Sem vložte zadání Vaší práce.

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

Bakalářská práce

Portál pro podporu tvorby článků na wikipedia.org

Václav Makeš

Vedoucí práce: Ing. Karel Klouda, Ph.D.

11. ledna 2016

Poděkování

Chtěl bych poděkovat vedoucímu bakalářské práce Ing. Karlovi Kloudovi, Ph.D. za rady a připomínky k práci. Dále bych rád poděkoval rodině za podporu během mého celého 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 11. ledna 2016 .....

Odkaz na tuto práci

Makeš, Václav. *Portál pro podporu tvorby článků na wikipedia.org*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Práce je zaměřena na řešení problému detekce a návrhu oprav chybných a chybějících dat z internetové encyklopedie Wikipedia. Výsledkem práce je automatický systém jenž stahuje, ukládá a analyzuje články české mutace Wikipedie. K analýze jsou navrženy tři metody identifikace článků k vylepšení a doplnění. Práce ukazuje možnosti navrhování vylepšení elektronické encyklopedie.

Klíčová slova Wikipedie, vytěžování znalostí z dat, analýza textů, webová aplikace, návrh software, články encyklopedie

Abstract

The thesis is focused on solving the problem of detection and design corrections of erroneous and missing data from an Internet encyclopedia Wikipedia. The result is an automated system that downloads, stores and analyzes the Czech edition of Wikipedia articles. To analyze the proposed three methods to identify articles for improvements and additions. Work shows the possibility of proposing improvements to the electronic encyclopedia.

Keywords Wikipedia, data mining, text analysis, web application, software design, articles of encyclopedia

Obsah

Seznam obrázků

Úvod

Wikipedie je největší encyklopedií na světě, jejíž obsah může editovat jakýkoliv uživatel Internetu. Volnost editace obsahu přináší komplikace spočívající ve výskytu nedůvěryhodných a neúplných informací. Proto jsem navrhl téma prospěšné pro čtenáře encyklopedie. Cílem této práce je vyvinutí systému, který pomáhá editorům encyklopedie hledat chybějící a nerelevantní informace.

Systém strojově nachází a navrhuje možné úpravy. Periodicky je procházen obsah české a anglické Wikipedie. Nalezené problémy, např. důležité chybějící či nevalidní informace, jsou uživatelům prezentovány jako seznam bodů ke kontrole. Na samotných editorech je kontrola navržených změn a jejich případné začlenění do obsahu Wikipedie.

Obecné řešení je velmi komplikované, proto se systém tématicky zaměřuje na téma kinematografie. Řešení je přispůsobeno k tomu, aby bylo rozšiřitelné o pokrytí dalších témat.

Články encyklopedie jsou periodicky analyzovány. Nejedná se pouze o analýzu článků jako odděleně vystupujích subjektů, ale využívám asociace mezi články jednotlivých jazykových mutací. Nalezené problémové články jsou společně s popisem nalezeného problému ukládány k následnému zobrazení editorům Wikipedie.

Práce je obdobou systému Check Wikipedia (https://tools.wmflabs.org/checkwiki), který ovšem řeší pouze syntaktickou kontrolu obsahu. Oproti němu můj systém řeší sémentickou analýzu. Prezentace dat koncovému uživateli je inspirována systémem syntaktické analýzy Check Wikipedia, ovšem v příjemnější a modernější podobě.

Kapitola 1

Cíl práce

Cílem rešeršní části práce je seznámit čtenáře s problematikou internetové encyklopedie Wikipedia a nastínit problémy k řešení. Další částí cíle je seznámit čtenáře se systémy pro pomoc editorům encyklopedie a shrnout výhody a nevýhody konkurenčních řešení.

Cílem praktické části práce je navrhnout a implementovat systém pro stahování, ukládání a analyzování dat z české Wikipedie. Pro analyzování dat je nezbytné navrhnout metody identifikace článků k vylepšení a doplnění. K systému je nutné navrhnout webové rozhraní.

Kapitola 2

Wikipedia

„Představte si svět, v němž může každý člověk svobodně přistupovat k veškerým lidským znalostem. Právě takový svět budujeme.“ [1]

Rešeršní částí chci ukázat na jakých principech Wikipedie stojí a proč je důležité se jí zabývat. Základní myšlenky Wikipedie jsou otevřenost světu o poskytování informací a zároveň otevřenost pro možné editace. Lidé editující Wikipedii (pro jednoduchost jim říkejme wikipedisté) sami nabádají k editování s odvahou [2]. Tyto myšlenky přinášejí výhody i nevýhody. Proto tyto výhody i nevýhody svobodné tvorby a editace článků shrnuji a předkládám částečné řešení kontroly obsahu.

2.1 Online encyklopedie

Wikipedia je jednou z nejnavštěvovanějších stránek na světě. Celosvětově se jedná o sedmý nejnavštěvovanější server [3]. Je často používána jako zdroj prvních informací o problematice. Více než 50% čtenářů dále používá citované zdroje k prohloubení znalostí. [4]

Elektronická podoba encyklopedie má tu výhodu, že není limitována velikostí a počtem stran. Na webu mohou být stránky takřka jakkoliv veliké. Limity týkající se schopnosti webových prohlížečů načíst obsah stránek daleko přesahují množství srozumitelného a významného textu, jenž jsme schopni k jednotlivým tématům vytvořit. A pokud bychom takové množství přeci jen vytvořit dokázali, tak není technický problém text rozdělit na několik menších článků a tyto části mezi sebou provázat odkazy.

Další výhodou je to, že jakákoliv nově aktualizovaná informace je ihned dostupná na webu všech návštěvníkům. Když se například změní prezident republiky, je informaci možné ihned vidět aktualizovanou.

Informace na Wikipedii může aktualizovat kdokoliv s přístupem na internet. Pokud si wikipedista (nebo též editor) není jistý svojí editací, tak má

k dispozici před každou úpravou náhled svých změn. Zároveň může každý wikipedista použít takzvané pískoviště. Pískoviště je určené k tomu, aby se wikipedisté naučili editovat stránky. Obě tyto praktiky omezují možné nechtěné omyly editorů při editacích.

Nevýhodou tohoto otevřeného editování textu je vandalismus. Jedná se o editace obsahující vulgarity, odstraňování částí či celých článků a podobně. Dalším negativním vlivem mohou být například reklamní editace za účelem zisku. Jako opatření k těmto editacím, které by jednoznačně zhoršovaly kvalitu Wikipedie, má encyklopedie prostředky pro zrušení (nazývané jako revertování) editací. Zrušení editace způsobí navrácení stránky do původního stavu. Více o vandalismu píši v sekci 2.4.1 na straně 9.

2.2 Pravidla publikování

Publikování na Wikipedii se řídí pravidly [5]. Pravidla mají ve Wikipedii ten význam, že určují co a jakým způsobem bude v encyklopedii popsáno. Celkově vzato jde o to, aby články byly hodnotné pro čtenáře encyklopedie. Z pravidel jsem vybral ta nejpodstatnější:

- ověřitelnost,
- významnost,
- nezaujatý úhel pohledu,
- žádný vlastní výzkum,
- věrohodné zdroje.

Myslím, že názvy pravidel jsou samovypovídající, přesto je popíši, protože jsou pro fungování veřejně editovatelné encyklopedie důležité.

2.2.1 Ověřitelnost

Informace v článcích musí být důvěryhodné. Tedy ke všem informacím, jenž nelze považovat za veřejně známé, musí být uveden zdroj. Informace, které lze zpochybnit a neobsahují referenci, může kdokoliv odstranit. Všechny informace, jenž nelze považovat za zjevné, je možné v článku publikovat, pokud již byly publikovány důvěryhodným zdrojem. [6]

2.2.2 Žádný vlastní výzkum

V článcích se nesmí zveřejňovat žádný vlastní výzkum. Vlastním výzkumem jsou myšlena doposud nepublikovaná data, teorie, sdělení, dojmy, argumenty a myšlenky. Vhodným citováním zdrojů lze předejít naříčením z vlastního výzkumu. Za vlastní výzkum není považováno uspořádávání a shromažďování informací z (důvěryhodných) zdrojů. [7]

2.2.3 Nezaujatý úhel pohledu

Články mají být napsány tak, aby byly nestranné a pojednávaly o předmětu článku nezaujatě. Princip nezaujatého úhlu pohledu nechce mít za následek dodržování jednoho „objektivního“ a nezávislého pohledu. Principiálně jde o to, že by se měly představit soupeřící názory bez jakéhokoliv stranění kterémukoliv z nich. Článek by neměl názory hodnotit, ale názory pouze představit a uvést k nim náležité zdroje. [8]

2.2.4 Nedodržování pravidel

Při nesplnění některé z podmínek uvedených výše se může zakročit několika způsoby. Metoda zakročení se vybírá na základě zhodnocení prohřešku. Jde o subjektivní vyhodnocení situace wikipedisty kontrolujícího editace. Možné zákroky jsou:

- Přidání šablony označující problém – šablona v článku oznamuje co je problémem a jaké je možné řešení (příkladem může být nabádání k dodání věrohodných zdrojů).
- Revertování změny – pokud je celá úprava nevhodná, tak je možné celou editaci vrátit do přechozího stavu. K tomuto kroku dochází například při vandalismu.
- Editace úpravy – pokud jde o problémy na dílčích částech článků, tak je možné článek znovu editovat. Editací se tak článek může zbavit například nevhodných částí.

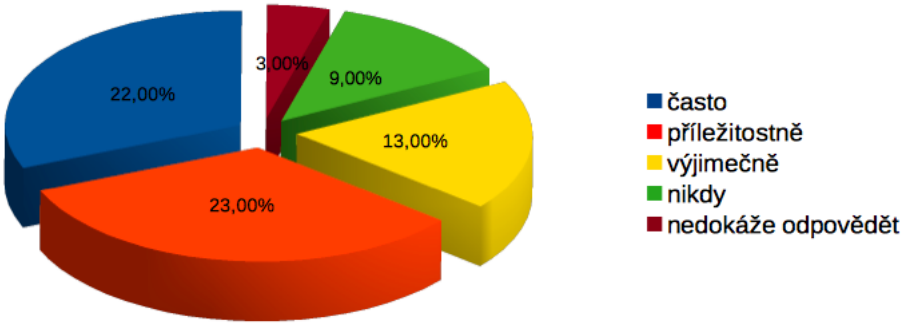
2.3 Kvalita článků

Wikipedie se stala objektem několika výzkumů. Ve výzkumech je porovnávána kvalita článků či například dopad rozšířenosti Wikipedie mezi studenty. Níže uvádím několik zajímavých studijí. Dále popíši metodu vnitřního posuzování kvality článků, kterou provádí sami wikipedisté.

