotlivých prací
- UC03 stáhnout dokument práce
- UC06 přidat svoji vlastní práci
- UC07 editovat svoji vlastní práci
- UC08 automaticky otagovat svoji vlastní práci
- UC09 odhlásit se ze systému

Administrátor Administrátor je přihlášený uživatel s administrátorskými právy, tato práva mu byla přiřazena jiným administrátorem. Administrátor má dostupné tyto činnosti:

- UC01 vyhledávat v databázi prací
- UC02 zobrazit detail jednotlivých prací
- UC03 stáhnout dokument práce
- UC06 přidat svoji vlastní práci
- UC09 odhlásit se ze systému
- UC10 editovat všechny práce
- UC11 automaticky otagovat jakoukoliv práci
- UC12 přidat, mazat a editovat uživatele
- UC13 přidat, mazat a editovat autory
- UC14 přidat, mazat a editovat katedry
- UC15 přidat, mazat a editovat fakulty
- UC16 přidat, mazat a editovat katedry
- UC17 přidat, mazat a editovat tagy

Správce aplikace Správce aplikace je uživatel, který má SSH přístup na aplikační server. Ten má k dostupné tyto činnosti:

- UC18 automatické stažení diplomových prací z Archívu diplomových prací ČVUT 1.1.1
- UC19 automatické otagování všech prací uložených v systému

2.1.3.3 Popis vybraných případů užití

UC01 vyhledávat v databázi prací Vyhledávání v závěrečných prací bude probíhat fulltext vyhledáváním nad názvem, textem dokumentu a tagy práce. Dále bude mít uživatel k dispozici filtry dle roku, druhu, fakulty nebo katedry, autora, vedoucího, oponenta a tagy práce. Filtry a fulltextové vyhledávání bude uživatel mít možnost kombinovat.

UC02 zobrazit detail jednotlivých prací Na detailu práce budou zobrazeny atributy:

- název
- druh
- jméno fakulta
- jméno katedry
- rok
- jméno autora
- jméno vedoucího
- jméno oponenta
- text abstraktu
- seznam tagů

UC04 registrovat se do systému Uživatel se do systému registruje pomocí emailu a hesla.

UC06 přidat svoji vlastní práci Uživatel přidá svoji vlastní práci a vyplní název, druh, katedru, rok a autory práce. Dále přidá PDF dokument s prací.

UC07 editovat svoji vlastní práci Po vložení práce může uživatel editovat pouze práce na kterých je uveden jako autor. Editovat může všechny parametry práce a dále může doplnit abstrakt a tagy. Po vložení již nelze znovu nahrát dokument práce.

UC08 automaticky otagovat svoji vlastní práci Na detailu práce bude uživateli, který je uveden jako autor práce, umožněno nechat svou práci automaticky otagovat na základě obsahu dokumentu a předem definovaných tagů.

UC18 automatické stažení diplomových prací z Archívu diplomových prací ČVUT 1.1.1 Aplikace bude moci automaticky stáhnout všechny dostupné závěrečné práce z archívu ČVUT automatickým procházením webu.

2.2 Výběr technologií

2.2.1 Programovací jazyk

Zadání nijak nespecifikovalo programovací jazyk, jelikož je náš program webovou aplikací, je to jediné omezení, které programovacím jazykům zadání aplikace dává. S vedoucím práce jsme se dohodli na programovacím jazyku Ruby, protože ho oba dva známe a chceme použít webový framework Ruby on Rails[3].

Dalšími možnými řešeními by byl například jazyk php a použití nějakého frameworku nebo Java Enterprise Edition.

2.2.2 Framework

Výběr jazyka byl ovlivněn právě tím, že jsme chtěli použít framework Ruby on Rails. Ruby on Rails je nejpoužívanější open-source webový framework pro jazyk Ruby. Dalšími konkurenty pro jazyk Ruby by v našem případě mohli být frameworky Sinatra nebo Rack, který je používán i v Ruby on Rails.

2.2.3 Databázový systém

Knihovna ActiveRecord, jenž slouží jako modelová vrstva frameworku Ruby on Rails, podporuje databázové systémy MySQL, PostgreSQL, SQLite, SQL Server a Oracle(všechny podporované databáze kromě DB2)[1]. Dále existuje spousta dalších knihoven pro podporu ostatních databázových systémů, takže jsme nebyli ve výběru databáze nijak omezeni. Nakonec jsme vybrali PostgreSQL, relační databázový open-source systém, který plně podporuje ACID[4] a je nyní nejpoužívanějším databázovým řešením pro aplikace vyvíjené v Ruby on Rails.

2.2.4 Nahrávání souborů

Pro nahrávání souborů(PDF práce) uživatelem do aplikace použijeme ruby knihovnu CarrierWave[5] v úvahu by přicházela celá řada nástrojů například Paperclip[6], naše požadavky na nahrávání souborů ale nejsou téměř žádné, takže nepotřebujeme provádět větší srovnání.

2.2.5 Přihlašování, registrace a odhlašování uživatele

Pro tyto funkcionality se v Ruby on Rails nejčastěji používá knihovna Devise[7], která bude pro náš případ nejlepším řešením.

[3] `http://rubyonrails.org/`
[4] Atomic, Consistent, Isolated, Durable
[5] `https://github.com/carrierwaveuploader/carrierwave`
[6] `https://github.com/thoughtbot/paperclip`
[7] `https://github.com/plataformatec/devise`

2.2.6 Frontend technologie

Z analýzy nefunkčních požadavků 2.1.2 máme zadáno, že aplikace musí mít responzivní webdesign. Proto jsme se rozhodli použít dnes nejpoužívanější css, html a javascript framework Bootstrap[8].

Pro psaní html stránek použijeme templatovací jazyk Slim[9]. Další možností by bylo použití templatovacího jazyky HAML[10] nebo templatovací jazyk nepoužívat. Pro přehlednost kódu a rychlost psaní aplikace ho ale použijeme.

Pro některé frontendové akce budeme muset jistě použít javascript, ten budeme psát pomocí jazyku Coffeescript[11] a pokud bude nějaký problém řešitelný pomocí jQuery[12], použijeme i tuto knihovnu.

Na selectboxy s větším počtem možností použijeme javascriptovou knihovnu Chosen[13], která do selectboxu přidává textové vyhledávání nad polem hodnot pro výběr.

2.2.7 Vyhledávání

Vyhledávání bychom mohli realizovat s použitím výše uvedených technologií. To znamená vyhledávat přímo nad PostgreSQL databází. Jelikož chceme ale aby vyhledávání bylo co nejrychlejší a ze zadání máme dané, že bychom měli použít open-source systém pro distribuované a škálovatelné vyhledávání, použijeme nějaké jiné řešení.

2.2.7.1 Apache Lucene

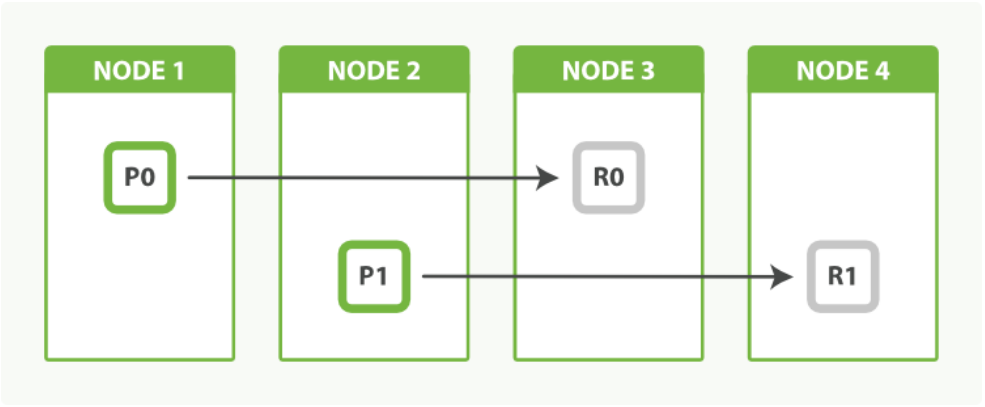
Lucene je výkonná open source knihovna pro fulltextové vyhledávání implementovaná v Javě. Mezi vývojáři je ceněná především pro svoji vyspělost a širokou komerční i komunitní podporu. Je to low-level knihovna, přímé použití není pro každého. Nabízí velmi detailní, nízkoúrovňové API, ale řadu problémů spojených s implementací rozsáhlejších systémů neřeší. [2]

Použití Lucene by pro náš účel bylo možné, ale existuje spousta rozšíření, která používají Lucene a řeší problémy, které bychom sami museli při návrhu a vývoji řešit. Například vlastní API[14].

2.2.7.2 Apache Solr

Apache Solr je open source rozšíření pro Lucene. Podporuje vyhledávání v textu, facetové vyhledávání, distribuované vyhledávání a mnoho dalších funkcí. Ob-

[8] `http://getbootstrap.com/`
[9] `http://slim-lang.com/`
[10] `http://haml.info/`
[11] `http://coffeescript.org/`
[12] `http://jquery.com/`
[13] `http://harvesthq.github.io/chosen/`
[14] Application Programming Interface



Obrázek 2.1: Ukázka distribuované architektury ElasticSearch, která obsahuje 4 nody a 2 shardy s replikou

sahuje REST[15] API s předáváním parametrů pomocí JSON[16]. Dále poskytuje distribuované vyhledávání, replikace indexu a je škálovatelný.

2.2.7.3 ElasticSearch

ElasticSearch je další řozšíření postavené na knihovně Lucene. Jeho popis by byl úplně stejný jako u Apache Solr, proto tato rozšíření porovnáme v další části.

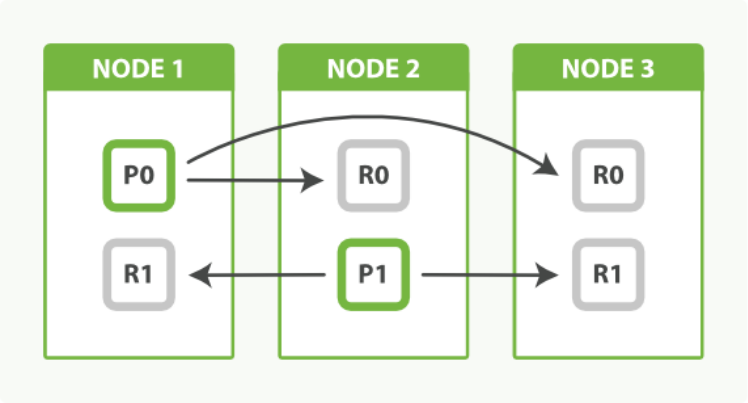
ElasticSearch jako distribuovaný systém Komponenty ElasticSearch jako distribuovaného systému se nazývají replica a shard.

Shard je hlavním úložištěm vyhledávacích indexů unikátní části dat. Systém může mít jeden nebo více shardů, které jsou na sobě vzájemně nezávislé.

Replica obsahuje přesnou kopii dat shardu. Pokud je část systému, kde se nachází shard, rozbitá a data nejsou dostupná, replica převezme činnost hlavního úložiště – shardu. Shard může mít libovolný počet replik. Replica je umístěna na jiné části systému než shard.

Na obrázku 2.1 je zobrazen systém, který obsahuje čtyři nody, dva shardy a každý shard má jednu repliku. Na obrázku 2.2 je zobrazen systém s třemi nody, dvěma shardy a dvěma replikami.

[15] Representational State Transfer
[16] JavaScript Object Notation - `http://www.json.org/`



Obrázek 2.2: Ukázka distribuované architektury ElasticSearch, která obsahuje 3 nody a 2 shardy s 2 replikami

2.2.7.4 Porovnání Apache Solr a ElasticSearch

Porovnání těchto dvou řešení se zabývá velké množství článků a plyne z nich mnoho různých názorů. Hlavním důvodem je, že jsou obě rozšíření velice podobná a podobně výkonná.

Chtěl bych uvést dvě citace z článku Ryanna Sonnka, který přešel ve svých aplikacích z používání Solr na ElasticSearch.

ElasticSearch byl postaven pro použití jako distribuovaný systém od začátku a nebylo mu to později „přilepeno“ jako tomu bylo u Solr. Je zřejmé, pokud porovnáme design a architekturu těchto dvou produktů i pokud zkoumáme kód [3].

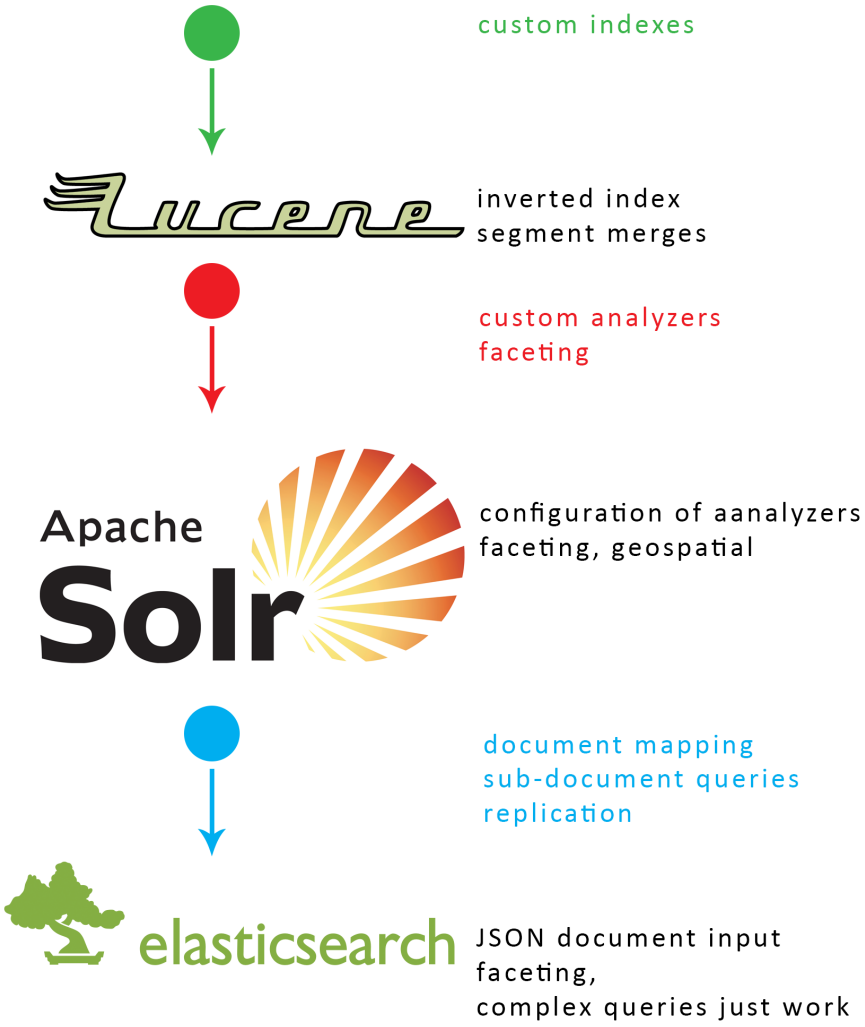
„Solr může být zbraň při výběru technologie pro standardních vyhledávací aplikaci, ale ElasticSearch ji posunuje na další úroveň s architekturou pro vytváření moderních realtime vyhledávacích aplikací. ElasticSearch je škálovatelný, rychlý a je sen ho integrovat. Adios Solr, bylo to fajn tě poznal“ [3].

Další odlišností kromě lepší podpory ElasticSearch pro škálovatelnost je vlastní DSL[17] jazyk, který ElasticSearch používá pro vytváření vyhledávacích dotazů. V Solr musíme na dotazování použít nativní Lucene textové dotazování. V neposlední řadě má ElasticSearch lepší podporu pro přístup k základní Lucene API.

Při porovnání rychlostí Solr a ElasticSearch je pro základní dotazy rychlejší Solr. Pokud ale provádíme více akcí najednou například aktualizace vyhledávacího indexu a vyhledávání je Solr řádově pomalejší [3].

Pro vyhledávání v naší aplikaci jsme zvolili ElasticSearch z důvodu lepší tvorby vyhledávacích dotazů pomocí DSL, jednoduššího nasazení, lepší správy a přehlednosti API. Dalším argumentem je rychlost, protože odhadujeme, že indexace textu diplomových prací bude zabírat více času. Solr by se v tomto

[17] domain-specific language



Obrázek 2.3: Vývoj popisovaných vyhledávacích technologií

ohledu hodil na použití, kdy indexujeme menší množství dat a požadujeme rychlejší vyhledávání.

Vývoj vyhledávacích řešení, které jsme popisovali ukazuje obrázek 2.3.

2.2.7.5 Integrace ElasticSearch a Ruby

Pro použití ElasticSearch v naší aplikaci budeme muset nějakým způsobem integrovat ElasticSearch do naší aplikace. Jedná se hlavně o připojení k vyhledávacímu clusteru a dále nástroj pro komunikaci s API ElasticSearch. Mohli bychom navrhnout vlastní řešení, ale použijeme raději nějaké dostupné řešení

popsané níže.