2.3.1 Studie Wikipedie versus Britannica

Nejznámenší studií porovnávající kvalitu Wikipedie lze považovat studii publikovanou časopisem Nature [9]. V roce 2005 byla expertně porovnána kvalita informací encyklopedie Britannica a Wikipedie. Expertní posouzení probíhalo tak, že posuzujícím expertům byly předány články z obou encyklopedií. Zároveň ovšem experti nevěděli z které encyklopedie ten který článek je. Při posuzování kvality byly hledány tři typy chyb:

- věcné chyby,



Obrázek 2.1: Odpovědi studentů na otázku: Jak často používáte Wikipedii?

- kritická vynechání důležitých informací,
- zavádějící formulace.

Výsledek studie ukázal, že průměrně byly v encyklopedii Britannica 3 chyby na článek. Ve Wikipedii byly průměrně 4 chyby na článek. Z této studie můžeme vyvodit to, že obě porovnávané encyklopedie jsou srovnatelné. [9]

Encyklopedie Britannica je největší tištěnou encyklopedií na světě. Encyklopedii edituje omezené množství přispěvatelů. Mezi přispěvateli jsou nositelé Nobelovy ceny či američtí prezidenti. Z ekonomických důvodů došlo v roce 2012 k zastavení vydávání tištené verze encyklopedie a dále je encyklopedie distribuována pouze na optických discích.

2.3.2 Využívání Wikipedie studenty

Studie [4] z roku 2009 zkoumala využívání Wikipedie americkými studenty vysokých škol. Do průzkumu byli zapojeni vysokoškolští studenti mezi 20-30 lety. Studentům bylo položeno několik otázek. Pro tuto práci je nejzajímavější otázka: „Jak často používáte Wikipedii?“. Procentuální vyjádření odpovědí je uvedeno na obrázku 2.1. [4]

2.3.3 Wilkinson – Huberman studie kvality článků Wikipedie

Dennis Wilkinson a Barnard Huberman ve své studii [10] analyzovali vztah mezi kvalitou článků a rozsahy jejich editací. Pro srovnání byl zohledněn poměr průměrného počtu editací článků a počet přispěvatelů (wikipedistů). Poměr byl srovnáván mezi nejlépe a průměrně hodnocenými články vůči ostatním článkům (toto hodnocení článků popisuji v následující kapitole). Výsledek studie ukázal závislost mezi počtem editorů, počtem editací a kvalitou článků.

Při analýze byly články porovnávány s jejich ekvivalenty v encyklopedii Britannica.

2.3.4 Posuzování článků wikipedisty

Wikipedisté posuzují vytvořené články a těm nejvíce povedeným, které splňují kritéria dle [11], přidávají ocenění „Nejlepší článek“ či „Dobrý článek“. Kritérií je mnoho. Cílem je označit ty nejlepší články z encyklopedického pohledu. Procesem hodnocení úspěšně prošlo jen velmi málo článků. Na konci října 2015 měla česká Wikipedia více než 335 tisíc článků. Mezi „Nejlepší článek“ bylo zařazeno pouze 145 z nich. Jako „Dobrý článek“ bylo zařazeno 427 článků. Tyto množiny jsou disjunktní, nejlepší článek není zařazen v "Dobrý článek"a naopak.

2.4 Svoboda tvorby a editace článků a její následky

Každý uživatel internetu může číst a zároveň přispívat do Wikipedie. Tato otevřenost editacím bohužel přináší negativa. Jsou jimi následky v podobě neencyklopedistických článků. Níže uvádím dva problémy, jež vidím při editaci Wikipedie nejčastěji.

2.4.1 Vandalismus

Otevřenost Wikipedie a možnost editování kýmkoliv s připojením k internetu láká některé lidi této encyklopedii škodit. Jedná se například o umísťování reklamních odkazů, propagaci firem, o úmyslné odstraňování částí článků, psaní vulgarit a tak dále. Obecně lze toto jednání brát jako vandalismus. Obranným mechanismem je ruční kontrola editací wikipedisty. Wikipedisté tedy kontrolují, zda ostatní editoři svým jednáním neškodí obsahu Wikipedie. Všechny typy vandalismu a prostředky zabraňující je možné najít na [12].

2.4.2 Zanášení chyb a nepřesností

Druhým typem problémů jsou neúmyslné chyby, nepřesnosti a zavádějící formulace. Bohužel tyto problémy nelze odhalit tak jednoduše. Může se jednat například o překlep ve jméně režiséra filmu. A právě tyto problémy mnou navržená aplikace detekuje a navrhuje možné opravy.

Kapitola 3

Analýza a návrh

Před vlastní implementací systému jsem provedl analýzu existujících řešení. Poté jsem analyzoval dostupná data, se kterými jde v analýze obsahu encyklopedie pracovat. Seznámíme se se strukturou dat a jejich zpracovatelností. Zpracovatelnost dat byla důležitým krokem pro implementaci systému.

3.1 Existující nástroje pro podporu editací

Pro podporu editací Wikipedie již existuje několik různých systémů. Níže uvádím ty nejzásadnější z pohledu mé práce.

3.1.1 Check Wikipedia

Projekt Check Wikipedia je projektech kontrolující syntaktickou správnost článků Wikipedie. Jedná se o webový systém dostupný na adrese https://tools.wmflabs.org/checkwiki.

Check Wikipedia prochází všechny jazykové mutace. V každé jazykové mutaci kontroluje skupinu pravidel. Z nichž každé pravidlo popisuje konkrétní problém. Jedná se například o tyto problémy:

- Neplatné ISBN-13: indikuje, že se v článku nachází neplatné ISBN.
- Název jako odkaz: indikuje, že se v článku nachází odkaz na ten samý článek. Tato skutečnost vytváří cyklické odkazy, tudíž po kliknutí na odkaz by se čtenář ocitl na té samé stránce.
- Název kategorie s mezerou: indikuje, že se v článku nachází v názvu zařazení do kategorie mezera.

Celkově systém Check Wikipedia obsahuje 102 různých kontrolních pravidel. Tato pravidla jsou podle důležitosti rozdělena na tři úrovně.

Z tohoto systému jsem při primárně vycházel. Systém Check Wikipedia kontroluje Wikipedii syntakticky, oproti tomu má práce kontroluje Wikipedii sémanticky.

3.1.2 Boti

Jako boti jsou označovány automatické programy, jenž řeší problémy, které mohou být strojově zpracovány. Obecné pravidlo pro tvorbu botů je, že „Dřina patří strojům“. Jedná se například o opravy překlepů nebo odstraňování problémů v syntaktickém zápise. Boti jsou vytvářeny a spravovány wikipedisty. V případě nalezení problému lze činnost bota zastavit bez součinnosti s autorem bota.

Boti vytvářejí na Wikipedii desítky tisíc úprav [13]. Jejich automatické fungování pomáhá s údržnou článků, kterou by jinak museli dělat wikipedisté sami. Boti si mohou brát data o chybách například z nástroje Check Wikipedia jenž jsem zmínil výše.

Touto cestou automatických úprav bez dohledu člověka jsem se vydat nechtěl. Důvodem bylo to, že syntaktické problémy, překlepy a podobně lze velmi dobře detekovat regulárními výrazy a opravovat bez dalších nutných kontrol. Zatímco já dělám nástroj zaměřený na sémantickou analýzu. Navrhuji doplnění informací a přepracování informací. Z tohoto důvodu chci, aby změny nebyly automatické a navržené úpravy zkontroloval člověk.

3.1.3 Wikidata

Projekt Wikidata je úložištěm strukturovaných dat. Tato data jsou dostupná pro sesterské projekty nadace Wikimedia včetně projektu Wikipedia.

Data se ale moc nepoužívají, například u filmů v infoboxech je použití takřka nulové.

Na Wikidatech se nyní primárně ukládají meta data o jazykových mutacích konkrétních témat. Například Turingův stroj má svůj článek na české Wikipedii, anglické a v dalších 50 jazycích.

3.1.4 Česko-slovenská Wikipedie

Projekt Česko-slovenská Wikipedie je projektem podporujícím spolupráci české a slovenské verze Wikipedie. Projekt je primárně překládacím nástrojem v obou směrech – z češtiny do slovenštiny a obráceně. Nástroj jako takový poskytuje technické prostředky pro:

- strojový překlad z češtiny do slovenštiny a opačně,
- vyhledávání článků chybějících na jedné z jazykových mutací Wikipedie,
- podpora tvorby článků pomocí přebírání materiálů z jedné jazykové mutace do druhé,

- některé drobné editace výsledného kódu, aby pomohl editorovi.

Nástroj jako takový poskytuje pouze podporu pro práci s články. Sám nástroj neprovádí žádné editace a tím zároveň tedy nenese žádnou odpovědnost za škody způsobené jeho používáním. Wikipedisté používající tento nástroj jsou povinni před uložením svých editací zkontrolovat správné použití jakyka a syntaxe zápisu. [14]

3.1.5 WPCleaner

WPCleaner je desktopový program napsaný v Javě, jenž má pomáhat wikipedistům v údržbě článků. Program dokáže brát data z nástroje Check Wikipedia a s pomocí wikipedisty opravovat detekované problémy. Hlavní funkcionality nástroje jsou:

- opravy odkazů na rozcestníky,
- opravy chyb detekovaných na projektu Check Wikipedia,
- opravovat překlepy a gramatiku,
- pomáhat s překlady článků z jiných jazykových mutací.