Elasticsearch-ruby Elasticsearch-ruby[18] je ruby knihovna která přidává:

- klienta pro připojení k ElasticSearch clusteru
- Ruby API pro komunikaci s ElastiSearch API
- další rozšíření a nástroje

Elasticsearch-rails Elasticsearch-rails[19] je knihovna využívající Elasticsearch-ruby, která tuto knihovnu rozšiřuje o použití v projektech postavených na frameworku Ruby on Rails o:

- integrace s ActiveModel a adapterý pro ActiveRecord
- vzor pro ukládání do persisteční vrstvy ElasticSearch pro ruby objekty
- vzor pro ukládání do persisteční vrstvy ElasticSearch pro ActiveRecord objekty
- třídu umožňující iterovat nad výsledky vyhledávání
- třídu která umožňuje vracet objekty z ActiveRecord z výsledků vyhledávání
- metody pro hledání, vytváření indexů pro vyhledávání a import dat
- tasky pro automatický import dat
- podporu pro stránkování
- příklady pro generování ukázkových aplikací

Chewy Chewy[20] je podobným rozšířením jako Elastic-rails, ale navíc přidává:

- třídu *Chewy::Index* pro tvorbu vlastních vyhledávacích indexů nezávislých na modelových třídách Ruby on Rails a ElasticSearch
- každý index má své vlastní volání pro aktualizaci při změně atributů v databázi, které je dostupné ze všech modelů
- nastavení pro to, aby byl index aktualizován pouze jednou i když byla tato žádost zavolána z více modelů najednou
- vlastní DSL pro tvorbu vyhledávacích dotazů

[18] `https://github.com/elasticsearch/elasticsearch-ruby`
[19] `https://github.com/elasticsearch/elasticsearch-rails`
[20] `https://github.com/toptal/chewy`

Obrázek 2.4: Architektura vyhledávání

Srovnání ElasticSearch-rails a Chewy Chewy je nejnovější[21] řešení pro integraci ElasticSearch do Ruby on Rails aplikace. Díky přidáním třídy *Chewy::Index*, metod pro aktualizaci indexu a svého vlastního DSL řeší nedostatky Elasticsearch-rails. Tyto nedostatky jdou sice vyřešit i s pomocí Elasticsearch-rails, ale vše se pak ukazuje na velké nepřehlednosti kódu. To může mít za následek jak horší možnosti při testování, tak omezení při rozšiřitelnosti.

Pro integraci s ElasticSearch jsme proto zvolili knihovnu Chewy. Celkovou architekturu vyhledávání zobrazuje obrázek 2.4.

2.2.8 Parsování dokumentu PDF

Ruby knihoven pro parsování PDF dokumentů je celá řada. My jsme vybrali Pdf-reader[22], který je jednoduchý na implementaci a má dostatečné funkce pro naše použití.

[21] První verze byla uvolněna v prosinci 2013
[22] `https://github.com/yob/pdf-reader`

2.2.9 Dolování dat z archívu ČVUT

Pro získání dat nemáme přístup do žádné databáze Archívu závěrečných prací, proto budeme muset data získat sami z webových stránek.

2.2.9.1 Manuální stahování

Jedním způsobem, jak data získat by bylo manuálně je stáhnout a nahrát do naší aplikace. Toto řešení jsme ze své podstaty zavrhli už na začátku návrhu.

2.2.9.2 Wget

Další možností je nástroj Wget[23], který umožňuje stahovat soubory pomocí protokolů HTTP, HTTPS a FTP. Nejdřív bychom museli získat seznam URL adres, kde se nachází jednotlivé práce, poté je pomocí Wget stáhnout, rozparsovat jejich obsah a uložit do naší aplikace.

2.2.9.3 Capybara

Capybara[24] je nástroj určený pro testování webových aplikací, který simuluje průchod aplikace a testuje, zda aplikace obsahuje definované prvky. Použitím Capybary bychom mohli dolovat data z archívu ČVUT.

2.2.9.4 Nokogiri

Nokogiri[25] je open-source nástroj určený pro parsování dokumentů HTML, XML a SAX v Ruby. Podporuje XPath a CSS3 selectory. Pro řešení našeho problému je tento nástroj nejvhodnější, proto ho použijeme.

2.2.9.5 SSL

Archív diplomových prací ČVUT používá protokol HTTPS. Proto potřebujeme řešení pro SSL[26], které HTTPS vyžaduje. Tento problém vyřešíme modulem OpenSSL[27], který obsahuje přímo Ruby.

2.2.10 Správa verzí a nahravání na server

Pro správu verzí použijeme systém Git[28] bežící na školní verzi repozitáře GitLab[29].

[23] `https://www.gnu.org/software/wget/`
[24] `https://github.com/jnicklas/capybara`
[25] `http://www.nokogiri.org/`
[26] Secure Sockets Layer
[27] `http://ruby-doc.org/stdlib-2.0/libdoc/openssl/rdoc/OpenSSL.html`
[28] `http://git-scm.com/`
[29] `https://about.gitlab.com/`

Nahrávání aplikace na server bude zajištěno pomocí nástroje Capistrano[30].

2.2.11 Měření času

Jelikož máme ze zadání dáno výpočítat náklady na vývoj a údržbu systému, pro měření času použijeme nástroj Toogle[31]

[30] `http://capistranorb.com/`
[31] `https://www.toggl.com/`

Kapitola 3

Implementace

V této kapitole popisuji analytický model aplikace, který koresponduje s výběrem technologií v předchozí kapitole. Dále vysvětlím základní implementaci návrhu aplikace a detailněji popíšu řešení zajímavějších problému doplněné o ukázky kódu řešení.

3.1 Analytický model aplikace

3.1.1 Diagram tříd

Obrázek 3.1 zobrazuje návrh diagramu tříd, které podrobněji rozepíšu v další části.

3.1.2 Popis jednotlivých tříd

3.1.2.1 User

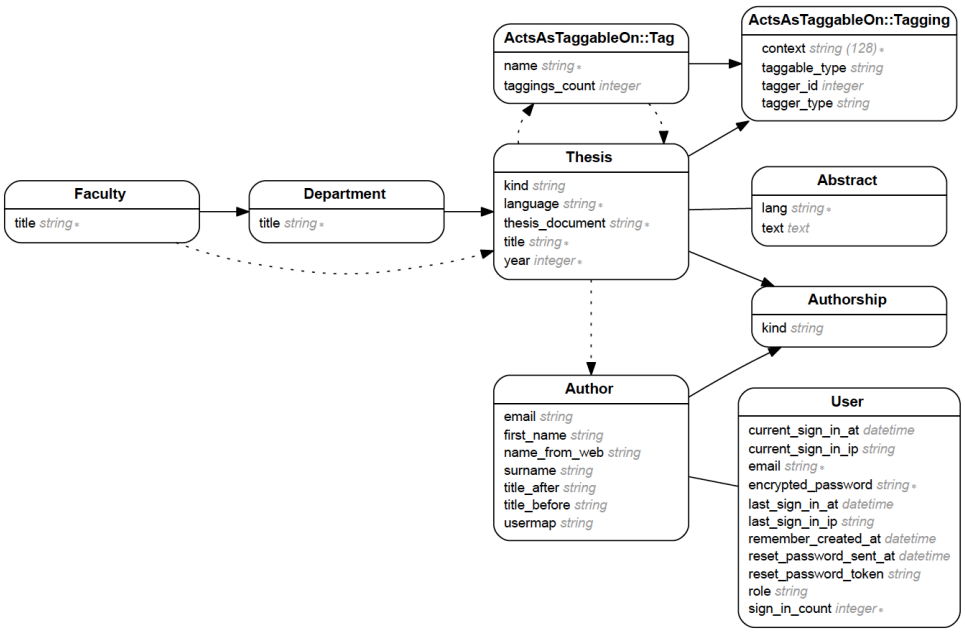
Třída *User* obsahuje atributy email a heslo potřebné pro přihlášení uživatele a údaj o roli. Uživatelské role budou v aplikaci dvě a to *admin* a *student*. Každý *User* může mít vazbu na jeden model *Author*.

3.1.2.2 Author

Třída *Author* představuje uživatele ČVUT, obsahuje atributy email, name_from_web a username pro identifikaci uživatele z webu ČVUT. *Author* může mít vazbu na třídu *Thesis* pomocí třídy *Authorship*.

3.1.2.3 Authorship

Třída *Authorship* přidává vazbu mezi třídy *Author* a *Thesis*. Navíc obsahuje atribut *kind*, který nabývá hodnot: *supervisor*, *opponent* nebo *author* a definuje tak o jakou vazbu se jedná.