3.2 Získání a zpracování dat

Články Wikipedia zpřístupňuje ve formátu XML na stránkách http://dumps.wikimedia.org/. Tato XML pro českou jazykovou mutaci automaticky stahuji. Poté je provedeno zpracování jenž spočívá v nahrání dat do relační databáze. Při nahrávání dat se zpracovávají (parsují) informace o kategoriích a portálech. Pro články zařazené do portálu „Film“ jsou dále stahovány anglické ekvivalenty.

Články z anglické Wikipedie se stahují jen některé. Důvodem je náročnost na diskový prostor na serveru. Články anglické Wikipedie se stahují tak, že z české stránky je nalezena URL vedoucí do anglické Wikipedie. Z této URL je posléze stáhnut zdrojový kód stránky. Z kódu stránky je pomocí Xpath vybrán text článku (v syntaxi jenž určila Wikipedie, nikoliv v HTML – to se děje z důvodu, aby bylo jednotné parsování informací jak z české, tak z anglické Wikipedie).

3.3 Syntaxe Wikipedie

Celá Wikipedie je editována pomocí webového rohraní. Wikipedia má vlastní syntaxi pomocí které se stránky editují.

Pro zpracování byly důležité zejména kategorie, portály a infoboxy.

3.3.1 Kategorie

Kategorie ve Wikipedii slouží k uspořádání článků podle témat. Kategorie se mohou do sebe zanořovat a pro články platí poté tranzitivní pravidlo. Například kategorie „čeští matematici“ patří do kategorie „čeští vědci“. Článek spadající do kategorie „čeští matematici“ automaticky patří do kategorie „čeští matematici“.

Příklad zařazení článku do kategorie:

```
... text článku ...

[[Kategorie:Prvočíslo]]

... text článku ...
```

Tímto způsobem se článek zařadí do kategorie „Prvočíslo“. Každý článek může mít více kategorií. Každá kategorie může mít více článků.

3.3.2 Portál

Poslední a největším analyzovaným celkem jsou infoboxy. Portály lze chápat jako velké nadkategorie. Portály se zabývají konkrétním tématem jenž je dostatečně velké. Do portálů patří například „Matematika“ či „Psychologie“.

Příklad zařazení článku do portálu je například

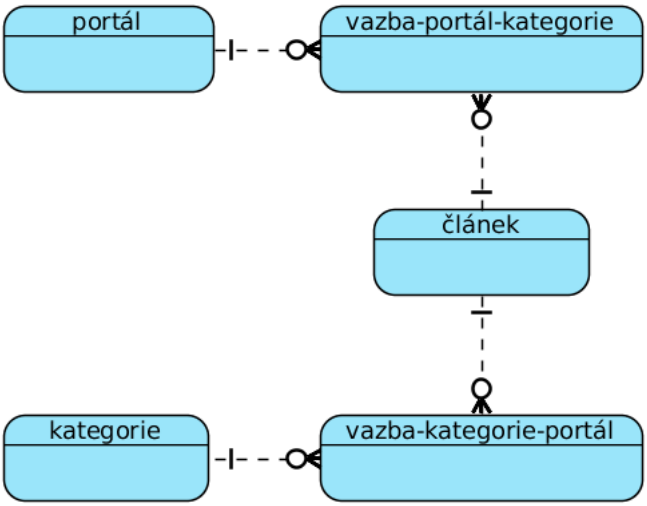
```
... text článku ...

{{Portály|Matematika}}

... text článku ...
```

Tímto způsobem se článek zařadí do portálu „Matematika“. Každý portál může obsahovat více článků. Každý článek může být zařazen ve více portálech.

3.3.3 Infobox

Zjednodušený infobox je ukázán níže. Infoboxy mohou být uzpůsobeny určitým typům článků. V analýze byly použity infoxy upravené speciálně pro kinematografii.

```
{{Infobox - film
| film = Kód Enigmy
| originál = The Imitation Game
| jazyk = [[angličtina]]
| délka = 114 minut
| žánr = [[thriller]], [[Filmové drama|drama]]
```



Obrázek 3.1: Vazby entit

```
| csfd = 283747
| kinobox = 373741
| fdb = 125641
| imdb = 2084970
}}
```

Jak fungují vazby mezi portály, kategoriemi a články je vidět na obrázku 3.1. Analogicky byly entity mapovány do relační databáze.

3.4 Hledání chybějících ČSFD identifikátorů

První analýzou je hledání chybějících identifikátorů filmové databáze ČSFD (http://www.csfd.cz/). Analýza funguje tak, že v textu hledá informaci o propojení článku s ČSFD. Informace o propojení získávám z odkazů vedoucích na projekt ČSFD. V odkazu je umístěn identifikátor filmu. Pokud odkaz systém nalezne, tak pokračuje analýzou infoboxu. Infobox článku o filmu totiž má obsahovat informaci o umístění filmu na filmovou databázi. Pokud informace v infoboxu není, tak je článek označen štítkem k úpravě. wikipedistům je pak předkládána přímo informace o možném opravené konkrétní části infoboxu včetně identifikátoru na databázi ČSFD.

Celá tato analýza probíhá pouze pomocí procházení článků regulárními výrazy.

3.5 Analýza info boxů

Články Wikipedie mohou obsahovat infoboxy. Infobox má vždy danou strukturu vzhledem k tématu. Pro analýzu jsem si vybral infobox z oboru kinematografie – infobox pro film. Infobox filmu existuje v české i anglické verzi. Toho jsem využil. Analýzu provádím v několika krocích:

- Pro české články pojednávající o kinematografii stahuji jejich ekvivalenty v anglické jazykové mutaci.
- Z obou jazykových verzí získávám celý infobox.
- Infobox rozděluji po sémantických částech (například název, režisér, apd.) do datové struktury mapy. Klíčem mapy je název vlastnosti (například režisér, rozpočet, apd.).
- Z konfigurace načítám nastavení pro porovnání. V nastavení se objevuje název klíče pro českou i anglickou verzi stránky. Dále je uvedena vlastnost. Vlastností může být například ekvivalence hodnot (například pro rok vydání), neprázdnost pole (například pro režiséra), apd.
- Pokud některá z kontrolovaných vlastností selže, je stránka označena pro editaci wikipedistou.

3.6 Návrh a analýza zařazení do existujících portálů

Některé články ve Wikipedii nemají přiřazené portály do nichž spadají. Portál je větší celek do nějž lze článek zařadit. Portály jsou například `Matematika`, `Astronomie`, apd. Každý článek může mít 0-N přiřazených portálů. Mezi články a portály je vazba M:N.

První možnost jenž mě napadla byla porovnat článek v češtině s článkem v angličtině. Na základě tohoto porovnání zařadit článek do konkrétního portálu. U tohoto řešení jsem ovšem narazil na problém s tím, že musím stáhnout všechny ekvivalenty pro české články. Zároveň by bylo nutné získat názvy ekvivalentů českých portálů v anglickém jazyce.

Z těchto důvodů jsem se uchýlil k druhému možnému řešení. Principem řešení je vyhodnocování zařazení kategorií k článkům. Tyto informace lze vyčíst pouze z dat načtených pro českou Wikipedii. U každého portálu analyzuji přiřazené kategorie článkům. Poté procházím všechny články nespadající do portálu a určuji koeficient s jakou pravděpodobností článek do portálu spadá. Lépe je celý proces vidět na následujícím pseudokódu:

```
proveď pro všechny portály, portál drž v proměnné 'portal':
  p = počet_všech_článků_v_portálu(portal)
```

```
  c = všechny_kategorie_pro_portál(portal)
  proveď pro všechny články mimo portál = portal:
    c2 = všechny_kategorie_v_článku(clanek)
    k = vypocet_koeficientu(c2, c, p)
    pokud k >= 0.5:
      navrhni zařazení portálu 'portal' k článku 'clanek'
```

Algoritmus pro výpočet koeficientu pravděpodobnosti je:

```
p = pocet_clanku_v_portalu
k = kategorie_portalu
vypocet_koeficientu(p, k, kategorie_clanku):
  spolecne_kategorie = prunik_mnozin(k, kategorie_clanku)
  koeficient = 0
  pro kazdou kategorie z 'spolecne_kategorie' proved:
    p = pocet_prvku(spolecne_kategorie)
    p2 = pocet_vyskytu(kategorie)
    koeficient += p * p2 / p
  return koeficient
```

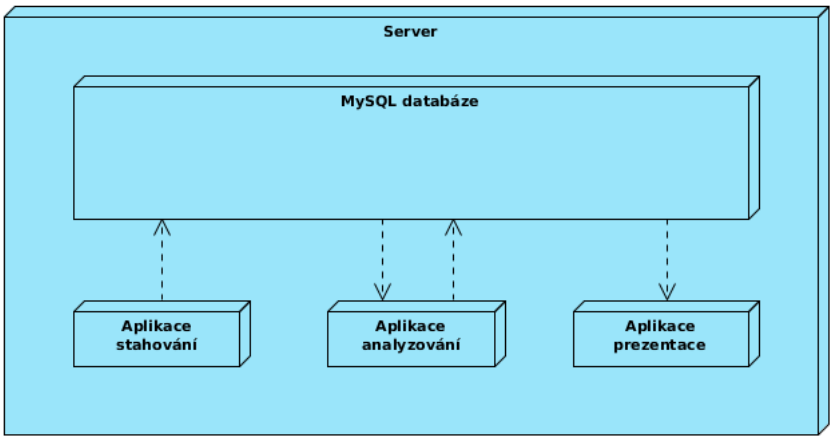
3.7 Návrh struktury softwaru

Systém byl rozdělen do tří menších částí implementovaných jako samostatné programy:

- Stahování: program stahuje XML data (zálohy) z centrálního repositáře Wikipedie. Data zpracuje (XML formát je ideální) a poté je ukládá do relační databáze. Systém zároveň stahuje HTML z anglické Wikipedie, data parsuje a poté rovněž ukládá do relační databáze.
- Analyzování: program analyzuje data uložená v relační databázi. V datech hledá problémy či chybějící údaje. Nalezené problémy poté ukládá zpět do databáze (do oddělené tabulky).
- Prezentace: program spočívá v prezentování dat uživatelům. Tento jediný program tedy řeší interakci s koncovými uživateli – wikipedisty.