Obrázek 3.1: Diagram tříd

3.1.2.4 Thesis

Tato třída definuje samotnou práci. Obsahuje atributy:

- *kind* – druh který může nabývat hodnot: *bachelor, master, seminar, other*
- *language* – jazyk který může nabývat hodnot: *cs, sk, en*
- *thesis_document* – odkaz na PDF
- *title* – název práce
- *year* – rok obhajoby práce

3.1.2.5 Abstract

Abstract obsahuje text a jazyk abstraktu. *Thesis* může obsahovat více abstraktů v různých jazycích.

3.1.2.6 ActsAsTaggableOn::Tag

ActsAsTaggableOn::Tag definuje tag neboli jmenný klíč, který obsahuje počet jeho použití a díky třídě *ActsAsTaggableOn::Tagging* má vazbu na více *Thesis*. *Thesis* může obsahovat více *ActsAsTaggableOn::Tag*.

3.1.2.7 Department a Faculty

Faculty představuje fakultu a má vazbu na více tříd *Department*. *Thesis* má vazbu na právě jednu *Department*, díky níž má vázbu i na *Faculty*. Třídy *Department* a *Faculty* definují název atributem *title*.

3.2 Standartní rozvržení aplikace v Ruby on Rails

Framework Ruby on Rails je postaven na návrhovém vzoru MVC. Díky tomu, že hlavní myšlenkou Ruby on Rails je upřednostnění konvence před konfigurací, máme jasně dané postupy, které při tvorbě aplikace musíme dodržovat.

3.2.1 Model

Model je v Ruby on Rails řešen pomocí knihovny ActiveRecord. Každá modelová třída dědí z třídy *ActiveRecord::Base*. V třídě modelu definujeme vazby na ostatní modely, validace vlastních atributů a metody spojené s ukládáním a načítáním dat z a do databáze.

3.2.2 Controller

Controller je třída, která dědí od třídy *ActionController::Base*. Tato třída zpracovává HTTP dotazy. Pokud je našemu webserveru zadán nějaký dotaz, nejdříve se pomocí souboru routes.rb[32] najde správný controller a jeho akce, na kterou je tento dotaz přesměrován. Controller má přístup k parametrům se kterými byl odeslán dotaz k session a cookies proměnným a samozřejmě k datům skrz modelové třídy. Jednotlivé metody controlleru představují akce, které jsou volány. Základní akce controlleru jsou: *index, show, new, edit, create, update a destroy*. Jedná se o předem definovaná akce, které by měli pokrýt funkcionalitu controlleru. Samozřejmě můžeme definovat akce vlastní.

3.2.3 View

View reprezentuje webové rozhraní aplikace. Controller předává data do view, která pomocí nich vykreslí odpověď na dotaz v požadovaném formátu.

3.2.4 Závěr

Ruby on Rails má velmi mnoho různých modelů a druhů tříd, které v naší aplikaci používáme, ale pro nastínění chodu aplikace v Ruby on Rails uvádíme tyto tři základní, aby byl správně pochopen vývoj aplikace.

[32]konfigurační soubor, který propojuje URL se správným controllerem a jeho akcí

3.3 Tvorba základní aplikace

Na začátku jsem vytvořil všechny potřebné modely z návrhu 3.1. Přidal jsem jim validace dat jako například pravidlo pro *Thesis*, která musí mít vyplněn jazyk, jenž musí být v rozsahu povolených jazyků, vazby na katedru, rok který musí být číslem, dokument a název. Tento příklad uvádím v následující ukázce kódu.

```
validates :language,        presence: true, inclusion: { in: LANGUAGES }
validates :department_id,   presence: true
validates :year,            presence: true, numericality: true
validates :thesis_document, presence: true
validates :title,           presence: :true
```

3.3.1 Administrátorská část aplikace

Poté jsem vytvořil základní controllery a jejich view pro běh aplikace. Začal jsem registrací, přihlašováním a odhlašováním uživatele, které řeší dříve zmíněný nástroj Devise. Dále jsem vytvořil administrátorskou část aplikace. Do níž mají přístup pouze uživatelé s rolí admin viz ukázka kódu z třídy *User*.

```
  def admin?
    role == 'admin'
  end
```

Všechny controllery obstarávající dotazy pro administrátorskou část pak dědí z třídy *Admin::AdminController* viz kód.

```
class Admin::AdminController < ActionController::Base
  layout 'admin'
  before_action :authenticate_user!
  before_action :authenticate_admin!

  private

  def authenticate_admin!
    redirect_to new_user_session_path unless current_user.admin?
  end
end
```

Before_action se zavolá před každou akcí. V tomto případě se nejdřív zavolá Devise metoda :authenticate_user!, která kontroluje zda je uživatel přihlášený, tudíž je dostupná metoda current_user vracející jeho instanci. Dále je zavolána metoda :authenticate_admin!, která kontroluje správnou roli uživatele. Pokud nemá uživatel roli admin je po dotazu na jakoukoliv akci třídy controlleru dědící z této třídy přesměrován.

Dál jsem v administrátorské části přidal formuláře pro editaci atributů, přidávání a mazání dat tříd, které může administrátor z definice návrhu spravovat.

3.3.2 Uživatelská část aplikace

Poté jsem implementoval uživatelskou část aplikace do níž má přístup i administrátor.

3.3.2.1 Oddělení práv přihlášeného a nepřihlášeného uživatele

Nejdříve jsem oddělil v controlleru *ThesisController* akce, které jsou dostupné i nepřihlášenému uživateli:

```
before_action :authenticate_user!, except: [:show, :index]
```

Toto řešení je podobné jako v předchozí ukázce autorizace administrátorských akcí.

3.3.2.2 Vazba mezi autory a prací

Dále jsem implementoval přidání samotné práce. Zde jsem musel vyřešit problém, jak přidat různé vazby na autora, oponenta a vedoucího práce, které zajišťuje třída *Authorship*. Tuto třídu jsem rozšířil o filtry:

```
scope :_author,    -> { where(kind: 'author').last }
scope :supervisor, -> { where(kind: 'supervisor').last }
scope :opponent,   -> { where(kind: 'opponent').last }
```

Pro vysvětlení například filtr _author vybere z množiny authorship poslední instanci, která má druh vazby typy author. Pokud budeme editovat vazbu práce a autora, budeme vytvářet nové instance třídy *Authorship*, což chceme, abychom měli záznam o změnách na této vazbě a proto také vybíráme poslední záznam.

Na modelovou třídu *Author* jsem pak přidal metody, které vrací a nastavují instance, jako by se jednalo o přímou vazbu. Můžeme si to ukázat například na vazbách autora.

```
def author
  authorships._author.try(:author)
end

def author_id
  author.try(:id)
end
```

```
  def author=(author_id)
    setter_by_role('author', author_id)
  end

  alias_method :author_id=, :author=

   private

  def setter_by_role(role, author_id)
    authorships.initialize_by(author_id: author_id, kind: role)
  end
```

Metoda author vrací instanci autora, autor_id jeho id z databáze. Metody author= a author_id= jsou totožné a volají metodu setter_by_role, kterou používají i metody spravující vedoucího a oponenta. Metoda setter_by_role inicializuje novou instanci *Authorship* s příslušnými parametry.

3.3.2.3 Nahrávání dokumentu

Řešení nahrávání dokumentu je z návrhu definováno nástrojem CarieWave. Pro jeho implementaci jsme nejdříve museli konfigurovat třídu *ThesisDocumentUploader*

```
class ThesisDocumentUploader < CarrierWave::Uploader::Base

  def store_dir
    "uploads/#{model.class.to_s.underscore}/#{mounted_as}/#{model.id}"
  end
end
```

A poté přidat její uploader do třídy *Thesis*

```
mount_uploader :thesis_document, ThesisDocumentUploader
```

Nyní už je vše zařízeno a knihovna CarrierWave obstará uložení souboru do definované složky. Aby jsme po každém nahrání nové verze aplikace nepřišli o nahrané soubory, bylo potřeba složku public/uploads přidat jako složku, která je na serveru pouze jednou a předává se odkazem. Pro nástroj Capistrano je nastavení v souboru *config/deploy.rb* následující:

```
set :linked_dirs, %w{bin log public/system public/uploads }
```

Indexování a přípravu dokumentu na vyhledávání budu řešit v další kapitole.