Data pro anglickou Wikipedii jsou k dispozici také ve formátu XML. Důvodem proč k tomuto stahování a zpracování nedošlo a bylo přistoupeno k jiné formě získání dat bylo to, že dat na anglické Wikipedii je velmi mnoho a diskový prostor serveru nebyl dostatečný. Z těchto důvodů stahuji HTML právě těch stránek, které pro analýzu potřebuji. Toto HTML je dále parsováno tak, abych z dat ve výsledku získal stejný výstup jako z XML dumpu.

Při navrhování této architektury celého systému jsem se inspiroval Unixovými nástroji. Myslím tím nástroje jako jsou například find, sed či grep.



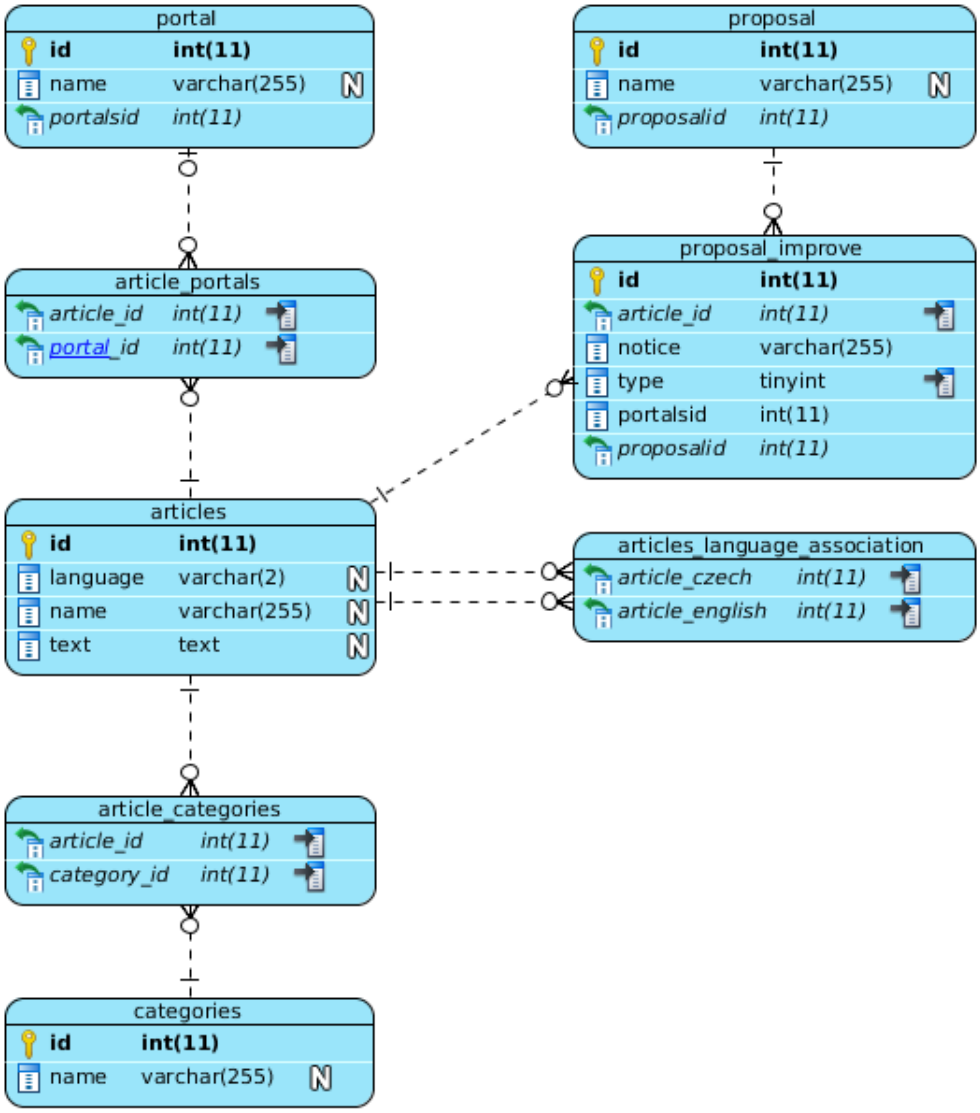
Obrázek 3.2: Deployment diagram

Obecně Unixové nástroje vynikají tím, že jsou malé a dělají právě jednu věc. To má velkou výhodu v tom, že je jednodušší je naprogramovat. Zároveň je poté jednodušší je ladit a vhodně rozšiřovat v případě potřeby. Unixové nástroje se vyznačují také tím, že fungují pomocí takzvaných pipe. Tzn. že dostanou vstup, na datech vstupu provedou nějakou operaci a vrátí výsledek. Výsledek je možné dále zpracovávat. [15]

Způsobem manipulace s daty z Unixových nástrojů jsem se inspiroval. Zdrojem dat a společným uložištěm všech tří aplikací jsou tabulky relační databáze. Data se postupně přesouvají z jedné tabulky do druhé na základě toho, kdo je právě zpracovává se transformují. Schéma aplikací a jejich práce s daty je vidět na obrázku 3.2.

Další výhoda tohoto přístupu je v tom, že jakýkoliv ze systémů je možné nahradit systémem jiným bez zásahu do dalších částí aplikace. Tato možnost byla na začátku vývoje využita. Původní proces analyzování dat byl psán pomocí jazyka Perl. V průběhu vývoje jsem část analýzy nahradil systémem psaném ve skriptovacím jazyku PHP.

V případě potřeby je možné jednodušeji celý tento systém horizontálně škálovat. Tedy přidávat další instance jednotlivých malých programů na další servery. Oproti monolitické architektuře je nutné nasazovat další instance pouze pro programy, jenž jsou zatíženy. Pro ilustraci škálování předpokládejme, že celý systém bude mít velkou návštěvnost uživatelů. Z tohoto důvodu není potřeba nasazovat na více serverů části kódu obsahující stahování či analýzu dat. Stačí zvětšit počet serverů s aplikací pro prezentaci dat.



Obrázek 3.3: ER model databáze

3.8 Schéma databáze

Na obrázku 3.3 je vidět ER model databáze. Schéma databáze rozšiřuje vztah entit jenž jsem uváděl na obrázku 3.1.

Obrázek 3.4: Úvodní obrazovka prezentační aplikace

3.9 Uživatelské rozhraní

Uživatelské rozhraní se skládá ze dvou hlavních typů obrazovek. První obrazovka je vidět na obrázku 3.4. Jedná se o snímek z webového prohlížeče po příchodu na adresu aplikace. Uživatel zde vidí rozcestník na jednotlivé typy detekovaných chyb. Po kliknutí na některý z odkazů je uživatel přesměrován na druhý typ obrazovky.

Příklad druhé obrazovky je zobrazen na obrázku 3.5. Druhá obrazovka obsahuje výpis článků s detekovaným problém. Konkrétně na obrázku 3.5 se jedná o detekované chybějící identifikátory z databáze ČSFD popsané v kapitole 3.4. Výpis detekovaných problémů spočívá v názvu článku s odkazem do české mutace Wikipedie a popisu problému. Například:

```
Až přijde kocour [odkaz na adresu
https://cs.wikipedia.org/wiki/Až_přijde_kocour]

Uvnitř článku 'Až přijde kocour' v infoboxu chybí ČSFD
identifikátor filmu. ČSFD identifikátor je '4754'.
Prosím, vložte do infoboxu filmu tento údaj: ' | csfd = 4754'.
```

Obrázek 3.5: Pohled na výpis chyb

Kapitola 4

Implementace

Při implementaci byly použity moderní postupy a metodiky. Cílem bylo vytvořit kvalitní software na kterém by mohl kdykoliv další po mě pokračovat.

V následující kapitole popisuji metody jakými jsem dbal na udržování kvality softwaru.

4.1 Výběr technologií

Níže popsané technologie jsem zvolil z důvodů, že s nimi mám dobré zkušenosti a tudíž jsem schopen v nich rychle pracovat. Zároveň tyto technologie jsou široce dostupné pro vývojáře i uživatele pod open-source licencemi. Mým požadavkem bylo to, aby jakýkoliv uživatel, jenž by chtěl s aplikací pracovat, mohl s aplikací pracovat bez překážek. To samé jsem požadoval pro vývojáře. Kdykoliv bude chtít software dále vyvíjet, tak musí být schopen získat všechny nástroje zdarma a pod open-source licencí.

4.1.1 Skriptovací jazyk

Jako skriptovací jazyk byl vybrán jazyk PHP. Původně byla jedna z částí systému programována ve skriptovacím jazyce Perl. Ovšem to se neukázalo jako dobrá volba vzhledem k tomu, že s jazykem jsem měl nulové zkušenosti. Dalšími alternativami byly skriptovací jazyky Python a Ruby. Při výběru jsem oba jazyky zvážil. Vybral jsem ovšem jazyk PHP z důvodu mých předchozích zkušeností. V jazyce PHP je napsána Wikipedie, což bylo dalším důvodem k výběru. Sdružení Wikimedia jenž programuje Wikipedii uvolňuje PHP knihovny pod open-source licencemi na serveru Github. Knihovny je možné najít na URL https://github.com/wikimedia. K použití těchto knihoven nakonec nedošlo z důvodu jejich složitějšího rozhraní, které pro moji potřebuji bylo příliš komplexní.

4.1.2 Databázový server

Jako relační databázový server jsem vybral MySQL, který je používán jako primární databáze pro Wikipedii. Při analýze jsem nenašel žádné specifické požadavky, které by dělaly problém obecně známým relačním databázím jako je MySQL, PostgreSQL, Oracle či MSSQL. MySQL jsem zvolil z důvodu dobré podpory databáze ve skriptovacím jazyce PHP.

4.1.3 HTML + CSS

Wikipedia je internetová encyklopedie. Z tohoto důvodu jsem chtěl, aby i prezentace analyzovaných výsledků byla dostupná pomocí internetu. Pro prohlížení není nutné mít zapnutý Javascript.