3.3.3 Editace vlastních prací

Editovat a automaticky otagovat práci může z návrhu pouze administrátor a uživatel, který je na práci uveden jako autor. Tento problém řeší metoda třídy *Thesis* can_updated_by? vracející boolean.

```
def can_updated_by?(user)
  author.try(:user) == user || user.admin?
end
```

Tato metoda je pak použita v *ThesesControlleru* následovně

```
  before_action :authorize_user!, only: [:edit, :update, :add_tags]

private

  def authorize_user!
    redirect_to theses_path unless @thesis.can_updated_by?(current_user)
  end
```

3.4 Automatické tagování

3.4.1 Přidání tagů do práce

Ze zadání bylo dané extrahovat data z dokumentu a ta použít. Já jsem si v návrhu pro toto zadání vybral extrahování tagů z dokumentu práce. V analýze jsem uvedl, že se práce bude tagovat na základě předem definovaných tagů. Tyto tagy, jak bylo popsáno výše, spravuje administrátor. Pro funkcionalitu tagování jsme využili knihovnu ActsAsTaggableOn[33], která rozšiřuje třídu *Thesis* o možnost tagování jedním řádkem kódu.

```
acts_as_taggable
```

3.4.2 Tf-idf

Dále jsme potřebovali nějaký způsob, jak hodnotit relevanci slov v dokumentu na základě četnosti jejich použití a jejich vyjímečnosti proti použitému jazyky, abychom je mohli označit za relevantní pro tag práce.

Pro řešení tohoto problému jsme vybrali statistickou metodu Tf-idf. Tato metodika se skládá ze dvou složek tf[34] a idf[35].

[33] https://github.com/mbleigh/acts-as-taggable-on
[34] term frequency
[35] inversed document frequency

Tf Složka Tf počítá četnost výskytu slova v dokumentu. Vysvětlím na následující rovnici pro její výpočet.

$$tf_{(i,j)} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{3.1}$$

$n_{i,j}$ je počet výskytů slova t_i v dokumentu d_j, který vydělíme součtem výskytu všech slov v dokumentu.

Idf Složka Idf reprezentuje důležitost slova v dokumentu.

$$idf_{(i)} = \log \frac{|D|}{|j : t_i \in d_j|} \tag{3.2}$$

|D| představuje počet dokumentů ve kterých hledáme. Jmenovatel představuje počet dokumentů, které obsahují slovo i. Z této rovnice tak dostáváme inverzní frekvenci výskytu slova, která hodnotí jeho důležitost v rámci dokumentu [4].

Hodnocení slova Vynásobením těchto dvou složek, tak dostáváme hodnocení významu slova, které jsme hledali.

3.4.3 Implementace Tf-idf

Pro implementaci Tf-idf v naší práci jsme použili knihovnu Tf-idf-similarity[36]. Dokument pro nás představuje jedna strana práce. Vyhledávání pak provádíme nad polem námi definovaných tagů, výsledkem je hodnocení výskytu tagů v práci pro každý samostatný tag od 0 to 10. Pomocí konstanty MIN_RATING, pak tagy s větším tagem přidáváme do tagů práce.

Tento popis doplňuji ukázkou důležité části výsledného kódu:

```
def tags
  thesis_tags = @thesis.tags.map(&:name)
  (ActsAsTaggableOn::Tag.all.map &:name).
    reject { |tag| thesis_tags.include? tag }
end

def sorted_tags_array
  tfidf_by_tags = {}

  tags.each do |tag|
    tfidf_by_tags[tag] = tfidf_model.tfidf(thesis_text_document, tag)
  end
```

[36] https://github.com/opennorth/tf-idf-similarity

```
  tfidf_by_tags.sort_by{|_,tfidf| -tfidf}
end


def rated_tags
  sorted_tags_array.reject { |tag| tag[1] < MIN_RATING }.
    map &:first
end
```

3.5 Vyhledávání

Základní funkcionalitou mé práce je vyhledávání. Vyhledávání pomocí ElasticSearch bych rozdělil na tři fáze.

3.5.1 Mapování

Mapování neboli tvorba vyhledávacího indexu je základní stavební kámen pro vyhledávání. Na index si namapujeme atributy potřebné k vyhledávání objektu. Poté je index uložen do vyhledávací databáze.

3.5.1.1 Analyzer

Analyzer je kombinace jednoho či více filtrů. Každý filtr provádí nějakou úpravu slov v textu. V našem případě používáme czech analyzer. Tento analyzer se skládá z filtrů: `lowercase`, `czech_stop`, `czech_keywords`, `czech_stemmer`, které se volají na analyzovaný text jednotlivě za sebou. Filtr `lowercase` převede všechna velká písmena na malá. Filtr `czech_stop` vyloučí z indexace slova, která jsou na seznamu `stop_words`. Tento seznam označuje často používaná slova, která nejsou nositelem informace jako jsou například spojky, předložky nebo zájmena. Filtr `czech_stemmer` provádí stematizaci, což je proces nalezení základu slova. Výsledkem stematizace nemusí být validní slovo, ale slovo vhodné k vyhledávání.

3.5.1.2 Text dokumentu

Obsah dokumentu nám vrací metoda *Thesis* `parsed_document`. Tato metoda vytváří instanci třídy *PdfParser*, která pomocí nástroje PdfParser vrací text práce.

3.5.1.3 Implementace

V našem případě vyhledáváme v aplikaci pouze nad modelem *Thesis*. Tvorbu a definici vyhledávacích atributů popíšu na ukázce kódu:

```
class ThesesIndex < Chewy::Index

  define_type Thesis.includes(:department, :tags, :abstract) do
    field :language, :kind
    field :title,           analyzer: 'czech'
    field :parsed_document, analyzer: 'czech'
    field :abstract,        value: ->{abstract.text if abstract.present?},
                            analyzer: 'czech'
    field :year,            type: 'integer'
    field :department_id,   type: 'integer', value: ->{ department.id }
    field :faculty_id,      type: 'integer', value: ->{ department.faculty_id }
    field :author_id,       type: 'integer', value: ->{ author.id }
    field :opponent_id,     type: 'integer',
                            value: ->{ opponent.id if opponent.present? }
    field :supervisor_id,   type: 'integer',
                            value: ->{ supervisor.id if supervisor.present? }
    field :tags,            value: ->{ tags.map &:name }
  end
end
```

ThesesIndex definuje vyhledávací atributy:

- language, kind – jazyk a druh práce
- title, parsed_document, abstract - název, text a abstrakt práce na které se použije czech_analyzer.
- year – rok práce, defaultně Elasticsearch ukládá atributy jako string, proto musíme označit atributy, které jsou jiného typu.
- department_id, faculty_id, author_id, opponent_id, supervisor_id – id jednotlivých atributů, které jsou použity pouze jako filtry, tudíž nejsou zahrnuty do fulltextového vyhledávání a stačí nám pouze jejich id
- tags – pole tagů z kterých mapujeme do indexu pouze jejich jméno

3.5.2 Aktualizace indexů

3.5.2.1 Implementace

Nyní máme data pro vyhledávání namapovaná na vyhledávacím indexu, ale pokud se reálná hodnota změní, potřebujeme tento index aktualizovat. K řešení tohoto problému slouží v Chewy metoda update_index, které předáváme který index a kterou metodu mapuje.

Pro *Thesis* je tento zápis takový:

```
update_index 'theses#thesis', :self
```

3.5.2.2 Atomicita

V souvislosti aktualizace indexů musíme vyřešit ještě jeden problém s hromadnou aktualizací indexů. Protože máme nastavenu aktualizaci na všech mapovaných objektech do databáze, tak by se v našem případě například po změně názvu práce, jména tagu a textu práce najednou aktualizoval index třikrát. Toto chování řeší Chewy.atomic. Tato metoda zakazuje after_save callback, udržuje id změněných atributů a nakonec po volání Chewy.atomic, provede jednu žádost o aktualizaci. V mé aplikaci se tato metoda volá před každou akcí controlleru.

3.5.3 Rozhraní pro vyhledávání

Nyní můžeme implementovat rozhraní pro vyhledávání. Pro implementaci rozhraní jsem použil modul ActiveData[37], který vytvoří frontend podobný ActiveRecord nad definovanými atributy. Jednotlivé atributy korespondují s názvy hledaných atributů. Pro ty pak vytvářím jednotlivé filtry nebo je přidávám do fulltextového vyhledávání dle návrhu.

Ukázka výsledného kódu:

```
attribute :full_text,     type: String
attribute :kind,          type: String
attribute :year,          type: Integer
attribute :department_id, type: Integer

def kind_filter
  index.filter(term: { kind: kind }) if kind?
end

def year_filter
  index.filter(term: { year: year }) if year?
end

def full_text_search
  if full_text?
    index.query(query_string: { fields:
                                   [:title, :parsed_document, :tags],
                                query: full_text,
                                default_operator: 'and'  })
  end
end
```

Spojení vyhledávacích dotazů a filtrů a volání na předem popsaném indexu:

[37] https://github.com/pyromaniac/active_data

```
def index
  ThesesIndex
end

def search
  [full_text_search, year_filter, department_id_filter, author_id_filter,
   opponent_id_filter, supervisor_id_filter, kind_filter, tags_filter].
    compact.reduce(:merge)
end
```

Kapitola 4

Vyhodnocení

V této kapitole vyhodnotím výsledek implementace vlastního řešení aplikace pro vyhledávání v databázi závěrečných pracích. Nejdříve porovnám toto řešení se současným řešením, dále uvedu ukázku aplikace a uvedu možnosti rozšíření aplikace. Nakonec uvedu přínos aplikace.

4.1 Porovnání se současným řešením

4.1.1 Porovnání na základě požadavků na vyhledávání v závěrečných pracích

Současné řešení Archívu diplomových prací ČVUT splňuje tři ze sedmi mnou popsaných požadavků na vyhledávání v závěrečných pracích 1.2 :

- fulltext vyhledávání nad obsahem dokumentu
- rychlost

Implementované řešení splňuje všechny.

4.1.2 Porovnání na základě vyhledávacích scénářů

Pro další porovnání stanovým tři různé informace, které požadujeme systémem vyhledat a popíšu scénáře a výsledky vyhledávání při použití jednotlivých systémů.

Seznam hledaných informací:

1. práce související s jazykem java
2. práce vydané na katedře softwarového inženýrství v roce 2014
3. diplomové práce vydané na katedře softwarového inženýrství v roce 2014

4.1.2.1 Řešení současným systémem

1. práce související s jazykem java Vyhledávání provedu na základě tohoto scénáře:

1. Do vyhledávacího pole napíšu‘java’.

Výsledky vyhledávání odpovídají zadání. Jinou možnost vyhledávání nemám k dispozici. Zadaný úkol se podařilo na tomto systému vyřešit v jednom kroku.

2. práce vydané na katedře softwarového inženýrství v roce 2014 Vyhledávání provedu na základě tohoto scénáře:

1. Na hlavní straně zvolím ‘Nahlížení do archívu’ a jsem přesměrován na výběr fakulty.
2. Musím vědět, na jaké fakultě se nachází katedra softwarového inženýrství, zvolím ‘Fakulta informačních technologií’ a jsem přesměrován na výběr katedry.
3. Zvolím ‘Katedra softwarového inženýrství (18102)’ a jsem přesměrován na výběr roku[38]
4. Zvolím ‘2014’ a jsem přesměrován na seznam prací.

Výsledky vyhledávání odpovídají zadání. Jinou možnost vyhledávání nemám k dispozici. Zadaný úkol se podařilo na tomto systému vyřešit ve čtyřech krocích.

3. diplomové práce vydané na katedře softwarového inženýrství v roce 2014 Nemám žádnou možnost provést výběr pouze diplomových prací. Jedinou možností je postupovat jako v předchozím případě a na seznamu vyhledávání pozorovat zda se jedná o diplomovou práci.

4.1.2.2 Řešení vlastním implementovaným systémem

1. práce související s jazykem java Vyhledávání mohu provést na základě dvou scénářů:

1. scénář:
    a) Do vyhledávacího pole napíšu ‘java’.
2. scénář

[38] Pokud by zadání úkolu nespecifikovalo rok, ale pouze katedru, nemělo by pro toto zadání hledání na tomto systému řešení.

a) Zvolím vyhledávací filtr dle tagu java.

Výsledky vyhledávání odpovídají zadání. K dispozici mám dvě možnosti vyhledávání. Zadaný úkol se podařilo na tomto systému vyřešit v jednom kroku.

2. práce vydané na katedře softwarového inženýrství v roce 2014 Vyhledávání provedu na základě tohoto scénáře:

1. Přidám filtr pro katedru softwarového inženýrství výběrem selectboxu
2. Přidám filtr pro rok 2014 výběrem v selectboxu

Výsledky vyhledávání odpovídají zadání. K dispozici mám dvě možnosti vyhledávání[39]. Zadaný úkol se podařilo na tomto systému vyřešit ve dvou krocích.

3. diplomové práce vydané na katedře softwarového inženýrství v roce 2014 Vyhledávání provedu na základě tohoto scénáře:

1. Přidám filtr pro druh diplomové práce
2. Přidám filtr pro katedru softwarového inženýrství výběrem selectboxu
3. Přidám filtr pro rok 2014 výběrem v selectboxu

Výsledky vyhledávání odpovídají zadání. K dispozici mám šest možností vyhledávání[40]. Zadaný úkol se podařilo na tomto systému vyřešit ve třech krocích kroku.

4.1.2.3 Závěr

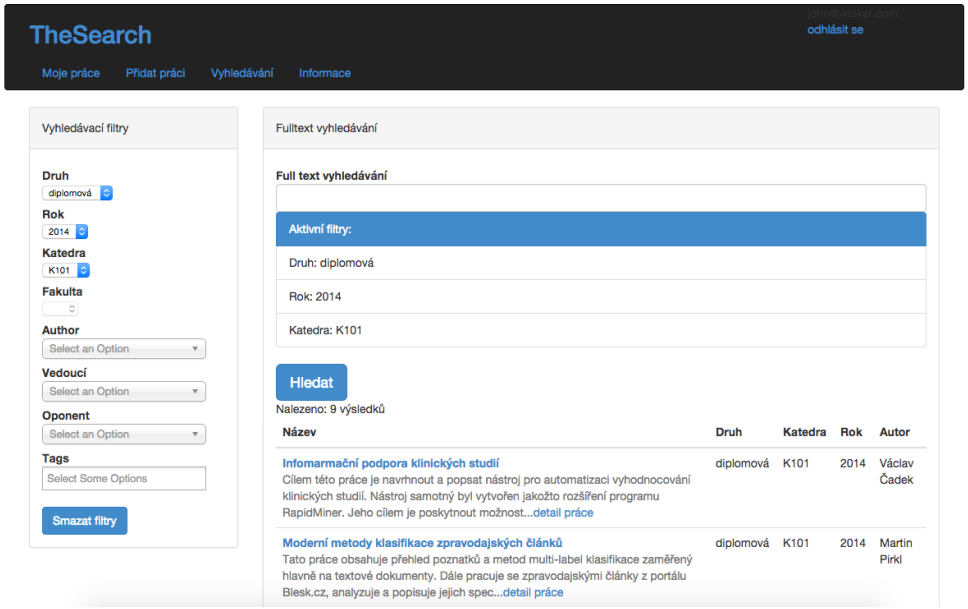
Pro jednoduchá vyhledávání v textu je scénář hledání podobný na obou systémech. Pro složitější nároky na hledání buď nejde zadání na současném systému vyřešit nebo obsahuje znatelně více kroků a méně možností pro zadání podmínek pro hledání.

4.1.3 Porovnání na základě výsledků hledání

4.1.3.1 Současný systém

Současný systém zobrazuje pokaždé jiné výsledky hledání. Při průchodu archívu je to seznam prací a při fulltextovém hledání je to seznam výsledků s odkazem na dokument práce nebo detail práce.

[39] Filtry mohu volit i v opačném pořadí.
[40] Filtry mohu volit i v jakémkoliv pořadí.



Obrázek 4.1: Náhled na stranu vyhledávání v testovací verzi aplikace

4.1.3.2 Vlastní implementovaný systém

Implementovaný systém zobrazuje vždy seznam použitých filtrů a hledaný text. Výsledky hledání jsou vždy stejné a to název, druh, katedra, rok, autor a úryvek abstraktu. Zobrazen je vždy i počet nalezených výsledků.