Wikipedia lze zobrazit na mobilních zařízeních jako jsou například mobilní telefony či tablety. Zároveň lze na těchto zařízeních obsah Wikipedie editovat. Proto bylo mým požadavkem to, aby prezentace nalezených výsledků z mého systému bylo možné na těchto zařízeních také zobrazit. K tomuto cíli mi pomohl CSS framework Twitter Bootstrap (https://getbootstrap.com/). S pomocí tohoto frameworku jsem vytvořil responsivní (tj. přispůsobitelnou) aplikaci pro uživatele.

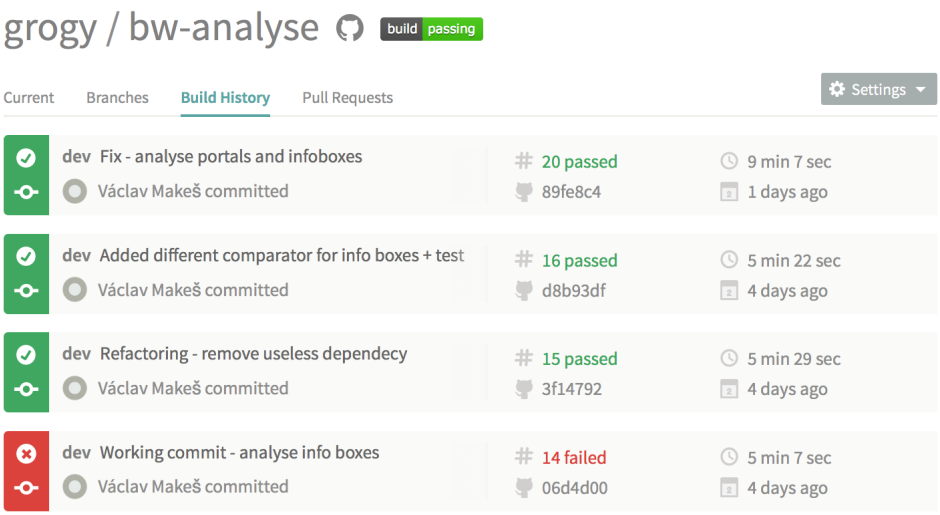
4.1.4 Externí knihovny

Rozdělení projektu na 3 menší aplikace přineslo možnost mít pro každou z aplikací jinou sadu závislostí na externích knihovnách. Tento výhodný aspekt byl při návrhu řešení zahrnut. Části Nette frameworku (https://github.com/nette/nette) byly použity napříč všemi třemi aplikacemi. Je to zvoleno z důvodu, že framework poskytuje DI (dependency injection) kontejneru, zpracování konfigurace, práce s databází a podobně. Dependency injection kontejner zajišťuje tvorbu instancí služeb bez manuálního zásahu programátora. Pokud například třída A potřebuje pro vytvoření třídu B, tak DI kontejner vytvoří obě třídy sám s tím, že třída B je automaticky předána třídě A v konstruktoru.

Oproti tomu například knihovna Matcher (https://github.com/kaja47/Matcher) byla použita pouze v aplikaci stahování. Knihovna Matcher umožňuje jednoduchým API přistupovat ke konkrétním částem HTML dokumentu pomocí definice Xpath.

4.2 Použité nástroje při vývoji

Důvodem použití níže uvedených nástrojů byl můj požadavek na rychlý a pohodlný vývoj.



Obrázek 4.1: Ukázka webového rozhraní continue integration serveru

4.2.1 Verzovací systém

Kód a veškerá jeho historie uchovávám v distribuovaném verzovacím systému Git. Vybral jsem jej z důvodu jeho rozšířenosti, možnostem, schopnostem a napojení na continue integration server. Git repositář byl umístěn na serveru Github.com. Github byl použit z důvodu jeho provázání s continue integration serveru Travis-CI. Travis-CI neumí pracovat s jiným hostingem. Propojení repositáře a continue integration serveru umožnilo automatické otestování kódu po každém provedení operace push do repositáře kódu. Zároveň každé spuštění testu je indikováno ikonkou. Zelená ikonka signalizuje to, že všechny testy procházejí. Protikladem je červená ikonka, která signalizuje problém. Stejné signalizační barvy jsou i uvnitř continue integration serveru. Příklad je vidět na obrázku 4.1.

4.2.2 Continue integration server

V kapitole 5 na straně 33 popisuji metody, pomocí kterých jsem udržoval kvalitu softwaru. K udržení kvality bylo nutné pravidelně pouštět napsané tety a kontroly kódu. Při každém provedení operace push do Git repositáře se provede předepsaná kontrola. Úspěch či neúspěch kontroly je signalizován na hostingu Git repositáře. Neúspěch je zároveň posílán na email (či při práci v týmu posílám na všechny emaily týmu vývojářů), a tak nutí programátora problém odstranit.

Existuje několik možností jak nastavených kontrol dosáhnout. Kontrolovat se může například formátování kódu či nepoužívání globálních proměnných. Systémy jenž tyto automatické kontroly provádějí se nazývají continue inte-

gration servery (CI server). Možností je například nainstalovat vlatní continue integration server Jenkins CI. Problémem takového řešení je to, že se musí nastudovat dokumentace systému a pak celý systém udržovat. Abych ušetřil čas i starosti s instalací a následnou údržou, tak jsem využil služeb třetí strany.

Vybral jsem continue integration server Travis-CI. Systém je poskytován jako cloudová služba (SaaS - software as a service). Pro konfiguraci stačí propojit účet Githubu s účtem Tavis-CI. Poté vybrat které repositáře se mají na CI serveru kontrolovat. A pak služba sama zajišťuje průběžné kontroly změn v repositáři (Github server posílá push notifikace). Kdykoliv se změní kód, tj. přibude nový commit, nová větev nebo například kdokoliv pošle pull-request, tak je spuštena celá skupina testů. Výsledek testů je zároveň vidět ve webovém rozhraní, viz obrázek 4.1.

Testy jsou definovány pomocí konfiguračního souboru `.travis.yml`. V konfiguračním souboru se řekne v jaké verzi (či verzích) PHP se mají testy spouštět. Jaké operace se mají se serverem provést (například instalace závislostí). Poté se definují spouštěče testů. Všechny tyto kroky jsou vždy provedeny a vyhodnoceny.

Ukázka konfiguračního souboru pro Travis-CI:

```
language: php

php:
  - 5.4
  - 5.5
  - 5.6
  - 7.0
  - hhvm

env:
  - TESTER_PHP_BIN=php
  - TESTER_PHP_BIN=php-cgi

matrix:
  allow_failures:
    - php: hhvm
    - php: hhvm-nightly

  exclude:
    - php: hhvm
      env: TESTER_PHP_BIN=php-cgi

before_install:
  - gem install scss-lint --no-ri --version '=0.30.0'
```

```
before_script:
  - echo "CREATE DATABASE IF NOT EXISTS test" | mysql -u root -ppass
  - composer install --no-interaction --prefer-source
  - make install-dependencies-for-travis

script:
  - make test-unit
  - make test-php-syntax
  - make test-forgot-dump
  - make qa-code-style
  - make qa-phpcpd
  - make qa-code-checker
  - make qa-phpmd
  - make test-scss-syntax
```

4.2.3 Lokální vývoj

Při vývoji jsem se setkal s několika problémy. Jedním z nich je rozdílnost platforem na kterých programátoři programují. Pokud bych chtěl sestavit přesný návod, jak nainstalovat celé prostředí pro zprovznění všech 3 aplikací, tak bych musel udělat 3 různé návody. Na Windows, OS X a Linux. Bohužel ani to by nemuselo být dostačující, protože se například některá Linuxová prostředí mohou od sebe lišit. Dále jsem se setkal s problémem, že programátor programuje pro svoji platformu. Tudíž někdy opomíná specifika prostředí na které bude aplikace nasazována.

Řešením tohoto problému může být virtualizace. Osobně jsem zvolil nástroj Vagrant. Ten dělá kompletní virtualizaci stroje. Tudíž když jsem jako operační systém na kterém aplikace poběží zvolil Debian, tak je vhodné virtualizovat Debian. Virtualizace pomocí Vagrantu má tu výhodu, že všechno nastavení může být verzováno. Tudíž se celý proces instalace prostředí pro programátora omezuje pouze na tyto dva kroky:

1. nainstalovat VirtualBox a Vagrant,
2. v repositáři pustit příkaz `vagrant up`.

Těmito dvěma jednoduchými kroky se nainstaluje naprosto kompletní prostředí včetně správné konfigurace. Výhodou je, že tato konfigurace ani nainstalované programy nezasahují do lokálního stroje programátora. Tudíž po skončení práce nezůstane programátorovi počítač nakonfigurovaný na konkrétní program.

Všechna konfigurace je uložena v textových souborech. Při spuštění příkazu `vagrant up` se tyto soubory přečtou a vykonají. Základní soubor s konfigurací uvádím níže. Celou konfiguraci jsem prováděl pomocí skriptovacího jazyka Bash. Vagrant podporuje i jiné technologie pro instalaci a konfiguraci

prostředí. Jsou jimi Chef, Puppet, Ansible, Bash, Docker a další. Osobně jsem zvolil Bash. Důvodem bylo to, že je dostatečně jednoduchý na čtení a zároveň schopný nainstalovat a konfigurovat prostředí pro běh aplikace.

Příklad základního konfiguračního souboru pro inicializaci Vagrantu:

```
VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.box = "f500/debian-wheezy64"

  config.vm.network "private_network", ip: "192.168.33.11"

  config.vm.provision :shell, path: "provisioning/upgrade.sh"
  config.vm.provision :shell, path: "provisioning/apache.sh"
  config.vm.provision :shell, path: "provisioning/php.sh"
  config.vm.provision :shell, path: "provisioning/mysql.sh"
  config.vm.provision :shell, path: "provisioning/mysql-data.sh"
end
```

Příklad instalace interpretu PHP ze souboru provisioning/php.sh

```
#!/usr/bin/env bash

apt-get install -y php5
apt-get install -y php5-mysql
apt-get install -y php5-cgi
apt-get install -y php5-curl
```

Výhodou takto uložených konfigurací ve verzovacím systému je jejich aktuálnost. Zároveň jsou soubory dokumentací pro instalaci produkčního serveru, na který jsou aplikace instalovány. Stačí postupovat pomocí předpisu, jenž je tvořen pro Vagrant, a celý nově nainstalovaný produkční server se dostane do požadavaného stavu.

4.2.4 Makefile

Některé příkazy pro spouštění testů či kontrol kódu jsou složité na zápis. Objevují se v nich například přípínače konfiguračních souborů a podobně. Použil jsem nástroj Make, abych všechny příkazy zjednodušil a popsal komentáři.

Příkazy je nutné spouštět lokálně a zároveň i na continue integration serveru. Abych neopakoval kód, tak neduplikuji ani příkazy. Sada příkazů pro spuštění testů je nadefinována pouze v souboru makefile. Příkazy v Makefile (například `make test-unit` pro spuštění jednotkových testů) jsou dále použity v konfiguračním souboru .travis.yml. Toto použití je vidět v textu výše – kde jsem uvedl ukázku souboru .travis.yml. Na konci souboru je vidět sekce

„script“ pod níž jsou make příkazy volány. Tudíž konfigurace je umístěna na jednom místě, odkud se dále používá. Používá se buď při vývoji (programátor si na lokálním počítači spustí příkaz `make test-unit` v příkazové řádce) a dále je používána na continue integration serveru. Toto zajisťuje, že při jakékoliv změně se budou spouštět stejné testy jak na stroji vývojáře, tak na continue integration serveru.

Ukázka části ze souboru Makefile:

```
...

# clean temporary files
clean:
    rm -rf tests/unit/temp/
    rm -rf src/temp/cache/


# run unit tets
test-unit: clean
    src/vendor/bin/tester -p php -c tests/php.ini tests/unit


# run SCSS syntax test
test-scss-syntax:
    scss-lint src/www/sass/

...
```

4.2.5 Regulární výrazy

Všechna data jsou uložena v textové podobě. O jejich rozdělení podle sémantiky jsem se musel postarat sám, protože všechna data jsou pomocí značek zapsána přímo v článku. Například kategorie je (většinou) úváděna na konci článku pomocí ohraničení značkou `[Kategorie:Název kategorie]`. Pro získání jednotlivých informací jsem využil regulární výrazy ve funkcích PHP, kterými disponuje. V PHP se využívá PCRE regulárních výrazů. Jde o regulární výrazy vycházející z jazyka Perl verze 5 [16].

4.2.6 Konfigurace

Všechna konfigurace je vedena textově. Celá konfigurace i produčkního stroje pro nasazení je vedena v repositáři a je tudíž vždy dostupná všem vývojářům. Proto je možné nainstalovat produkční server bez dalších znalostí systému – stačí pouze přečíst a aplikovat skripty a konfigurace v repositáře.

Výhodou textových souborů pro konfiguraci je to, že jednotlivé příkazy je možné kopírovat a například spustit pomocí příkazové řádky na produkčním

serveru. Při změnách je zároveň vidět které konkrétní části nastavení se měnily. Při změně závislostí na lokálním stroji se změna projeví ve změně instalačních skriptů. Historie změn lze díky verzovacímu systému snáze procházet a hledat například problémy po instalaci nového nastavení.

4.3 Použité postupy a techniky

4.3.1 Single responsibility principle

Třídy jsem psal dle principu jedné zodpovědnosti (SRP – Single responsibility principle). Každá třída má na práci právě jednu „věc“. Díky tomu jsou třídy i metody malé, znovu použitelné a snadno udržitelné. Nevýhodou tohoto přístupu je to, že vzniká mnoho tříd. Ovšem při praktickém používání tohoto principu se nevýhoda množství tříd stává výhodou, protože se neduplikuje žádný kód. Příklad – třída potřebuje zpracovat například externí HTML. Návrh nebude vypadat tak, že by všechno chování bylo implementováno v jedné třídě. Třída zpracování má závislost na třídě schopné stahovat HTML stránky. Takto malé třídy se i dobře testují. Mají málo chování k testování a tudíž testy jsou krátké a popisné. Závislosti tříd se dobře mockují, protože mají transparentní předávání závislostí (v kontruktoru či v setterech).

4.3.2 Kompozice před dědičností

Díky udržování principu SRP bylo pro mě snažší používat kompozici objektů a nemusel jsem používat dědičnost. Jednotlivé třídy dělají vždy jednu věc. Pokud potřebují nějakou další funkčnost, tak je zajištěna pomocí kompozice z dalších objektů. Kompozice objektů by bez kontejneru na správu závislostí byla obtížná z důvodu jejich nutnosti manuální správy.

4.3.3 DI – Dependency injection

Tříd je v aplikacích větší množství, přestože aplikace jsou malé. Pro správu tříd a správné vytvoření instancí používám dependency injection kontejner. Ten se stará o vytvoření tříd podle definic v konfiguračním souboru. Konfigurace tohoto dependency injection kontejneru jsem psal pomocí formátu neon. Neon je odvozený formát od formátu yml. Pro třídy jsem nemusel používat inicializace objektů v mém kódu – o to se staral dependency injection kontejner. Tedy volání „new MyObject“ bylo jen pro přepravky dat. Přepravkou myslím třídu obsahující data bez funkcionality. Všechnu funkcionalitu zajišťují služby. Služby fungují jen jako zpracovatelé dat. Na vstup dostanou data ke zpracování a na výstupu vracejí zpracovaný výsledek. Tím jsem zajistil jednodušší testovatelnost kódu. Zároveň je tak kód transperentnější. Třídy jsou bezestavové a tudíž nevznikají problémy s tím, že by v jiné části aplikace bylo zapomenuté nastavení dat a celá služba se pak chovala nepředvídatelně.

Pro denpendency injection lze použít několik přístupů. Vkládat závislosti lze těmito způsoby:

- Vkládáním přes konstruktor – všechny závislosti jsou viditelně předávány.
- Vkládáním přes settery – u této metody je problémem to, že se objekt může dostat do nekonzistentního stavu.
- Vkládáním přes veřejné atributy třídy – tato metoda porušuje zapouzdření objektů a beru ji jako nejhorší možnou.

V práci jsem používal striktně vkládání závislostí přes konstruktor. To má nejvíce výhod. Všechny závislosti musí být předány, jinak nejde objekt vytvořit. Neporušuje se zapouzření. Zároveň je na první pohled do hlavičky třídy vidět jaké závislosti třída má.

4.3.4 TDD – Test-driven development

Jednotkové testy byly psány metodikou TDD (test-driven development). To znamená, že jsem postupoval pomocí následující sekvence kroků:

1. Napsat minimální test – tedy nejmenší možný test, jenž nutí vylepšit implementaci o malou dílčí část.
2. Napsat nejmenší možnou implementaci, která bude procházet testy.
3. Refaktorovat kód.
4. Jít na krok číslo 1, minimální test bude obsahovat test další části požadované funkcionality.

Tato metodika má několik výhod. Největší výhodu vidím v tom, že návrh kódu je již od základu dobrý a přispůsobený na testování. S tímto kódem se dlouhodobě dobře pracuje. Kód na testování je přispůsobený z důvodu, že napřed byl napsán test. Až teprve poté implementace. Pokud bych postupoval obráceně, tak je možné že bych v testu musel provádět některé nevhodné praktiky (například použití reflexe – například pro testování privátní proměnných tříd), abych kód mohl otestovat. Další dobrým následkem metodiky TDD je to, že kód obsahuje testy. Tudíž pokud někdo bude měnit impementaci, tak systém bude automaticky testován vůči předchozímu předpokládanému chování. Testy jako takové jsou také formou dokumentace. Popisují konkrétní ukázku toho, jak autor plánoval implementaci používat.

V soupisu bodů jsem zmínil refaktorizaci. Refaktorizace znamená změny v kódu bez ovlivnění funkcionality. Změny v kódu jsou dělány s úmyslem zlepšit čitelnost a pochopitelnost kódu pro ostatní programátory. Refaktorizace by měla být prováděna po částech. Zároveň pokud se refaktorizuje kód, tak buď

měním jen implementaci nebo jen testy. Pokud bych měnil obě části najednou, tak bych mohl do systému zanést chyby a automatické testy by ztratily svůj význam. V případě, že se v kódu dělá mnoho změn za účelem zlepšení čitelnost jedná se o redesign a ne refaktoring. Dle doporučení takové změny nedělám a redesign si rozděluji na menší části, které jsem pak schopen pojmout jako refaktoring.

4.3.5 Malé úkoly

Práci jsem si plánoval po menších částech. Například když jsem sepisoval tuto textovou část bakalářské práce tak úkol z nejvyšší pohledu byl „napsat text bakalářské práce“. To je obrovský úkol. Proto jsem jej rozdělil. Úkol se rozpadl na menší části. Při procházení nových úkolů nebyly úkoly dostatečně malé na to, aby se daly zpracovat v jednotkách hodin. Proto jsem úkoly postupně rozpadal na menší a menší úkoly. Toto rozdělování úkolů má několik pozitivních následků. Při plánování práce jsem si více uvědomoval složitost a závislosti úkolu. Zároveň jsem úkoly řadil dle priority a tak mohl jednodušeji udělat MVP (minimal variable product, více podrobností níže). A poslední výhoda, kterou jsem si uvědomil byl psychologický následek – do zpracování úkolů jsem byl více motivovaný, protože to byly úkoly na menší časový úsek a byl jsem je schopen dodělat v ten den když jsem je začal dělat. [17] [18]

4.3.6 MVP – Minimum Viable Product

Celou semestrální práci jsem řešil formou minimálního prodejního produktu. Nejdříve jsem řešil to, abych pokryl naprosto celé zadání. A poté jsem průběžně zlepšoval každou část až k nynějšímu výsledku. To mi umožnilo pracovat v menším stresu. [19]

Kapitola 5

Udržování kvality

Kvalitu softwaru chápu nejen jako průběžné testování chování aplikace z pohledu uživatele. Kvalitu softwaru posuzuji i z hlediska jenž není vidět koncovým uživatelům. Myslím tím především zdrojový kód a jeho vytváření. Pokud je zdrojový kód „čistý“, tak lze jednodušeji produktovat funkční software pro uživatele.

5.1 Čistý kód

Jak poznat a hlavně jak vytvářet „čistý“ kód? Tímto tématem se zabývá mnoho knih a článků. Svoji inspiraci jsem čerpal především v [20], [21] a [22]. V angličtině je rozšířený pojem „clean code“. Vidím několik možností jak „clean code“ chápat. Především to je:

- funkční kód, který dělá právě to, co je očekáváno od aplikace,
- čitelný kód, který není složité číst a udržovat,
- otestovaný kód, který lze testovat jednoduše a ideálně automaticky,
- kód založený na standartech a to ať na standardech veřejných či standardech interních přímo pro daný projekt.

Problémem může být soustředit se pouze na první z bodů. Z tohoto důvodu vzniká spousta kódu, který sice formálně funguje a splňuje požadavky na funkčnost softwaru. Ovšem spousta tohoto softwaru je bohužel napsáno tak, že v lepším případě mu rozumí pouze autor. V horším případě kódu po uplynutí několika měsíců nerozumí ani autor a jakákoliv další údržba, oprava chyb či rozšiřování funkcionality je nákladné, frustrující a časově náročné. [20]

Uvádí se, že 90% času kód pouze čteme a nepíšeme [20]. Z tohoto důvodu kladu nejen já, ale například převážná část knihy [20] důraz na čitelnost kódu. Nejde jen o nové programátory, kteří budou na projektu pokračovat a se zdrojovým kódem pracovat. I pro samotného autora může po delší odmlce být kód

nečitelný. O tom co jsem dělal pro zvýšení čitelnosti kódu píši více v kapitole 4.3.4 věnované TDD. [20]

Otestovaný kód v tomto kontextu chápu jako kód, jenž lze testovat automaticky. Tedy na základě předem daného vstupu se testuje reálný a očekávaný výstup. Například v knize [22] se uvádí čtyři jednoduchá pravidla. Jejich znění (které jsem volně přeložil) je:

1. automatické testy signalizují úspěch,
2. kód vyjadřuje autorův záměr,
3. kód neobsahuje duplicity,
4. kód neobsahuje zbytečné části (je nejmenší možný pro to, aby automatické testy úspěšně procházely).

1. pravidlo je nejdůležitější, poslední pravidlo je nejméně důležité. Důvodem proč je první pravidlo nejdůležitější je to, že nás nutí psát dobré automatické testy. V kapitole 4.3.4 o TDD je více popsán další smysl automatických testů – důslednost v dobrém navržení designu kódu. Když mám napsány testy, tak mohu libovolně kód upravovat. S jakoukoliv úpravou vidím, zda jsem nepoškodil očekávanou funkcionalitu. Tudíž mohu lépe vyjádřit svůj záměr, mohu odstranit duplicitní části kódu a více experimentovat s implementací. Zároveň se nemusím obávat toho, že kód rozbiji, protože ihned vidím zda kód funguje či ne.

Kódem založeným na standardech myslím především obecné konvence pro psaní kódu. Důvodem proč neodlišuji zda jsou standardy veřejné či interní je to, že standardů existuje velká šíře a programátoři mohou preferovat různé z nich. Proto beru jako důležitější si v rámci projektu stanovit, který ze standardů se bude tým řídit. A poté podle vybraného standardu kód psát. Ve standardech je udáno například zda se bude odsazovat mezery či tabulátory, zda se budou závorky těla podmínek psát na stejném řádku jako podmínka samotná nebo na řádku novém a podobně. Dodržování těchto konvencí pomáhá v čitelnosti. Programátor nemusí v rámci projektu číst různě naformátovaný kód. To pomáhá udržet soustředěnost programátora na smyslu kódu nikoliv na jeho textové reprezentaci. [20]

5.2 Testování

Již jsem popsal proč je testování, a to i automatické, důležité. Software byl testován jak manuálně, tak především automaticky. Důvodem takto pečlivého testování byla má snaha vytvořit dlouhodobě udržitelný software a tím si ušetřit čas při vývoji i dalším rozvoji systému. Zároveň jsem tak ošetřil funkcionalitu systému aniž bych musel při každé změně kódu testovat celý

systém manuálně či se spoléhat na štěstí. Testování jsem prováděl na několika úrovních. Úrovně testování popisuji v kapitolách níže.

Jako podpůrné prostředky pro testování byly zvoleny open-source knihovny Nette Tester (https://github.com/nette/tester) a Mockery (https://github.com/padraic/mockery). Důvodem zvolení těchto knihoven byla jejich jednoduchost použití a zároveň má dobrá předešlá zkušenost. Použití jiných podpůrných nástrojů by nebylo překážkou.

5.2.1 Jednotkové testy

Cílem jednotkového (unit) testování je ověřit nejmenší části kódu. V mém kódu se objevují jako nejmenší části funkce a metody tříd. Všechny tyto jednotky byly testovány pomocí unit testů. Výjimkou byly metody jenž pracují s databází. Ty byly testovány jinak – především integrační a manuální testy. Na těchto částech kódu byla nejjednodušeji aplikována pravidla

- automatické testy signalizují úspěch,
- kód vyjadřuje autorův záměr,
- kód neobsahuje duplicity,
- kód neobsahuje zbytečné části (je nejmenší možný pro to, aby automatické testy úspěšně procházely).

Před popisem důvodů, proč byla tato pravidla aplikována nejjednodušeji, zavedu dva termíny:

Single responsibility principle (SRP) dle [22] říká, že jednotky by měly mít právě jednu zodpovědnost (dělat právě jednu věc). Na základě tohoto principu v kódu vzniká větší množství tříd. Například místo jedné velké třídy „Stahovače anglických článků“ vzniká více tříd. Třídy zařizují

1. stáhnutí článku z webového serveru,
2. rozparsování článku do entit,
3. entity chovající se jako přepravky dat,
4. třídy sloužící k persistování dat do relační databáze.

FIRST pravidlo dle [20] popisuje vlastnosti, který by měly jednotkové testy mít:

- Fast – jednotkové testy musí být rychlé. Programátor nesmí na vyhodnocení jednotkových testů čekat, protože by jej mohl dlouhý čas zpracování odradit od jejich spouštění.

- Isolated – izolované. Test nesmí svým chováním ovlinit jiné testy. Například při neúspěšném projití testu nesmí dojít k ovlivnění ostatních testů.

- Repeatable – opakovatelné. Testy musí být opakovatelné na jiných prostředích. Například konstantní názvy adresářů jako `/home/jenik/project/` jsou špatně, protože jiný programátor bude mít prostředí pravděpodobně nastavené jinak.

- Self-validating – vypovídající. Když test úspěšně neprojde, tak to musí být jednoznačné znamení pro programátora, že je v kódu něco špatně. Testy nesmí náhodně (ne)fungovat, protože by se na ně nedalo spolehnout a tým by je mohl začít ignorovat.

- Timely – včasné. Testy jsou psány včas, ideálně před samotným kódem. Pokud by se testy psaly až po delší době, tak hrozí opomenutí a neotestování důležitých částí kódu.

Jednotkové testování bylo nejjednodušší z důvodu, že tyto malé celky se rychle testují a měl jsem zaměření pouze na jednu konkrétní činnost. To bylo způsobeno jednak tím, že jsem postupoval pomocí metodiky TDD a zároveň se držel pravidla FIRST pro testy a pravidla SRP pro jednotky.

Závislosti tříd (například na databázi) byly mockovány pomocí externí knihovny Mockery. Testovala se vždy tak jen jednotka samotná bez vnějších závislostí.

Jednotkové testování probíhalo automaticky pomocí spoštění příkazu na lokálním počítači. Po každém provedení operace push do Git repositáře s kódem proběhlo zkontrolování na CI serveru.

5.2.2 Integrační testy

Integračními testy jsem testoval komunikaci mezi jednotkami. Zároveň v tomto testování byly třídy testovány zároveň s databází. Integrační testování s databází s sebou nese problém s konzistencí dat. Jelikož při testování je vždy potřebné vědět vůči jakému stavu (většinou databáze, ale stavem může být i systém souborů) se testuje, tak bylo nutné databázi ošetřit. Možností ošetření mě napadlo několik:

1. Uzavřít každý test do transakce a po každém ukončení testu na databázi provést rollback. Při testování této metody jsem narazil na problém, že při vyhození výjimky byl PHP skript schopen celý spadnou a nebylo možné již akci zachytit a rollback provést. To by šlo ošetřit například obalením spouštění jednotlivých testů do skriptu například v jazyce Bash.

2. Zvolil jsem metodu kdy před spuštěním každého jednoho integračního testu provádím vyčištění a nové naplnění celé databáze. Výhodou je jednoduchost řešení v poměru s obalením spouštění testů do dalšího jazyka. Nevýhodou je delší čas pro zpracování všech testů. Mimo samotných testů se vždy musí vyresetovat celý stav databáze.

Integrační testování probíhalo také jako testování jednotkové automaticky pomocí spoštění příkazu na lokálním počítači. Z důvodu časové náročnosti testu nebylo testování na lokálním počítači spouštěno často. Ovšem po každém provedení operace push do Git repositáře s kódem proběhlo zkontrolování na CI serveru.

5.2.3 Akceptační testování

Posledním testováním bylo testování manuální. Toto testování jsem prováděl vždy po dokončení nějaké ucelené části systému. Především jsem se soustředil na testování správnosti dat a algoritmů. Dále jsem dával pozor na používání softwaru z pohledu běžného uživatele. Za účelem ověření jednoduchosti uživatelského rozhraní jsem oslovil i členy rodiny. Bohužel jsem se nedostal k ověření na uživatelích Wikipedie což by přineslo nejvíce zpětné vazby. Toto testování nebylo nijak automatizováno.

Při akceptačním testování byl systém testován napříč několika zařízeními. Zařízeními byly: stolní počítač s 22"displejem, notebook s 13"displejem, tablet s 9.7"displejem a mobilní telefon s displejem o uhlopříčce 3.5". Aplikace při prohlížení pomocí mobilního telefonu a tabletu byla testován na výšku i šířku. Tímto testováním jsem ověřil zpracování mého požadavku, aby aplikace byla zobrazitelná na různých zařízeních včetně dnes často používaných mobilních telefonů.

5.3 Statická analýza kódu

Na začátku vývoje jsem si určil několik pravidel, kterými jsem se při vývoji řídil. Jednalo se o standard formátování kódu, generování dokumentace a podobně. Všechna tato pravidla jsem zanesl do kódu a zároveň do CI serveru pomocí statické analýzy kódu. Nástroje statické analýzy kódu pomáhají kontrolovat rutinní záležitosti jako například to, že kód konzistentně odsazuji tabulátory. Všechny nastavená pravidla se prochází při každém provedení operace push do Git repositáře s kódem. Při zjištění problému se automaticky odesílá email. Pro všechny nástroje pro kontrolu pravidel byly připraveny příkazy na vývoj na lokálním počítači a tak bylo možné provádět rychlé odstranění nalezených problémů bez čekání na spuštění všech testů na CI serveru.

Jedním z prvních pravidel, které jsem si zavedl, byla detekce kopírovaného kódu. Tuto detekci do repositářů zahrnuji automaticky z důvodu, že nikdy nechci mít v repositáři duplicitní kód. Duplicitní kód se špatně udržuje, zvyšuje

komplexitu kódu a zbytečně vznikají chyby [20]. Open-source knihovna phpcpd (https://github.com/sebastianbergmann/phpcpd) prochází všechen kód a na duplicity upozorňuje.

Standard pro psaní kódu jsem vybral PSR2. V tomto standardu je psáno mnoho knihoven v jazyce PHP a proto jsem se k němu přiklonil. Standardy do projektů zavádím z důvodu, že kód pak vypadá v rámci projektu jednotně a není nutné se při čtení kódu soustředit na formu zápisu.

Analyzátor PHPMD (http://phpmd.org/) kontroluje výskyty nepoužitých proměnných, počty argumentů v metodách a podobně. Pomocí tohoto nástroje je možné najít špatně navržené části kódu a nebo části kódu, které by mohly způsobit problémy. Například používání globálních proměnných nástroj přímo zakazuje.

Pro části kódu, kde by žádný test neexistoval a kód by se nespouštěl, byla přidána knihovna kontrolující kód z pohledu syntaxe. Tato kontrola se nespouštěla jen na PHP soubory, ale například také na CSS soubory ve formátu SCSS z preprocesoru Sass.

5.4 Dokumentace

Generování dokumentace kódu bylo prováděno pomocí open-source nástroje Apigen (http://www.apigen.org/). Po každém provedení operace push do repositáře se vygenerovala kompletní dokumentace projektu. Dokumentace by mohla posloužit budoucím programátorům aplikace pro zjednodušení orientace v projektu.

Závěr

Cílem práce bylo analyzovat existující řešení a na základě analýzy navrhnout systém podpory tvorby článků v české jazykové mutaci Wikipedie.

Podařilo se navrhnout systém, jenž automaticky kontroluje chybějící a nevalidní informace. Systém byl navržen jako několik oddělených služeb tak, aby celý problém byl rozdělen na dílčí podproblémy a tudíž jejich řešení bylo jednodušší. Systém pro koncové uživatele je příjemnější pro používání než obdobná konkurenční řešení. Navíc se podařilo zpracovat systém pro koncové uživatele tak, aby byl zobrazitelný responsivě – tedy zobrazitelně na displejích jakýchkoliv rozměrů. Systém analyzuje články z pohledu tří různých navržených metrik. Další metriky lze přidat, protože je systém modulární.

Nyní se systém zabývá analýzou dat z oboru kinematografie. Do budoucna je možné aktuální řešení vylepšit tak, že nebude zaměřeno na konkrétní témata, ale bude analyzovat články Wikipedie obecně. K dosažení tohoto vylepšení je nutné zapojit znalosti z datového inženýrství.

Literatura

[1] Kolektiv autorů: *Wikipedia - průvodce na cestě za informacemi*. Prostějov, první vydání, ISBN 978-80-7402-062-9.

[2] Wikipedie:Editujte s odvahou [online]. listopad 2015, [Citováno 2015-11-21]. Dostupné z: `https://cs.wikipedia.org/w/index.php?title=Wikipedie:Editujte_s_odvahou&oldid=12994371`

[3] Alexa Top 500 Global Sites [online]. září 2015, [Citováno 2015-09-07]. Dostupné z: `http://www.alexa.com/topsites`

[4] Alison J. Head, M. B. E.: How today's college students use Wikipedia for course-related research. January 2000, [Citováno 2015-09-07]. Dostupné z: `http://firstmonday.org/article/view/2830/2476`

[5] Wikipedie:Pravidlo Wikipedie [online]. leden 2016, [Citováno 2016-01-04]. Dostupné z: `https://cs.wikipedia.org/w/index.php?title=Wikipedie:Pravidlo_Wikipedie&oldid=13059430`

[6] Wikipedie:Ověřitelnost [online]. říjen 2015, [Citováno 2015-10-25]. Dostupné z: `https://cs.wikipedia.org/w/index.php?title=Wikipedie:Ověřitelnost&oldid=11742769`

[7] Wikipedie:Žádný vlastní výzkum [online]. říjen 2015, [Citováno 2015-10-25]. Dostupné z: `https://cs.wikipedia.org/w/index.php?title=Wikipedie:Žádný_vlastní_výzkum&oldid=12636011`

[8] Wikipedie:Nezaujatý úhel pohledu [online]. říjen 2015, [Citováno 2015-10-25]. Dostupné z: `https://cs.wikipedia.org/w/index.php?title=Wikipedie:Nezaujatý_úhel_pohledu&oldid=12636008`

[9] Giles: *Battle of Britannica*. Nature, 2006.

[10] Dennis M. Wilkinson, B. A. H.: Assessing the value of cooperation in Wikipedia. April 2007, [Citováno 2015-11-21]. Dostupné z: http://firstmonday.org/article/view/1763/1643

[11] Wikipedie:WikiProjekt Kvalita/Kritéria [online]. listopad 2015, [Citováno 2015-11-21]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Wikipedie:WikiProjekt_Kvalita/Kritéria&oldid=9418979

[12] Wikipedie:Vandalismus [online]. říjen 2015, [Citováno 2015-10-31]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Wikipedie:Vandalismus&oldid=12245840

[13] Wikipedie:Nejaktivnější wikipedisté [online]. listopad 2015, [Citováno 2015-11-04]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Wikipedie:Nejaktivnější_wikipedisté&oldid=12959686

[14] Wikipedie:WikiProjekt Česko-slovenská Wikipedie/Nástroje [online]. listopad 2015, [Citováno 2015-11-04]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Wikipedie:WikiProjekt_Česko-slovenská_Wikipedie/Nástroje&oldid=12872325

[15] Eric S. Raymond: *Umění programování v Unixu*. Computer Press, a.s., první vydání, ISBN 80-251-0225-4.

[16] PCRE - Perl Compatible Regular Expressions [online]. listopad 2015, [Citováno 2015-11-21]. Dostupné z: http://www.pcre.org/

[17] Knesl, J.: Agile v reálném životě [online]. listopad 2015, [Citováno 2015-11-21]. Dostupné z: http://www.knesl.com/articles/view/agilni-metodiky-v-realnem-zivote

[18] Dresler, R.: Malými krůčky ke zvládnutí velkých úkolů [online]. listopad 2015, [Citováno 2015-11-21]. Dostupné z: http://www.robertdresler.cz/2014/05/malymi-krucky-ke-zvladnuti-velkych-ukolu.html

[19] Šimeček, P.: MVP – Minimum Viable Product [online]. listopad 2015, [Citováno 2015-11-21]. Dostupné z: http://blog.pavelsimecek.cz/mvp-minimum-viable-product-2/

[20] Robert C. Martin: *Čistý kód*. Computer Press, a.s., první vydání, ISBN 978-80-251-2285-3.

[21] Andrew Hunt, David Thomas: *Programátor pragmatik*. Computer Press, a.s., první vydání, ISBN 978–80–251–1660–9.

[22] Corey Haines: *Understanding the Four Rules of Simple Design.* LeanPub, první vydání.

Příloha **A**

Seznam použitých zkratek

API Application Programming Interface

APT Advanced Packaging Tool

DFSG Debian Free Software Guidelines

DI Dependency Injection

PCRE Perl Compatible Regular Expressions

SaaS Software as a Service

SRP Single responsibility principle

TDD Test-driven development

Příloha **B**

Obsah přiloženého USB flash disku

- `readme.txt` ..... stručný popis obsahu USB flash disku
- `src` ..... zdrojové kódy
  - `bw-analyse` ..... implementace aplikace analýzy
  - `bw-download` ..... implementace aplikace stahování
  - `bw-presentation` ..... implementace aplikace webové prezentace
  - `bw-server` ..... skripty a konfigurace pro nasazení aplikace
- `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF

Příloha C

Uživatelská příručka

C.1 Instalace serveru

Na přiloženém CD v adresáři `src/bw-server` je umístěn soubor `install.sh`. Jedná se o Bashový skript pro Unixové servery s balíčkovacím systémem APT. Bashový skript obsahuje příkazy k instalaci celého serveru (webového serveru, skriptovacího jazyky, databáze, atd.). Po spuštění skriptu jsou nainstalovány všechny podpůrné programy.

Celá instalace byla testována na serveru s operačním systémem Debian 8.1.

C.2 Konfigurace DNS záznamů

Soubor `src/bw-server/000-default.conf` obsahuje konfiguraci pro webový server Apache ve verzi 2. V souboru je uvedeno doménové jméno k serveru. Toto jméno je nutné změnit a na server nasměrovat CNAME záznamy pro DNS.

C.3 Zprovoznění aplikací

Po nainstalování serveru je nutné nainstalovat aplikace. Instalace se provede pomocí spuštění bashových skriptů. Do skriptů je nejprve nutné nastavit IP adresu serveru. Poté je možné skripty spustit:

```
sh CD/src/bw-download/dev/deploy-example.sh
sh CD/src/bw-analyse/dev/deploy-example.sh
sh CD/src/bw-presentation/dev/deploy-example.sh
```