4.1.4 Závěr

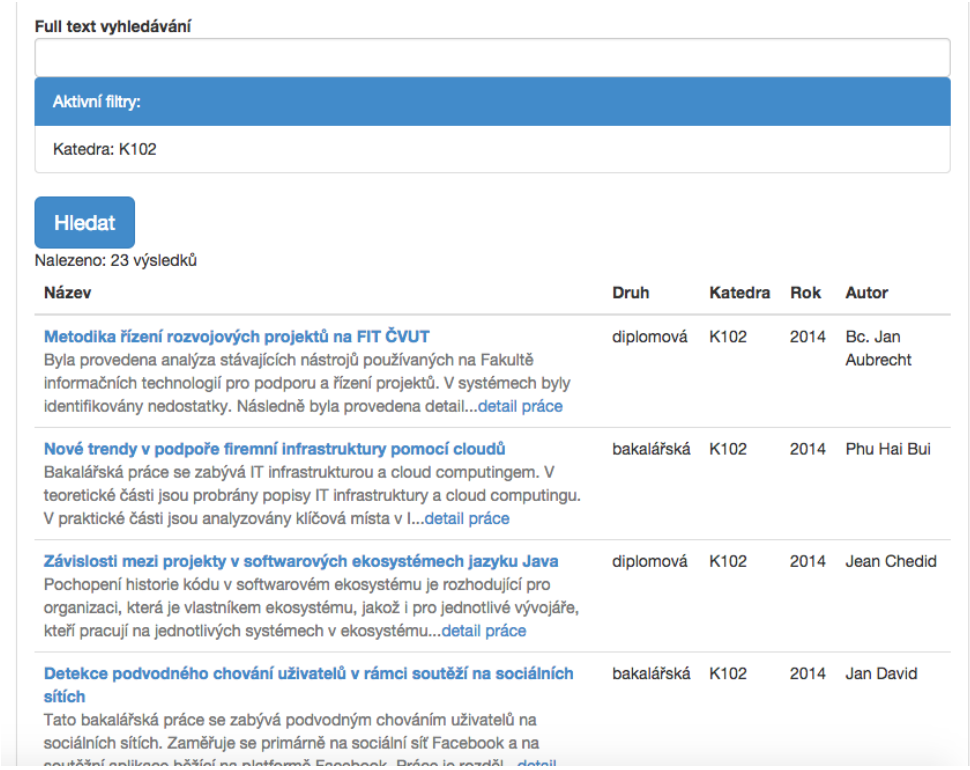
Implementovaný systém oproti současnému řešení splňuje požadavky na moderní vyhledávací systém. Vyhledávání podle složitějších parametrů je možné a jednodušší. Výsledky vyhledávání mají vždy stejný formát.

4.2 Ukázka aplikace

V této části ukazuji pár vybraných částí aplikace pro představu. Aktuálně je testovací verze aplikace umístěna na adrese `http://storm2.fit.cvut.cz:9000/`, kde je volně přístupná k nahlédnutí.

4.2.1 Výsledky vyhledávání

Na stránce vyhledávání 4.1 máme k dispozici vyhledávací filtry v levé části, pod nimi se nachází tlačítko na vymazání všech filtrů. V pravé části stránky se nachází vstup pro fulltextové vyhledávací pole. Pod ním je zobrazen seznam



Obrázek 4.2: Náhled zobrazení detailů vyhledávání v testovací verzi aplikace

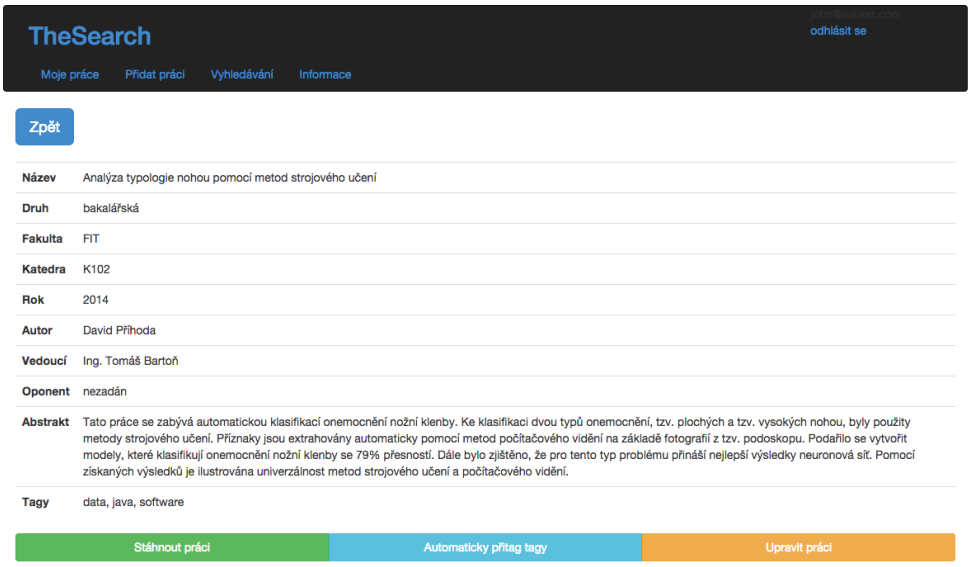
aktivních vyhledávacích filtrů a počet nalezených výsledků. Následuje výsledek vyhledávání4.2, což je seznam prací s názvem, druhem, katedrou, rokem, autorem a úryvkem abstraktu práce a odkazem na zobrazení detailu.

4.2.2 Detail práce

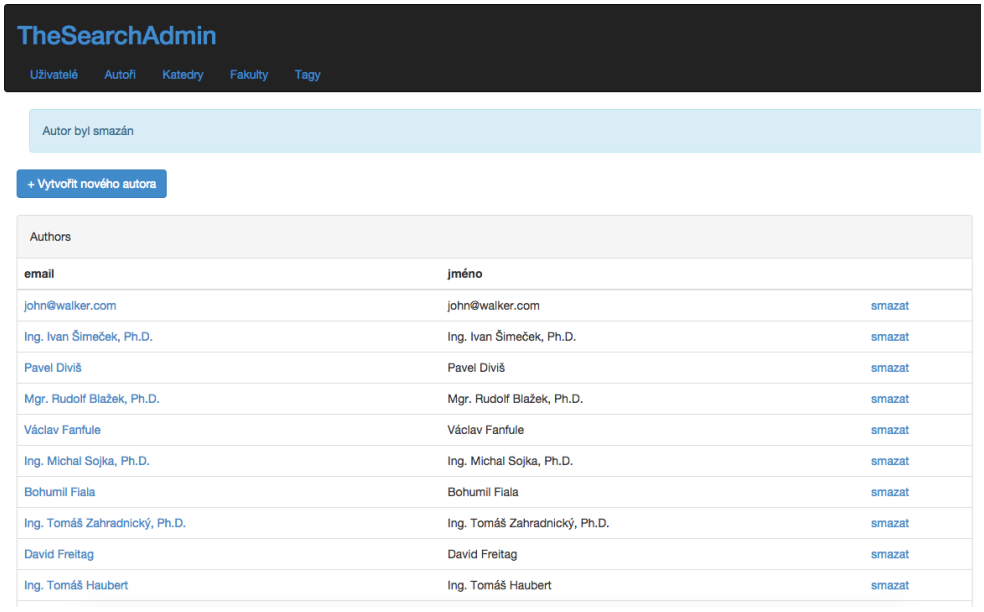
Na stránce detailu práce 4.3 se nachází tlačítko zpět pro návrat na přechozí stránku. Dále následuje výpis všech atributů práce. V dolní části stránky, se nachází ovládání pro editaci práce a tlačítko pro stáhnutí práce. Stáhnutí práce je dostupné vždy, ostatní tlačítka jsou zobrazena pouze, pokud je přihlášený uživatel uveden jako autor práce nebo má roli administrátor.

4.2.3 Administrátorská část

V administrátorské části můžeme přidávat, editovat a mazat předem popsané atributy. Náhled ukazuje stav aplikace po přidání autora na kterém vidíme informaci o úspěšném vytvoření autora, tlačítko pro možnost vytvoření dalšího autora a seznam autorů s odkazem na jejich editaci nebo smazání.



Obrázek 4.3: Náhled zobrazení detailu práce



Obrázek 4.4: Náhled administrátorské části aplikace ve stavu po vytvoření autora

4.3 Možnosti rozšíření

Při vývoji aplikace jsem určil další rozšíření aplikace, která by bylo možné implementovat.

4.3.1 Přihlášení do aplikace pomocí hesla ČVUT

Nyní se do aplikace mohou přihlásit uživatelé, kteří se do ní registrovali nebo jim administrátor vytvořil účet. Bylo by dobré vytvořit autentifikaci uživatele, podobnou jako do ostatních systémů ČVUT. Pro toto řešení bych doporučoval LDAP.

4.3.2 Další možnosti vyhledávání

Na základě uživatelského průzkumu a následného AB testování zkusit na reálných uživatelích o jaké další možnosti vyhledávání, především filtry, by měli zájem. Ty pak implementovat do aplikace. Nabízelo by se například:

- filtrování prací za časovou periodu
- filtrování prací na kterých se autor podílel v jakékoliv roli
- přidání dalších atributů do fulltextového vyhledávání

4.3.3 Další atributy práce

Při návrhu aplikace jsem vycházel z atributů, které jsou nyní dostupné na Archívu diplomových prací ČVUT, abych je mohl importovat. Při přidávání nových prací by bylo možné přidat další atributy práce. Jako například:

- hodnocení vedoucího a oponenta
- studijní plán autora
- výsledek obhajoby
- seznam citací
- seznam kapitol

4.3.4 Možnost ohodnotit práci

Přihlášení uživatelé by měli možnost ohodnotit, jak jsem jim práce líbila. Tento atribut by poté mohl být zařazen do vyhledávacích filtrů.

4.3.5 Možnost komentovat práci

Přihlášení uživatelé by měli možnost přidat komentář pod detail práce. NA základě počtu i obsahu komentářů by se dal vytvořit vyhledávací atribut.

Tabulka 4.1: Konzultace a správa serveru

název činnosti	**čas**
konzultace návrhu aplikace	12
nastavení serverů	6
konzultace chyb aplikace	10
celkem	**38**

4.3.6 Možnost sdílet práci

Přidat možnosti pro sdílení práce. Práce by se dala sdílet odesláním emailu nebo na sociálních sítích.

4.3.7 Profil uživatele

Přidat profily uživatelů na kterých by byly informace o uživateli, kontakt a seznam závěrečných prací na kterých se podílel.

4.4 Náklady a zhodnocení

4.4.1 Náklady na vývoj aplikace

4.4.1.1 Čas vývoje aplikace

Pro změření času na vývoj aplikace, jsem použil v analýze definovaný nástroj Toggl. Tímto nástrojem jsem měřil veškerou aktivitu na projektu, abychom mohli následně určit náklady na vývoj aplikace. Tyto náklady jsem rozdělil do dvou kategorií. První je konzultace a správa serveru4.1, která probíhala s vedoucím aplikace. Druhá je čas vývoje aplikace4.3, který jsem prováděl sám. Čas je uveden v hodinách

4.4.1.2 Výpočet nákladů

Svojí práci jakožto junior programátora jsem ohodnotil na 300 Kč za hodinu. Práci vedoucího – senior programátora jsem ohodnotil na 500 Kč za hodinu. Výsledný výpočet ceny ukazuje tabulka.

4.4.1.3 Závěr

Časové náklady na vývoj aplikace pro jednoho člověka jsou 367 hodin, což by při 6 pracovních hodinách denně znamenalo okolo 61 dnů vývoje tj. 12 týdnů. Finanční náklady na vývoj jsou **116 200 Kč**. Tyto náklady nezahrnují režijní náklady o které by byl vývoj aplikace reálně dražší.

Tabulka 4.2: Vývoj aplikace

název činnosti	**čas**
základní nastavení aplikace a technologií	15
tvorba modelů	35
registrace, přihlášení a odhlášení uživatele	5
administrátorská část	36
uživatelská část	48
nastylování aplikace	18
implementace vyhledávání	37
import prací z Archívu ČVUT	29
automatické tagování	18
oprava chyb	27
testování	23
celkem	**291**

Tabulka 4.3: Vývoj aplikace

aktér	**typ práce**	**hodinová sazba**	**počet hodin**	**cena**
vedoucí	konzultace a správa serverů	500 Kč	38	17 500 Kč
autor	konzultace a správa serverů	300 Kč	38	11 400 Kč
autor	vývoj aplikace	300 Kč	291	87 300 Kč
celkem			367	116 200 Kč

4.4.2 Odhad nákladů na údržbu a provoz aplikace

4.4.2.1 Náklady na provoz serveru

Pro odhad nákladů na provoz aplikace jsem zvolil řešení od firmy Wedos[41]. Pro naší aplikaci bychom potřebovali dedikovaný server s alespoň dvěma clustery a 16 GB RAM. Dále potřebujeme diskovou kapacitu alespoň 500 GB. Takové řešení u Wedosu stojí **3350 Kč měsíčně**.

4.4.2.2 Náklady na správu chyb

Chyby bychom zaznámenávali pomocí informačního emailu přímo od uživatelů, dále bychom automaticky zpracovávali chyby ze serveru nástrojem Airbrake[42], který stojí 39 USD měsíčně, což je **910 Kč**.

4.4.2.3 Náklady na údržbu aplikace a řešení chyb

Na údržbu aplikace a řešení chyb odhaduji 3 man-day měsíčně, hodinovou sazbu jsem stanovil na 500 Kč. Při délce man-day 6 hodin jsou měsíční náklady

[41] `http://hosting.wedos.com/`
[42] `https://airbrake.io/`

na údržbu aplikace a řešení chyb **9 000 Kč**.

4.4.2.4 Závěr

Výsledný odhad měsíčních nákladů na údržbu a provoz aplikace je **13 260 Kč**.

4.4.3 Zhodnocení

4.4.3.1 Finanční zhodnocení

Finanční zhodnocení při využití aplikace pouze ČVUT se mi nepodařilo nalézt. Zhodnotit aplikaci by se podařilo, pokud bychom poskytli řešení jiné organizaci. Náklady na přizpůsobení aplikace by byly závislé na požadavcích změn. Náklady na údržbu a provoz aplikace se odvíjí dle zatížení a požadavků, ale pro jakoukoliv univerzitu v České republice by měly být stejné.

4.4.3.2 Nefinanční zhodnocení

Aplikace má tato nefinanční benefity:

- Lepší prezentace ČVUT
- Snadnější přístup k informacím závěrečných prací
- Větší možnosti při výběru témata a vedoucího závěrečné práce
- Zvýšení povědomí o tvorbě a rozsahu práce ČVUT

4.4.4 Závěr

Aplikace nemá přímé finanční zhodnocení, ale popsal jsem možnost, jak by ho mohla dosáhnout. Nefinančních zhodnocení jsem našel více jak pro lepší obraz ČVUT na veřejnost, tak pro využití studenty a zaměstnanci ČVUT.

Závěr

Hlavní cíl práce, kterým bylo vyvinout rychlý a použitelný archív a vyhledávací systém v závěrečných pracích byl splněn.

Podařilo se mi popsat současná řešení na ČVUT i ostatních vybraných univerzitách. Dle těchto poznatků jsem navrhl vlastní řešení. Implementoval jsem webovou aplikaci pro fulltextové vyhledávaní za použití populárního frameworku Ruby on Rails a systému ElasticSearch. Výsledná aplikace odpovídá požadavkům na moderní webovou aplikaci a vzhledem k zavedeným konvencím, je jednodušší pro další programátory navázat na moji práci. Implementovaný systém zobrazuje ve výsledcích vyhledáváni smysluplné názvy dokumentů a relevantní text. Navíc je možné jednoduše vyhledat všechny práce například s konkrétním vedoucím nebo kombinovat nastavení filtrů, což současný systém neumožňuje.

Do systému jsem importoval data ze současného systému a na základě automatického i manuálního testování nastavil jeho nejlepší použití pro práce vzniklé na ČVUT.

Systém byl nasazen na školní cluster Storm. Při testování běžely 4 instance ElasticSearch (4 shards, 2 replicas).

Při návrhu a implementaci se mi podařilo popsat nejvhodnější nalezená řešení a technologie pro vývoj aplikace v Ruby on Rails, která se zabývá vyhledáváním dat a vyhledáváním v textu.

Dále jsem vyhodnotil jaké finanční i nefinanční benefity toto řešení přináší a přidal další možnosti jejího rozšíření.

Na půdě ČVUT vzniká spousta zajímavých prací a doufám, že moje řešení by mohlo přispět k jejich větší dostupnosti a tím ke znovupoužitelnosti myšlenek a rozšíření prací z ČVUT.

Literatura

[1] Patterson, A.: *ActiveRecord*. Dostupné z: `http://api.rubyonrails.org/classes/ActiveRecord/Migration.html`

[2] Vlček, L.: Elasticsearch: Vyhledáváme hezky česky. *zdrojak.cz*, 2013. Dostupné z: `http://www.zdrojak.cz/clanky/elasticsearch-vyhledavame-cesky/`

[3] Sonnek, R.: Realtime Search: Solr vs Elasticsearch. *socialcast*, 2011. Dostupné z: `http://blog.socialcast.com/realtime-search-solr-vs-elasticsearch/`

[4] Manning, C. D.; Raghavan, P.; Schütze, H.: *Introduction to information retrieval*, ročník 1. Cambridge university press Cambridge, 2008.

Příloha A

Seznam použitých zkratek

ACID Atomicity, Consistency, Isolation, Durability

API Application programming interface

CSS Cascading Style Sheets

CUNI Charles University

FTP File Transfer Protocol

HTPPS Hypertext Transfer Protocol Secure

HTTP Hypertext Transfer Protocol

IDF Inversed Document Frequency

LDAP Lightweight Directory Access Protocol

MUNI Masarykova univerzita

MVC Model View Controller

PDF Portable Document Format

REST Representational State Transfer

RoR Ruby on Rails

SAX Simple API for XML

SSH Secure Shell

SSL Secure Sockets Layer

TF Term Frequency

TUL Technická univerzita Liberec

URL Uniform Resource Locator

USD United States Dollar

XML Extensible Markup Language

ČVUT České vysoké učení technické

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF