VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

GENERÁTOR VĚDECKÝCH WEBOVÝCH PORTÁLŮ

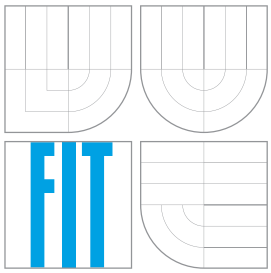
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JÁN MERTEL

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

GENERÁTOR VĚDECKÝCH WEBOVÝCH PORTÁLŮ

SCIENTIFIC WEB PORTAL GENERATOR

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JÁN MERTEL

VEDOUCÍ PRÁCE
SUPERVISOR

RNDr. PAVEL SMRŽ, Ph.D.

BRNO 2013

Abstrakt

Bakalářská práce se věnuje problematice automatickýho přidávání relevantních článků z nejakého úložistě, přičemž rešpektuje vymezený cílený záměr webového portálu. Jako součást práce je podpora personalizace portálu dle různých nastavění. Práce se soustředí na práci s článkama všeobecně, kterých odkazy jsou uschované v databáze, a další aktualizací portálu.

Abstract

Bachelor's thesis is dedicated to dealing with problems concerning automatic adding of relevant articles from some storage system, while respecting final goal of web portal. As a part of the thesis there is a support for portal personalization according to various options. Project focuses on working with articles generally, whose references are kept in database, and further actualization of the portal.

Klíčová slova

web, portál, automatizace, vyhledávaní, články, nástroj pro vyhledávaní, databáze, návrh, komentáře, administrace, PHP, Ajax, CSS, JQuery, ElasticSearch

Keywords

web, portal, automatization, searching, articles, search engine, database, plan, comments, administration, PHP, Ajax, CSS, JQuery, ElasticSearch

Citace

Ján Mertel: Generátor vědeckých webových portálů, bakalářská práce, Brno, FIT VUT v Brně, 2013

Generátor vědeckých webových portálů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Pavla Smrža, doc. RNDr., Ph.D.

.....
Ján Mertel
15. května 2013

Poděkování

Děkuji doc. Pavlovi Smržovi, doc. RNDr., Ph.D. za poskytnutou teoretickou i praktickou pomoc při navrhování a dosažení cílů práce.

Obsah

Kapitola 1

Úvod

V dnešnej dobe moderných webových technológií mnoho ľudí, spoločností publikuje svoje alebo prevzaté texty, dokumenty a iné materiály na webových stránkach, či už sa jedná o *blogy* alebo prepracovanejšie portály. Takéto webové stránky alebo portály riešia všeobecne napĺňanie obsahu manuálne. To zahŕňa napríklad aj prispievanie externými spolupracovníkmi, ako aj dopĺňanie obsahu samotnými správcami.

Dnešná doba je však zameraná obecne na automatizáciu a táto myšlienka sa nevyhýba ani webovým technológiám. Je pohodlnejšie mať automatizovaný sysstém, ktorý by samostatne dokázal riešiť vytváranie obsahu. Keďže písanie článkov je naďalej v rukách ľudí, nedá sa vravieť o úplne automatizovanej činnosti. Časť automatizácie preto začína medzi uloženou množinou článkov a končí pri publikácií.

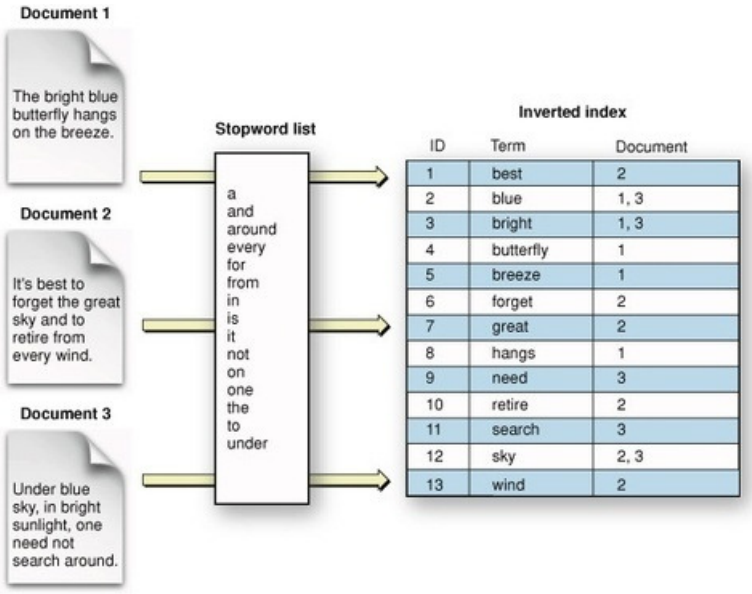
Môže sa ďalej predpokladať, že automatizácia v oblasti generovania automatického obsahu webových stránok znamená nahradenie nutnosti prostriedkov (znalostí) pre ručné dohľadanie článkov a celkovo celý proces pre konečného užívateľa podstatne zjednodušuje. Užívateľ tak môže mať k dispozícií potrebné články z určitej oblasti jedným kliknutím myši.

Podobné technológie samozrejme už existujú a sú dostupné pre návštevníkov. Avšak rovnako ako aj tieto systémy, tak aj oblasť ohraničenejších portálov má stále mnoho smerov, v ktorých sa ich ďalší vývoj môže uberať.

Tento dokument zahŕňa popis práce na možnom riešení vlastného portálu, ktorý sa zameriava na jeden, alebo viac oblastí výskumu, pričom im dokáže poskytovať potrebné články na vytvorene obsahu. V tejto funkcií veľkú rolu hraje práve vyhľadávanie relevantných článkov podľa stanovených pravidiel, aby bol výsledok dostatočne špecifický a vyhovoval správcom. V sekciách tejto kapitoly bližšie rozoberám problematiku, ktorá úzko súvisí s projektom a jej pochopenie bolo potrebné pre celkový vývoj. V úvode kapitoly **??** je bližšie rozobraná práca s náveznosťou na možné postupy a myšlienky, ktorými som sa riadil počas vývoja. Je taktiež popísaná práca na prototype a konečnom systéme, a popis ich hlavných častí, ako aj riešení rôznych problémov, na ktoré som narazil. Ďalej sú v texte použité rôzne ilustračné obrázky a schémy, ktoré popisujú niektoré algoritmy a postupy. V závere som zhodnotil dosiahnutý výsledok a výstupy práce. Záver nakoniec dopĺňa aj príloha vo forme popisu demonštračných dát, ktoré sa získali krátkym pozorovaním výsledného systému ako aj jednoduchý manuál pre inštaláciu. Často sa v niektorých prípadoch vraciam v texte nazad, aby som lepšie priblížil aktuálnu tematiku, ktorá má návaznosť na už popísánú.

1.1 Čerpanie obsahu

Základnou myšlienkou automatického napĺňania obsahu webovej stránky je mať vhodný zdroj, odkiaľ sa dajú čerpať vytvorené články, a *rozhranie*. Zdroj by mal mať dostatočnú veľkosť obsahu na demonštráciu funkčnosti a rozhranie by malo umožnovať voliteľné dotazy nad zdrojom pre nájdenie požadovaných dát a ich filtrovanie. Obsah je možné získať z mnohých úložísk, ktorých je často viac než dosť. V *open-source*, ako aj v platenej sfére sa ponúka niekoľko možností, ktoré poskytujú často aj prijateľné rozhranie. Spomeniem príkladom systém *Mendeley*[1]. Je to riešenie, ktoré ponúka, mimo iné, aj vytváranie vlastných knižníc a je dostupné v vo forme webovej aplikácie, ako aj *desktopového programu*. Podporuje taktiež metódu získavania dát s využitím *API* – aplikačné programové rozhranie. Z iných nástrojov potom spomeniem *EndNote*[2], ktorý je platený, ale ponúka pohodlný nástroj na prácu s dokumentami, ďalším vyhľadávaním a podobne. Je taktiež orientovaný ako aplikácia pre rôzne platformy. Alternatívou k takýmto už fungujúcim riešeniam (plateným aj neplateným) je použitie nástroja pre správu obsahu a vyhľadávaní v nich. Dá sa obecne uvažovať o nástrojoch podporujúcich vyhľadávanie v *indexovaných textoch*.



Obrázek 1.1: Princíp práce s invertovaným indexom

Indexácia ako koncept ponúka oproti sekvenčnému skenovaniu všetkých dát obrovskú úsporu času aj prostriedkov. Analyzovaním textov (prípadne dát obecne) sa vytvárajú štruktúry, ktoré dokážu rýchlo a pomerne presne vrátiť relevantné výsledky. Pre samotnú oblasť tvorenia dát táto práca nebola plánovaná, od začiatku sa uvažovalo iba o preberaní článkov. Analýzu a tvorbu mal na starosti úložiskový systém. Pre implementáciu výkonných vyhľadávacích nástrojov sa preto musí počítať minimálne nad rovahou okolo použitia indexácie a invertovaného indexu[8].

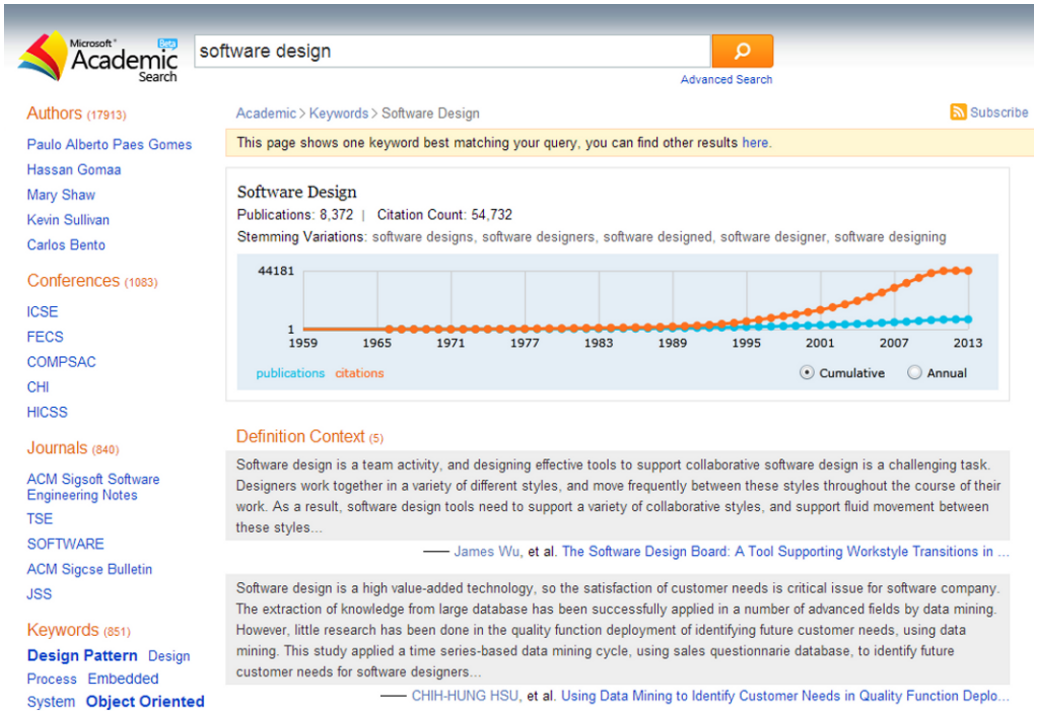
[1] https://www.mendeley.com
[2] http://endnote.com

1.2 Problematika webového portálu

Vedecký portál môže byť chápaný ako web portál zameraný na určitú oblasť alebo oblasti, z ktorých publikuje rôzne články. Zámer mojej práce spočíval v práci s takou oblasťou a jej napĺňaním vhodným obsahom. Z pohľadu článkov sa zameriava ďalej na jej obsah, Nie je žiadúce aby obsah v oblasti venovanej napríklad *polovodičovej technológií* obsahoval články z oblasti kulinárskej. Počas celého vývoja sa muselo pamätať na myslené hranice oblasti výskumu, ktoré museli dostať nejakú formu. Táto forma najlepšie inklinuje k textovému popisu. Textový popis obsahuje potrebné informácie, ktoré dokážu vhodne popísť problematiku, v mojom prípade oblasť výskumu. Bez špecifikácií hraníc oblasti by musel zdroj poskytovaťdáta spadajúce do presne vymedzenej oblasti, alebo by portál publikoval články všeobecné. Tento popis (špecifikácia) nemusí byť nutne statický, môže sa vyvýjať, prípadne do neho môže správca zasahovat a pozmeniť ho, aby mohol byť ďalej použiteľný v budúcnosti. Tým je myslený jeden z hlavných zámerov práce, konkrétne aktualizácia.

Aktualizácia, definovaná ako proces alebo nástroj, má prispieť k automatickému doplňovaniu obsahu. Pre realizáciu tejto práce sa neuvažovalo o manuálnom doplňovaní článkov, hoci vylúčené to nebolo. Istá interakcia medzi užívateľom a systémom je navyše výtaná, pretože sa často môžu objaviť zmeny, ktoré treba zohľadniť, prípadne je potrebné odstrániť niektoré publikované články. Výsledný hlavný obsah portálu je nakoniec množina článkov, ktoré boli publikované. V reálnom portále sa často stretávame s obsahom, ktorý sa nedá priamo kategorizovať ako vedecký článok. Často ide napríklad o „novinku na web stránke“. Takýto obsah táto práca neuvažuje. Uvažuje sa čisto iba preberanie článkov z vhodného zdroja (2.1.1). Čisto okrajovo by sa dalo ešte uvažovať o manipulácií s analyzačnými nástrojmi, ak by to predpisoval daný nástroj.

Články by následne mali byť vhodne prezentované. Prezentácia by mala poskytovať obsah článku v niektorej jeho forme, podľa toho ako to štruktúra dovoľuje. Prezentačná forma je veľmi často považovaná za hlavnú časť aplikácie hlavne bežným užívateľom, ktorého *backend* aplikácie nezaujíma. Výnimkou nie je ani táto práca, pri ktorej som čerpal inšpiráciu zo zaužívaných internetových vedeckých portálov, spomeniem okrajovo *Microsoft Academic* alebo *Google Scholar*. Tieto portály si taktiež zakladajú na istej intuitívnosti v prístupe a nie je vôbec zložité pochopiť ako fungujú. Výstupom práce s nimi sú odkazy na dokumenty, články a zobrazenie informácií o nich.



Obrázek 1.2: Microsoft Academic portál

Oproti týmto portálom bola ale moja práca orientovaná na napĺňanie obsahu a prezentácia výsledného obsahu bola myslená na štýl klasickej stránky. To znamená oddelenie vyhľadávacej sekcie a prezentačnej sekcie. Bežný návštevník by nemal mať prístup k vyhľadávaniu, mohol by iba zobrazovať výsledný obsah, ktorý mu už bol pripravený správcom portálu. Tento obsah môže byť zadaný kompletne manuálne alebo sa možu použiť prvky pre jeho automatické pridávanie v závislosti od nastavení. Personalizácia sa taktiež chápe ako možnosť prispôsobiť si vlastnosti a chovanie systému. Medzi to spadá mimo iné aj nastavanie vzhľadu bez nutnosti meniť pracne zdrojový kód. Taktiež by sa mohlo očakávať užívateľsky priateľské prostredie. Tieto skutočnosti preto priamo implikujú použitie nejakého administračného systému.

1.3 Administračné systémy, CMS

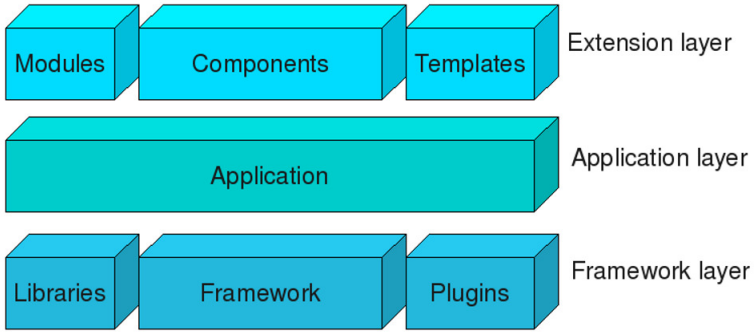
V rámci klasickej stránky alebo portálu sa dá spomenúť ľubovoľný blog ale aj napríklad webstránka internetového časopisu. Vytváranie obahu cestou editácie samotných *HTML* súborov je dávno preč. Takýto spôsob bol nielenže príliš pracný, vyžadoval navyše aj znalosť minimálne značkovacieho jazyka, nehovoriac o ďalších možnostiach, ktoré nemal. Bežný spôsom vytvárania obsahu u novodobých webových stránok je výtváranie cez nejaký administračný systém – ktorý môže byť implementovaný samostatne alebo ako súčasť *CMS* („Content Management System“ – systém na správu obsahu).

Pri vytváraní *CMS* je predpoklad používať niečo viac než len „HTML zápis“. Typicky sa používajú jazyky *PHP, ASP, Python* a iné. *CMS*, taktiež niekedy označovaný ako WCMS(„Web Content Management System“) systém, ktorý zahŕňa viaceré súčasti, ktoré majú uľahčovať prácu užívateľom.

V prvom rade pri tvorbe obsahu poskytuje nástroje, ktoré používateľom, ktorí nemajú skúsenosti s programovaním, alebo so značkovacími jazykmi poskytuju vhodnú úroveň abstrakcie pre pohodlnejšiu správu. Ďalej umožňujú meniť nastavenia a preferencie bez zásahu do zdrojového kódu, čím značne uľahčujú prácu. Pokročilejšie systémy podporujú možnosti takzvazných zásuvných modulov, ktoré ďalej rozširujú použitie. Preto sú tieto systémy značne obľúbené, pretože pri rovnakom zachovaní trendu správy umožnujú pohodlne vytvárať obsah podľa svojích predstáv.

Ďalšou komponentou, ktorá sa hojne využíva je databáza. Jedná sa o prostriedok, ktorý udržuje informácie v navrhnutej forme, ktorý nad nimi ďalej ponúka možnosti vytvárania dotazov. V databázi bývajú uložené informácie, ktoré sa môžu meniť a *CMS* priamo s týmito informáciami pracuje. Ako príklad môže poslúžiť databáza *MySQL*, ktorá ponúka podobne, ako obdobné databázy, isté rozšírenie dotazovacieho jazyka *SQL* pre komunikáciu s ňou. Pri porovnaní databázy a *search enginu*sa tu objavuje istý kontrast, napríklad spomeniem *MySQL* a *Solr*[10]. Je samozrejme možné vytvoriť cielený systém len za pomoci databázy. Taktiež ponúka vyhľadávanie v textoch, je optimalizovaná, dotazy nad niekoľko tísic článkami sú dostatočne rýchle. Väčší rozdiel sa dá všimnúť práve pri zvážení, koľko možností ponúkajú vhodné *search enginy*. Taktiež, ak sa uvažuje o rýchlosti, tak výhodu má práve obecne *search engine*.

CMS systémy sú dnes veľmi rozšírené. Ich použitie je dnes aj tam, kde sa pôvodne predpokladal statický obsah stránok. Za to môžu hlavne *open source* systémy, ktoré si môže užívateľ nainštalovať aj sám bez väčších problémov. Príkladom je napríkad *Wordpress*,*Joomla*[6] a mnohé iné.



Obrázek 1.3: Príklad *CMS* – Joomla! 1.5 Architektúra

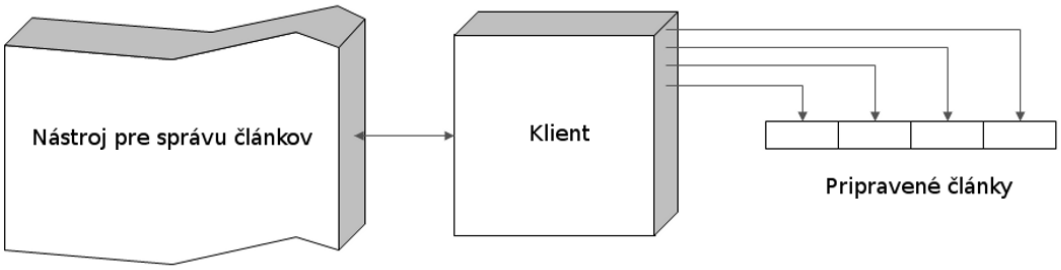
Kapitola 2

Vlastný text práce

2.1 Formulácia cieľa

V úvode som v skratke načrtol niektoré myšlienky, ktoré platili obecne pre prácu od začiatku až po jej koniec. V tejto časti ich rozšírim.

Potreboval som zdroj článkov, logiku, ktorá bude komunikovať so zdrojom a spracovávať výsledky, a vhodné zabalenie výsledkov do výstupu. Automatické vyhľadávanie malo fungovať rovnako ako manuálne, ale mala fungovať aj bez prvkov interakcie so správcom. Základná funkčnosť teda mala odrážať základnú schému práce podľa obrázku . Nebolo vylúčené začlenenie niektorých ďalších modulov do systému. Niektoré návrhy boli dokonca integrované.

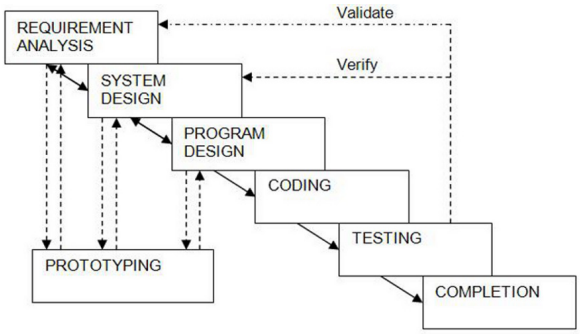


Obrázek 2.1: Základný obraz o systéme

Ďalším krokom bolo vhodne takýto systém navrhnúť, aby poskytoval prijateľné užívateľské prostredie s mnohými možnosťami (nastavenia a podobne), a následne ho implementovať. Z počiatku sa vyvýjal iba prototyp, ktorý ponúkal základnú iba funkčnosť. Tým je myslené vyhľadávanie a prezentácia článkov z „vyhľadávacieho nástroja“ (ďalej pod originálnym pojmom *search engine*). Počítalo sa taktiež s možnosťou integrovať takýto systém do zaužívaného *CMS*, vo forme *pluginu*. Vzhľadom na značnú odlišnosť takýchto *CMS* systémov nakoniec rozhodnutie padlo na vytvorenie vlastného odľahčeného *CMS*, ktorý by ale umožnoval pomerne ľahkú transformáciu do formy pluginu pre existujúce *CMS*.

Implementácia práce počítala s použitím databázy, ktorá mi bola taktiež k dispozícií. Jednalo sa o *MySQL* databázu na fakultnom počítači *athena2*. Malo sa jednať o webovú aplikáciu, to znamená použitie značkovacieho jazyka *HTML* bolo samozrejmosťou. Otázkou bolo, ktorý jazyk pre vytvorenie logiky použiť. Použitie *ASP, ASP.NET* sa na *Unixovom*

systéme naďalej vylučuje, jediné nad čím som rozmýšľal bolo *PHP* a *Python*. Nakoniec som sa rozhodol pre *PHP*, ktorý mi je ako skriptovací jazyk pre web, sympatickejší[1]. Vzhľadom na fakt, že pre obhajobu semestrálnej práce v zimnom semetri bolo potrebné mať hotový prototyp riešenia, nerozmýšlalo sa moc nad kvalitatívnou stránkou práce, jej optimalizáciou a celkovým zhotovením. Cieľ bol v prvom rade vytvoriť systém, ktorý by demonštroval možné riešenia pre konečný systém. Nebolo nutné uvažovať o administračnej a prezentačnej stránke, nevadilo ich zjednotenie. Ako metodiku vývoja som preto hneď uvažoval *Prototypový model*.



Obrázek 2.2: Prototypový model vývoja softwaru

Ak by tento prototyp vyhovoval podmienkam, plánoval som vytvoriť robustnejší systém na starom základe, čo sa nakoniec aj uskutočnilo. To znamenalo väčšiu časť prototypu kompletne zrušiť a jeho súčasti použiť v ďalšej časti práce. Vzhľadom na to, že som použil iba jeden prototyp a napokon som sa chcel toho držať, prešiel vývojový model z prototypového skôr na inkrementačný, pretože akonáhle bola architektúra CMS (ďalej bude v texte pod pojmom „administračný systém“, kôli možným rozporom pri porovnaní s reálnym CMS) implementovaná, nebolo by vhodné ju zahodiť a vytvárať nový prototyp. Bližšie detaily sú popísane v sekcií o implementácií.

2.1.1 ElasticSearch

Pre funkciu zdroja článkov, podľa sekcie v úvode (1.1) som si musel nájsť vhodný nástroj. Ako vhodná voľba sa ukázalo použitie *search enginu*), nazvaného *ElasticSearch*. Jedná sa o výkonný *search engine*, ktorý je voľne dostupný (open source) a je napísaný výhradne v *Jave*. Je postavený na *Apache Lucene*, čo je open source knižnica pre udržiavanie / získavanie dát. Jej úspech je podložený mnohými nadstavbami a skutočnosťou, že bol prepísaný do viacerých programovacích jazykov. Je známa hlavne kôli možnostiam vyhľadávania v indexovaných textoch.

[1] Manuálová stránka – http://www.php.net/manual/en/

Prístup k *indexu* som mal prakticky od začiatku práce, preto už v raných fázach bolo zrejmé jeho použitie ako *search enginu*, z ktorého sa budú získavať články. Už v tej dobe bol *ElasticSearch* nainštalovaný na fakultnom testovacom počítači a mal v sebe vyše milión článkov v konkrétnom používanom *indexe*. Tieto články pochádzajú z databázy *ReReSearch*. Každý z nich mal okrem štandardných parametrov dokumentu priradené aj dáta v poliach. Týchto polí je konkrétne 6, menovite:

- rrsid – *ID* konkrétneho článku v podľa databázy *ReReSearch*
- title – názov článku
- abstract
- authors – viac autorov oddelených bodkočiarkou
- filename
- text – obsah

ElasticSearch naďalej plne podporuje formát *JSON*, ktorý rozhodne nie je neznámy a taktiež sa používa v mnohých jazykoch. Rozhranie u *ElasticSearch* poskytuje prehľadné formovanie dotazov v tomto formáte. Poskytuje rôzne operácie a prístupy (ich zoskupenie sa na ich stránke označuje pod pojmom *API* – aplikačné programové rozhranie). Napríklad pre normálne získavanie dát (v *ElasticSearch* sú označované aj ako „dokumenty“) je dostupné „get API“. Ďalšie poskytnuté API:

- index API pre pridávanie alebo zmenu dát v niektorom indexe
- search API pre vyhľadávanie dokumentov
- update API pre zmenu dokumentov
- + iné . . .

Použité API sú ďalej popísane v príslušných častiach dokumentácie.

ElasticSearch som použil ako hlavný zdroj na vyhľadávanie článkov a vlastne celý systém bol postavený na základe fungujúcej interakcie s týmto nástrojom. Dokázal vyhovieť náročným, opakujúcim sa dotazom ako aj jednoduchým. Z dôvodu testovania a taktiež zdielaného používania na viacej projektoch bol *ElasticSearch* dostupný výhradne z daného počítača.

Keďže *ElasticSearch* ponúka priaznivé možnosti použitia *HTTP* protokolu, naskytlo sa využiť štandardnú metódu *GET*, pomocou nástroja *curl*. Ten je často štandardom v mnohých operačných systémoch a bol preto použiteľný aj na testovacom počítači *knot08*. Použitie tohoto nástroja umožnovalo preskúšanie základných ale aj pokročilejších dotazov pre *ElasticSearch*. Ako referenciu som použil oficiálnu webstránku[5].

Niektoré príklady dotazov[2]:

dotaz	funkcia
curl -XGET 'localhost:9200/'	Získanie základných informácií
curl -XGET 'localhost:9200/rrs/article/_search/ ?pretty=true'-d '{"match": {"text": "test str"}}'	Vyhľadanie spojenia „test str“
curl -XGET 'localhost:9200/rrs/article/_search/ ?pretty=true'-d '{"prefix": { "author": "Oba"}}'	Hľadanie dokumentov, ktoré majú v poli „authors“ prefix „Oba“

Samozrejmosťou sú omnoho zložitejšie dotazy, ktoré môžu byť rôzne zanorené, ako aj použitie filtrov a iných pokročilejších možností, ktoré sa v tejto práci neuvažovali.

S nástrojom *curl* prichádza s ním asociovaná knižnica *libcurl*, ktorá je podporovaná taktiež aj v *PHP* (prípadne v iných jazykoch).

2.2 Implementácia prototypu

Úlohou prototypu bolo celkovo demonštrovať možnú prácu výhradne s *ElasticSearch*. Celý prototyp zahŕňal niekoľko *.php* skriptov, klasický súbor pre kaskádové štýly (*.css*), *Javascriptovú* knižnicu (*JQuery*) s obslužným *Javascriptovým* skriptom. *Javascript* sa mal starať o užívateľky priateľské rozhranie, štýl mal ponúknuť dostatočnú estetickú úpravu a *PHP* skripty riešili logiku a funkčnosť obecne. Prototyp zahŕňal taktiež konfiguračný súbor, ktorý sa využíval po celú dobu funkčnosti prototypu a taktiež sa jednalo o *PHP* skript.

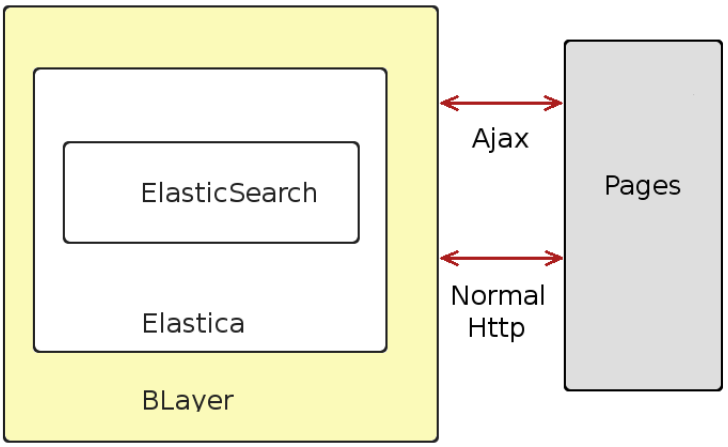
2.2.1 Elastica

Toto teda zaistilo komunikáciu a samotné vytváranie dotazov pre *ElasticSearch*. Bolo možné vytvoriť vlastný framework pre generovanie dotazov, avšak už nejakú dobu sa vyvýja jeden obecnejší, nazvaný *Elastica*. Jedná sa o typického *open source PHP* klienta, ktorý je zameraný na určitú oblasť, a tou je práve *ElasticSearch*. Obsahuje niekoľko vzájomne poprepájaných tried, ktoré sa dajú chápať ako isté abstrakcie, ktoré dokopy vytvárajú plnohodnotný dotaz. V dobe písania tejto práce nebol kompletne hotový, no ponúkal dostatnočnú podporu aj pre tvorenie pokročilejších dotazov.

Práve kôli nekompletnosti a možnosti budúcej aktualizácie som *Elasticu* vôbec neprerábal. Namiesto toho som sa rozhodoval pre možnost vytvoriť odvodenú triedu od inicializačnej triedy *Elastici*, alebo vytvoriť zcela novú triedu, ktorá by obalovala celého klienta. Vhodnejšia sa ukázala druhá možnosť, ktorá síce v ničom neposkytovala značné výhody alebo nevýhody oproti prvej možnosti, ale z pohľadu konceptu sa mi zdala prijateľnejšia. Išlo mi o to, aby som vytvoril triedu, ktorá by transformovala dotazy od iných tried (a celkovo od užívateľa, či správcu) do vhodnej formy pre *Elasticu* a obsahovala teda len základné obslužné funkcie použiteľné v tejto práci. Triedu som nazval `BLayer` podľa istej analógie — „Between Layer – medzivrstva“ (vrstva medzi *Elasticou* a vstupom od užívateľa).

[2]Doplnenie: štandardne beží ElasticSearch ako služba pod portom 9200, prípadne ďalší uzol 9201 a podobne

Trieda v tejto dobe poskytovala základnú možnosť vytvárania dotazov, pričom bolo možné použiť filtre pre upresnenie výsledku. Pri vhodnom volaní príslušných metód s parametrami sa dá potom vyhľadať jeden konkrétny článok (napríklad pri zobrazovaní detailu) alebo celá skupina. Celý prehľad tried a ich metód je dostupný na webe[11].



Obrázek 2.3: Schéma prototypu

Práca s prototypom

Práca s takýmto prototypom bola priamočiara — vo vyhľadávaní nájsť požadovaný článok, uložiť ho, a následne dohľadať články podobné. Články boli v konečnom znení zoskupené, oddelené iba podľa sekcií. Prezentačná stránka umožnovala prebehávať medzi zobrazeniami týchto sekcií. Nepodporoval funkčnosť pre hľadanie podobných dokumentov. Pri tejto oskutočnosti som použil prístup, ktorý napodobňuje cielenú funkčnosť, pre získanie článkov podobných. Jednalo sa o spôsob, pri ktorom sa použilo niekoľko filtrov a na výstupe sa objavili články, ktoré sa na základe jednoduchého algoritmu najviac podobajú prezeranému článku. O možnosti použiť *MLT Api* (More Like This — skrátene *mlt*), ktorá je priamo určená ná získanie podobných článkov som vedel a chcel som ju aj priamo implementovať. Vlastnú metódu v prototype som implementoval z dôvodu, lebo som chcel vedieť rozdiely medzi ich výsledkami. Výsledky takejto metódy boli mierne odlišné od dotazu typu *mlt Query*, nezohľadňovali ale celý obsah článku, preto som ju vypustil a spoliehal sa na celú *mlt* funkcionalitu.

Koncept prezerania obsahu v prototype bol riešený na spôsob sekcií. Každá sekcia obsahovala články, ktoré im boli pridelené podľa toho, ako boli vyhľadané. Celkovo boli tri.

- sekcia 1 – články pridané priamo z vyhľadávacieho rozhrania (manuálne)
- sekcia 2 – články, ktoré boli vyhľadané manuálne, avšak na základe toho, že boli relevantné k prezeranému článku
- sekcia 3 – články pridané automaticky administračným systémom.

Sekcia 3 nebola v tejto dobe dotiahnutá do konca, ponúkala minimálne rozdiely oproti spôsobu, ktorý používala druhá sekcia. Hlavným a najpodstatnejším rozdielom bolo pridávanie niekoľkých článkov naraz automaticky.

Prototyp uvažoval portál ako stránku, kde sú publikované články podľa nejakej témy. Prezenčná stránka bol teda jeden dokument, ktorý publikoval všetky uložené články, ktoré boli navyše oddelené podľa sekcií, do ktorých spadali.

Mnohé veci v prototype museli byť prerobené, vrátane spomínaného vyhľadávania relevantných článkov. Túto problematiku môžem rozdeliť do dvoch častí. Prvá z nich pojednáva o vyhľadávaní podobných článkov k inému dokumentu. Druhá rieši problém, ako nájsť relevantné články k téme, ktorou sa chce portál zaoberať. Prototyp sa zaoberal a načrtol možné riešenie prvej časti. Následné dokončenie sa udialo v rámci finálneho systému.

2.3 Konečný systém

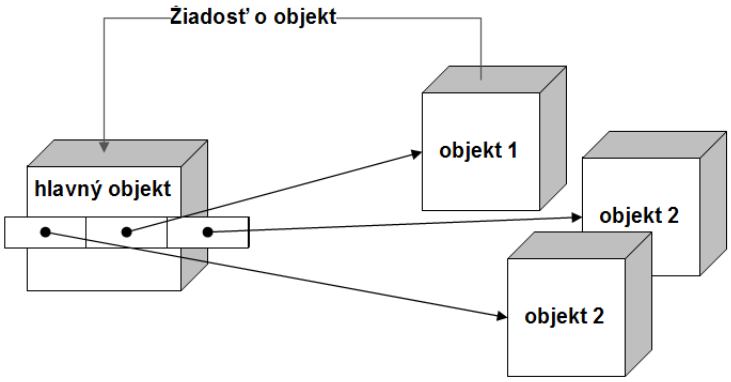
Po zhodnotení mojej práce na prototype som určil ďalší smer práce. Konečný systém som sa rozhodol spracovať ako samostatný projekt, ktorý by bol rozšíriteľný a súčasne by nevyžadoval veľa úsilia na transformáciu pre použitie v stávajúcich systémoch ako modul. Ako bolo spomínané v časti „Problematika“1.2, bolo vhodné mať oddelenú prezentačnú stránku od administračnej. Taktiež vhodné bolo vytvoriť rozhranie pre správcov, kde by bolo možné meniť nastavenia priamo, bez zásahu do kódu. Prototyp tieto prvky neposkytoval. Poskytol ale užitočné poznatky, hlavne pre prácu s *ElasticSearch*, na ktorých mohol konečný systém stavať. Niektoré funkčné celky sa zachovali (do istej miery napríklad jednoduchý *Ajax* framework), iné nevyhovovali a museli sa celé odstrániť. Zmeny sa očakávali od začiatku práce na prototype, hlavne sa nepočítalo s použitím architektúry, ktorá bola použitá.

Koncept sekcií bol zrušený, informácia o článku pridelenom ku konkrétnej sekcií sa zachováva, ale zatiaľ sa to na výstupe nijako nefiltruje. Namiesto toho sa rozvinuli kategórie, do ktorých sa články pridávajú. Platí tu ale pravidlo, že článok môže existovať iba v rámci jednej kategórie. Išlo o praktickú stránku — nebolo by žiadúce aby sa dáta na prezentačnej stránke duplikovali.

2.3.1 Architektúra

Architektúra pri prototype bola jedna z vecí, ktorá nepostačovala pre konečné riešenie. Nešlo by o realizáciu na stávajúcej architektúre, pretože to by bolo ešte realizovateľné. Išlo naviac výhradne o procedurálny štyl programovania, kde je možné systém vytvoriť rýchlo, ale za cenu ťažšej orientácie. Problém by potom mohol nastať pri väčších zmenách, nehovoriac o obecnejších rozšíreniach o nové skripty, ktoré by bolo treba začleniť do systému.

Počiatočný návrh zahrňal vytvorenie vzájomne komunikujúcich objektov, ktoré by boli prepojené cez nejaké rozhranie. Uvažovalo sa o pojme ako „zreťazenie objektov“, ktoré bolo použité nakoniec iba do istej úrovne.



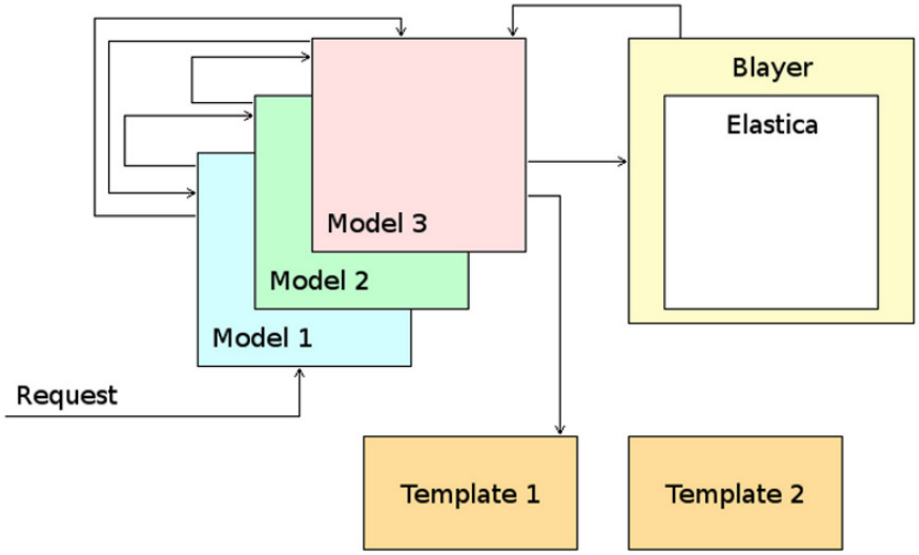
Obrázek 2.4: Myslené zreťazenie objektov

Objekty systému sú v ponímaní tejto architektúry ako myslené prvky schémy. Preto by som rád, aj napriek možnému konfliktu pojmov, ďalej v určitých častiach textu používal výraz ako napríklad „databázový objekt“ namiesto „trieda pre databázu“. Podobný návrh ponúka model *MVC* (Model – View – Controller). Tento model je široko rozšírený a používa sa rovnako ako aj pri väčších, či menších *desktopových* aplikáciách, ako aj v iných oblastniach, napríklad konkrétny web. Tento model ma svoje silné stránky, no osobne som ho nikdy moc nepreferoval tak moc, ako jeho modifikácie (v žargóne sa spomínajú *hybridy*). Formu, ktorá výtvára primárnu logiku v modeloch a snaží sa zmenšovať logiku v kontroléroch, som počas výtvárania posunul do takej, ktorá hraničí s princípmi *MVC*. Išlo konkrétne o fakt, že som modely a kontroléry skombinoval. Hlavné riadenie som posunul do jednej triedy, ktorá riadi celý tok. Neskôr som modely upravil do takej podoby, že fungujú ako klasické objekty a reagujú na podnety iných objektov. Komponentu *View* som degradoval na šablóny, ktoré sú súčasťou modelov (modelov pre stránky), čo napríklad mierne zjednodušuje štruktúru, ale pridáva malé riziko pri výskyte chyby v týchto modeloch – výstup sa deje priamo v skripte, kde je logika stránky. Pri istej úrovni abstrakcie sa tak dá nahliadať na podobnú komponentu ako *View* v *MVC*.

Bola tak použitá architektúra, ktorá sa dá oproti klasickému *MVC* lepšie pochopiť (diskutabilné, osobný názor) a rozhranie pre komunikáciu objektov. Podobné riešenie sa taktiež dá vidieť v rôznych oblastiach programovania. Modely / kontroléry sú v tejto architektúre v troch skupinách objektov (skôr ako o instancie tried sa tu jedná o spomínané prvky systému). Klasické objekty, ktoré sú nutné pri inicializácií a je ich možné ďalej využívať.

Stránkové objekty, ktorých trieda je odvodená od prvotnej. Ich logika siaha od vetvenia podľa predaných parametrov až po inú jemnú funkcionalitu. Tretia skupina je vlastne iba jediný inicializačný objekt, ktorý inicializuje ostatné ostatne objekty a predáva im riadenie.

Inicializačný objekt funguje aj ako pojivo, poskytuje rozhranie, cez ktoré môžu komunikovať ostatné objekty. Je možné pre stránkový objekt komunikovať napríklad s autentizačným objektom a podobne. Táto architektúra sa ukázala postačujúca a bola preferovaná oproti *MVC*, hlavne kôli jednoduchosti, ktorú si môže táto práca dovoliť (iba niekoľko stránok / podporných tried bez vážnejšej znovupoužiteľnosti). Pri väčších projektoch by táto architektúra mohla stagnovať, záleži na charaktere projektu.



Obrázek 2.5: Architektúra systému

Táto architektúra obsahuje niekoľko hlavných položiek. V rodičovskej zložke sa priamo jedná o súbor *index.php*, ktorý slúži ako „prístupový bod“, a konfiguračný súbor. Ďalej sú tu zložky *api*, *admin*, *js*, *style*, *templates*, *cron*, doplnené o zložku *lang*.

Prístupový bod sa dá chápať ako vstup do projektu. Zatiaľ čo hlavný bod poskytuje prístup pre bežných návštevníkov, ostatné poskytujú prístup k ďalším funkciám. Takéto body sú vytvorené taktiež v zložke *admin* a *cron*. Všetky obsashujú základné príkazy pre rozbehnutie systému. Načítajú konfiguračný súbor a vytvoria instanciu inicializačnej triedy. O ostatné sa sama postará. Takýto prístup dovoľuje vytvárať pristupnejšie *url* adresy bez použitia iných prostriedkov (*htaccess* súbor). Je možné vytvoriť jednoducho ďalšie vlastné prístupové body na ďalšiu funkcionalitu.

2.3.2 Databáza

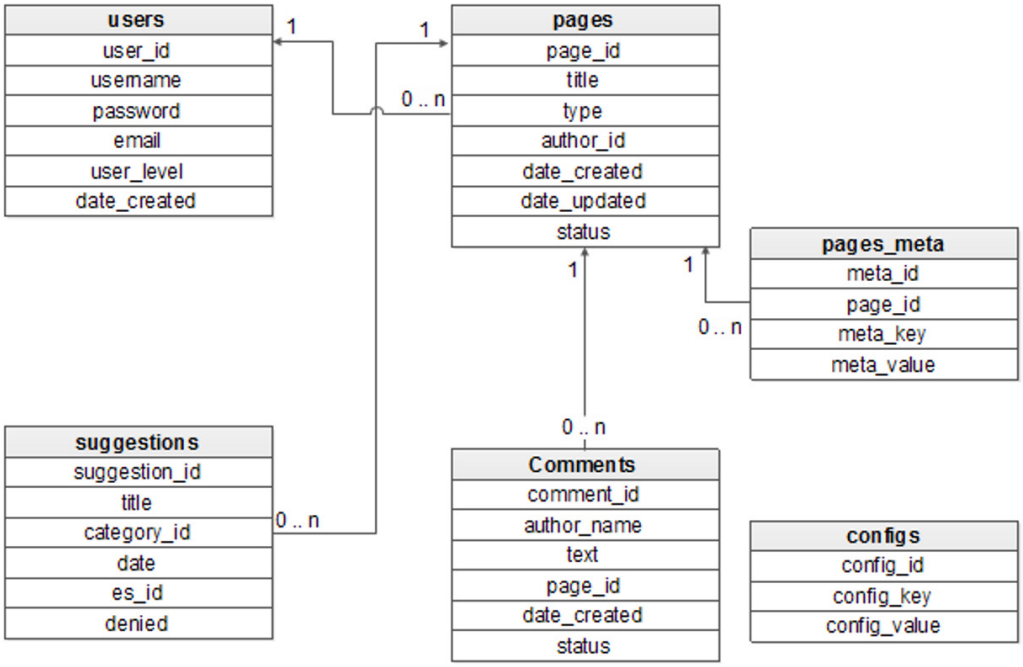
Je neodmysliteľnou súčasťou podobných systémov. Pre mňa bola poskytnutá databáza na fakultnom počítači. Prístup k nej bol možný práve z počítača, na ktorom bola práca vyvýjaná. K nástroju *phpMyAdmin* som sa nedostal, riešením bolo pre mňa napríklad vytvorenie malej správy tabuliek, pôvodne k testovacím účelom. Napokon bola táto súčasť plne integrovaná do systému a ďalej popísaná v časti 2.3.9.

Pre pripojenie a obsluhu databázy som použil triedu *mysqli* a následne ju rozšíril zdedením. Bolo možné metódy takejto triedy zcela prispôsobiť len pre aktuálne potreby, v mojej práci som ale uvažoval o väčšej obecnosti takýchto metód. Preto sú tu hlavne metódy, ktorý dovoľujú podla parametrov vybudovať rôzne dotazy do databázy na jednom mieste. Taktiež sú tu ale metódy určené iba na jedno použitie, nakoľko ich integrácia do obecnejších metód by vytvorila zložitejšiu štruktúru, nehovoriac o tom, že skladanie parametrov by bolo obtiažne. Takáto metóda je napríklad taká, ktorá zisťuje počet článkov v kategórií. V obecnej metóde by sa tak miešal *SQL* príkaz *COUNT* medzi ostatné, čo z môjho pohľadu nebolo žiadúce, aj keď by to bolo realizovateľné. Alternatívou k takýmto špecifickejším metódam by bolo použiť skrátené verzie v podobe globálnych funkcií v podpornej funkcionalite súboru pre to určenom 2.3.10. V prototype to takto bolo riešené až do rozmeru, kedy takýchto obslužných funkcií bola väčšina v danom súbore.

Databázová trieda teda ponúka rozšírenú funkčnosť oproti klasickej triede *mysqli*, ktorá je určená hlavne pre túto prácu. Zastrešuje operácie, ktoré manipulujú s databázou. Podporné metódy dovoľujú pohodlne vytvárať nové stránky, kategórie a podobne, ako aj získavať presnejšie zvolené výsledky. Sú tu taktiež metódy, ktoré slúžia na insťaláciu (2.3.9), prípadne kontrolu konzistencie databázy. Iná metóda je rozšírením funkčnosti klasickej metódy na odoslanie dotazu. Ošetruje možnosť, ak by bol dotaz neúspešný. Pri takomto type sa očakáva návratová hodnota v podobe poľa celých výsledkov. V prípade neúspešného dotazu sa musí vrátiť hodnota, ktorá nespôsobí ďalšie problémi a uľahčí sa tým ďalšie spracovanie vo volajúcej funkcií (metóde) — vráti sa jednoducho prázdne pole.

Objekt tejto triedy je ako niekoľko ďalších hneď vytváraný po štarte systému (štart v tomto ponímaní znamená inicializáciu prvej triedy). Tento konkrétne po inicializácií zisťuje stav datábázovej štruktúra a naďalej poskytuje podpornú funkčnosť. Ak nie je, systém vyhodí chybu vo forme chybovej stránky. a skončí.

Štruktúra databázy pozostáva zo šiestich tabuliek. Návrh štruktúry zodpovedá tretej normovanej forme. Tabuľka *pages* zoskupuje všetky objekty, ktoré sú chápané ako stránky a odlišuje ich jediným atribútom *type*. Keďže sa očakáva, že rôzne typy stránok budú mať navzájom odlišný súbor ďalších atribútov, bola vytvorená tabuľka *pages_meta*. Vzťah medzi týmito tabuľkami je teda *1:N*, keďže jedna stránka môže mať viac pridaných atribútov, praticky 0 až N. Takéto atribúty majú pomenovanie „meta atribúty“ alebo skrátene „meta“. Bolo by možné schému preorganizovať a mať tabuľku len pre konkrétny typ stránok bez *pages_meta* tabuľky. Osobne preferujem aktuálne použitú schému, lebo dovoľuje pohodlnejšie a kompaktnejšie rozšírenie v prípade nového typu stránok. Celú schému popisuje obrázok.



Obrázek 2.6: Architektúra systému

Databázový objekt naďalej slúži k získavaniu nastavení, či už globálnych alebo tých, ktoré sú určené pre konkrétnu stránku. Takéto funkcie hľadajú konkrétny riadok v príslušnej tabuľke a vrátia potrebnú hodnotu. Je treba počítať, že niektorá hodnota bude prázdna, prípadne nebude zadaná. Toto je možné napríklad po inštalácií, kedy sa používajú v nastaveniach portálu hodnoty len z konfiguračného súboru, ako pevný základ. Inak povedané — nie sú v databáze uložené žiadné nastavenia. Pre tento prípad je v metódach vracajúcich takéto parametre integrovaná podpora pre vrátenie poistnej hodnoty, ktorá sa použije, ak sa potrebná hodnota nezíska. Je tak zaistený stav, kedy by sa inak žiadne nastavenie nepoužilo z dôvodu absencie. Podobná metóda je použitá aj v iných systémových objektoch.

Zmyslom databázového objektu je taktiež kontrolovať chyby, ktoré by sa zaniesli do databázy. Kontrola spočíva v správnosti parametrov, je potrebné aby tam, kde je konštrukcia na prechádzanie poľom, nebol použitý reťazec. Kontrola hodnôt bola schválne umiestnená až do tohoto objektu, pretože pri ďalšom používaní metód danej triedy je pohodlnejšie

vedieť, že chyby su kontrolované a na úrovni volajúcej vrstvy sa stačí venovať logike pre prácu s výsledkami. Podobné chyby by mohli vyvolať rôzne varovania počas spracovávania skriptov, prípadne až fatálne chyby s následkom okamžitého ukončenia vykonávania skriptu. Toto by sa dalo odstrániť častým používaním *try* a *catch* konštrukcií, názornejšie je podľa mňa ale použitie stávajúceho spôsobu. V extrémnych prípadoch taktiž môže nastať situácia, ktorá má označenie *SQL injection*, ktorá pri správnom zneužití dokáže kompletne zničiť databázu.

Je to technika, ktorá sa často používa k útokom na aplikácie používajúce databázu, v tomto prípade konkrétne na báze *SQL*. Základný spôsob ochrany spočíva v použití ošetrujúcej funkcie, ktorá skontroluje reťazec predávaný ako dotaz. Technika využíva faktu, že je niekedy možné ako vstup do funkcií zadať taký reťazec, ktorý narúša očakávnú formu reťazca pre dotaz. V praxi to znamená, že očakávaný dotaz, ktorý zmaže jedného užívateľa podľa podmienky zhodného *ID*, sa zmažú všetci úžívatelia. Nedokonalý dotaz môže namiesto formy `... WHERE user_id = 2` dostať formu `... WHERE 1 = 1`[3]. Podobné odkryté chyby by sa mali eliminovať a je otázkou dobrých návykov ošetrovať ľubovolný parameter, ktorý sa má v databáze použiť. Pre doplnenie uvediem, že ako funkciu pre ošetrenie myslím funkciu, ktorá kombinuje viac prístupov. Často panuje názor, že sa *SQL injekcií* dá predísť metódou triedy *real_escape_string* (plus obdobou v pôvodnom *mysql* rozšírení). Nie je to kompletne pravda, názorne sa dá použiť mnoho príkladov z internetu. Problémom je nedostatok danej metódy (funkcie)[4]. V kóde v niektorých prípadoch filtrujem predané parametre. *PHP*, ako dynamicky typovaný jazyk, dokáže pobrať do premennej pole ale aj reťazec. Má to svoje výhody aj nevýhody, ale v tomto konkrétnom prípade ide skôr o pridanú nutnosť kontrolovať aj typ premennej. Nesprávny typ by mohol v ďalšom spracovávaní spôsobiť fatálné chyby. Kontrolou sa dajú potom odhaliť možné chyby a vhodne prerušiť ďalšie vykonávanie kódu.

Štruktúra tabuliek navyše zahrňa použitie istého prefixu. Dôvod k tomuto je možnosť použitia viacerých portálov na jednom systéme. Tabuľky tak môžu koexistovať vo viacerých instanciách. Podkladom pre tento návrh sa dá nájsť aj u úspešných *CMS* systémoch, napríklad *Wordpress*[7]. Konkrétny prefix pre portál je uložený ako konštanta v konfiguračnom súbore. Po inštálácií stráca význam ho meniť.

Pre kompletnosť návrhu sa použili pri tvorení štruktúry aj isté obmedzenia (anglicky *constraints*). Konkrétne ide o použitie cudzích kľúčov. Neuvažoval som tu možnosť kaskádových operácií, rieši sa to vhodným poradím dotazov, napríklad pri mazaní. Ďalšie obmedzenie, ktoré spomeniem, je použitie unikátnosti niektrých hodnôt. Nie je dovolené vytvárať užívateľa s rovnakým menom, aké má už existujúci, rovnako ako je nariadené mať stránku s unikátnym nadpisom.

[3]Tento príklad je čisto ilustračný, mohol by nastať iba ak by užívateľ mohol nejako zmeniť celú časť podmienky. V bežnej injekcií sa používajú šikovnejšie techniky

2.3.3 Beh systému

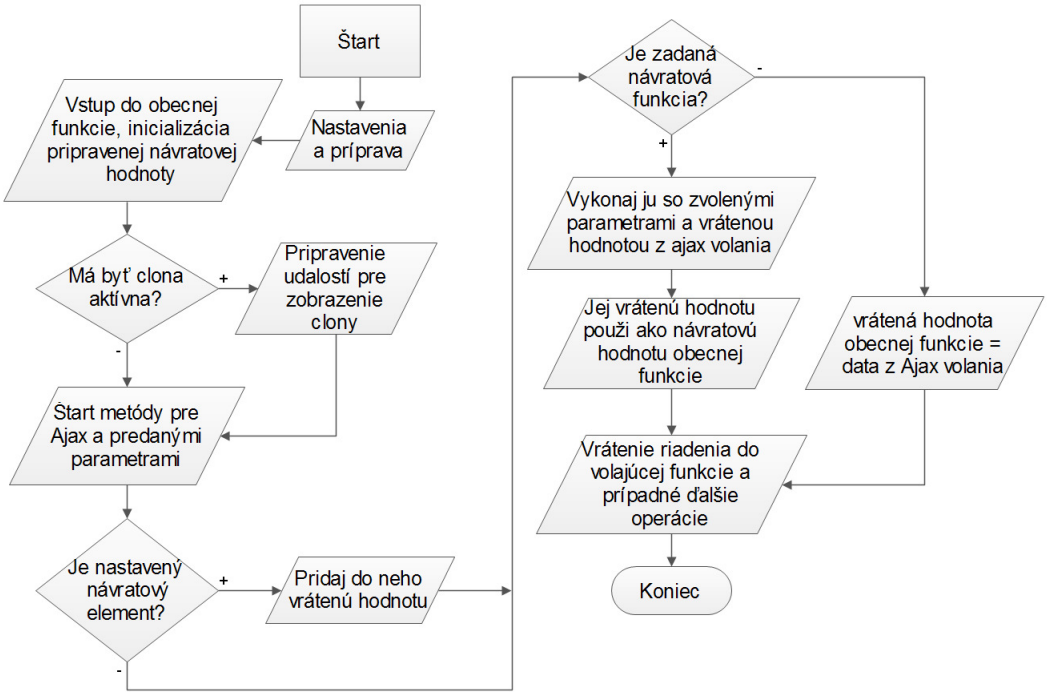
Pri štarte systému sa inicializujú potrebné triedy. Takéto systémové objekty sú určené k špecifickému účelu. Ide napríklad o už spomínaný databázový objekt ale aj o objekt pre podporu *Ajax* volaní. Použité sú aj mnohé ďalšie, ktoré sú určené pre správny chod systému. Je tiež možné pridávanie nových prvkov do procesu inicializácie podobným spôsobom. Stránkové objekty by sa mali chápať ako navzájom zamieňatelné, pričom systém beré v konkrétnu dobu do úvahy iba jediný stránkový objekt. Na základe tejto myšlienky je možné vyvolať chybu, ktorá je taktiež riešená ako stránkový objekt, a pohodlne sa s ňou dá nahradiť stávajúci stránkový objekt. Súčasne pri štarte prebiehajú aj obvyklé rutiny ako kontrola databázy, napĺňanie štruktúr, prípadne aj skontrolovanie plánovania pre kategórie. Ďalšou metódou inicializačnej triedy sa potom spustí pridávanie stránkového objektu a kontrola parametrov pre neho určených. Stránkový objekt sa naďalej ujme réžie, vykoná nutné operácie a uloží získané dáta do štruktúr v sebe.

Následne na radu prichádza kontrola, či sa jedná o *Ajax* volanie, alebo o normálny požiadavok. Ak sa jedná o *Ajax*, vykoná sa predvolená operácia a činnosť sa skončí. Ak nie, použije sa metóda rodičovskej triedy pre stránky, ktorá použije (načíta)zvolenú šablónu. Šablóna teda ako jej súčasť jednoducho pristúpy k dátam, ktoré boli v predchádzajúcom kroku uložené a vypíše ich na potrebné miesta, alebo v prípade zvolenia inej metódy sa žiadny výpis nekoná. V prípade, že počas vykonávania kódu nastane udalosť, ktorá si žiada zobraziť chybovú stránku, tak sa jednoducho nahradí aktuálna stránka chybovou. Ďalšie vykonávanie kódu sa buď ukončí alebo sa pokračuje ďalej.

Ajax a akcie v stránkovom objekte

Pre podporu *Ajaxu* ako takého je vytvorená trieda, ktorá uchová v sebe dvojice identifikátor a názov asociovanej metódy. Následne sa stará aby táto metóda danej stránky bola volaná. V jednej *Ajax* žiadosti je volaná práve jedná metóda, nikdy nie viac. Je to dané zajedno tým, že nebolo potreba vytvárať *Ajax* žiadosť pre dve a viac súčasných volaní z hľadiska použitia, a za druhé tým, že sa ako identifikátor používa iba jedna hodnota v *HTTP* žiadosti.

Takéto prepojenie na metódy má na starosti konkrétny stránkový objekt. Pre *Ajax* sa vytvoril malý *framework*, ktorý za pomoci knižnice *jQuery* dostatočne zobecňuje takéto žiadosti. Skôr ako o *frameworku* sa dá vravieť o rozšírení funkcie zo spomínanej knižnice (túto ďalej volám ako obecná) a používanie ďalších podporných funkcií. Je to napríklad funkcia pre zobrazenie prebiehajúcej udalosti. Prakticky sa to na obrazovke premietne ako clona s malou animáciou. Cez to, ako som to nazval, sa stále jedná o prístup, ktorý využívajú všetky *Ajaxovské* volania a ponúka komfortné možnosti. Je dovolené taktiež používať obecnú funkciu aj pre *GET* a *POST* metódy. Stačí len zmeniť parametre a použiť podpornú funkciu na prevod dát z *GET* do *POST* zápisu. Pri metóde v obecnej funkcií sa následne využíva návratová metóda *done*, ktorá sa stará o obsluhu úspešnej žiadosti. Metóda *fail* je metóda, ktorá sa s ňou môže dopĺnať, je ako poistka pre prípad chyby. Je to bežný postup pri používaní *jQuery* verzie 1.8 a vyššej. Vo funkčnosti nebráni, ale môže informovať užívateľa o chybe. Použitie tejto metódy som ale zamietol, pretože kôli lokalizácií nie je možné inak pohodlne, ako predávaním chybovej správy ako parametra oznámiť užívateľovy chybu v potrebnom jazyku. Nebolo mojím úmyslom zbytočne pridávať parameter, a tak som metódu *fail* ďalej neuvažoval, to znamená, že pri chybe volania sa prakticky nič nestane, okrem zobrazenia možnej clony na pár okamihov. Základ je v hlavnom *Javascrptovom* súbore. Konkrétnejšia funkcionalita je v iných, pri ktorých ale neplatí, že sa nemôžu používať len v jedinom stránkovom objekte.



Obrázek 2.7: Obecný prístup k Ajax volaniu

Algoritmus popisuje štart od špecifickej funkcie na volanie obecnej, ktorá volá metódu knižnice *jQuery*. Zohľadňuje prístup s voľbou použiť *asynchrónny* mód. Normálne sú všetky *Ajax* volania *asynchrónne*, ale s použitím vhodného prepínača sa to dá zmeniť a získa sa tým možnosť, aby vykonanie obecnej funkcie, presnejšie vrátenie jej návratovej hodnoty, počkalo na dokončenie *Ajax* dotazu. Pri niektorých volaniach sa tým dá docieliť väčší komfort, prípadne odhalenie možnej chyby, ktorá nastane, keďže sa dá manipulovať s kontrolovaním vrátenej hodnoty. Naneštastie má tento mód nedostatky a nedoporčuje sa ho používať, lebo môže donútiť prehliadač, aby prestal reagovať[9]. Ak je nutnosť používať návratovú hodnotu pre ďalšie spracovanie, nedá sa očavávať vrátenie tejto hodnoty von z obecnej funkcie. Na to je nutné používať takzvané *callback* prostriedky. Pri volaní obecnej funkcie je možné predať takýto prostriedok, ako názov funkcie, ktorá má byť následne spracovaná. Samozrejmosťou je predanie aj dalších parametrov, ktoré sa doplnia s vrátenými dátami z *Ajax* volania.

Akcie sú potom metódy, ktoré majú podobný charakter ako metódy pre *Ajax*, s tým rozdielom, že sú volané synchrónne pri príchode *HTML* dotazu a tok v nich sa vypisuje normálne na obrazovku. Sú ale určené napríklad na spracovanie dát z formulára a na ich konci sa volitelne používa presmerovanie. Správy sú metódy čisto informatívneho charakteru. Doplňujú dáta pre šablónu, ktorá určitým spôsobom informuje užívateľa. Napríklad o tom, že kategória bola vytvorená.

Tieto tri typy podporných metód sa viažu na konkrétne dáta v *HTTP* žiadosti. Nerozlišuje sa potom, či ide o hodnoty metód *POST* alebo *GET*. Je potrebné ale zachovať určitý systém. Tieto prepojenia sa musia uskutočniť na začiatku, hlavne pred tým než je volaná rodičovská metóda, ktorej stačí, aby boli názvy Akcií a správ definované a aby metódy mali požadovaný tvar názvu (správa označená ako „pozor“ by mala mať definovanú me-

tódu „message_pozor“). Pri *Ajaxe* to môže byť aj neskôr, hlavne nech je to pred kontrolou, o ktoré volanie sa jedná (*Ajax* alebo klasické *HTML*. Táto kontrola na výskyt hodnoty v *HTML* žiadosti je nutná, kedže prehliadač nevidí normálne rozdiel medzi týmito typmi dotazov.

Samozrejmosťou je pridanie potrebných skriptov do šablóny. To je možné napríklad použitím vhodnej metódy rodičovskej stránky, ktorá ďalej používa iné globálne funkcie (2.3.10). Očakáva sa pridanie knižnice *jQuery*, malého *frameworku* pozostávajúceho z niekoľko málo funkcií, ktoré uľahčujú a zobecňujú prácu a skriptu so špecifickými volaniami pre daný účel. Je tak oddelená funkcionalita napríklad vyhľadávacej stránky od inej.

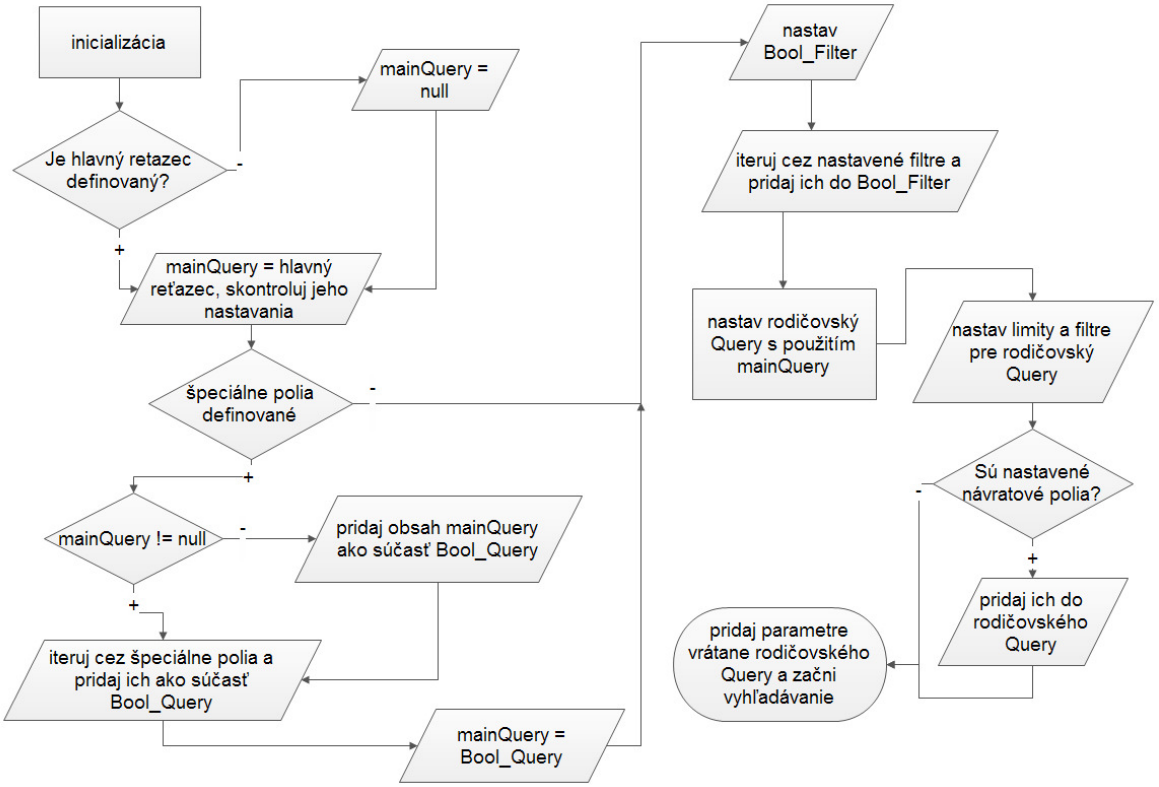
Komunikácia objektov spočíva v tom, že všetky sú obsiahnuté v inicializačnom. Jeho trieda ma definované metódy, ktoré potom jendoducho vracajú konkrétny objekt. Od verzie *PHP* 5 sú objekty vracané metódou vo forme referencie. Toto značne uľahčilo prácu a nebolo treba riešiť dodatočné zápisy. Objekty ďalej neukladajú ako atribút odkaz na niektorý objekt, namiesto toho je potrebné pristupovať k inicializačnému objektu ako ku globálnemu. Takto môžu objekty kedykoľvek komunikovať medzi sebou, napríklad stránkový objekt s databázovým.

2.3.4 Začlenenie *ElasticSearch*

Ako bolo spomínané, interakcia s *ElasticSearch* aktívne využíva možnosti dostupného klienta a ten je „obalený“ do pomocnej triedy. Klient poskytol potrebné primitíva pre budovanie dotazu a pomocná trieda využíva metódy, ktoré tieto primitíva spája a vytvára funkčné dotazy podľa potreby. Niektoré parametre, ktoré sa pre klienta musia dodať, sú požadované vo forme polí a niektoré ako samotné reťazce. Pre uschovanie týchto premenných sa preto celkovo používa pole a metódy obsluhújúce pridávanie alebo menenie parametrov. Parametre nemusia byť všetky dodané. Tie, ktoré nebudú zadané, majú niektoré pôvodné hodnoty nastavené aj samotným *ElasticSearchom*.

Aktívne metódy sú potom celkovo tri, ktoré postačujú pre funkčnosť portálu. Je tu klasická metóda na bežný vyhľadávací dotaz s filtrami a podobnými možnosťami. Táto metóda je použitá všade tam, kde sa jedná o vyhľadanie obecne 1 až n článkov. Ďalšie metódy implementujú podporu pre *More Like This* funkčnosť. *More Like This* ako prístup k dátam v *ElasticSearch* má dve podoby, bližšie popísané v časti 2.3.8. Pre hľadanie textov v prvej metóde sa používa dotaz *query_string*. Jedná sa o formu dotazu, ktorá priamo navädzuje na logiku z *Lucene* ako predchodcu, ktorého syntax v tomto prípade plne využíva.

Použité články sa ukladajú vo forme referencií, sú uložené pomocou položky z poľa *rrsid*. Naviac sú identifikované aj časťou názvu. Nepoužíval som celý názov hlavne kôli možnej dĺžke reťazca, ktorý ho predstavoval. Poskytnutie celého obsahu dokumentu a taktiež aj celého názvu je úlohou náhľadu alebo prezentačnej stránky.



Obrázek 2.8: Algoritmus normálnej činnosti medzivrstvy

Pojednávanie o tom, ako *ElasticSearch* v spolupráci s ostatnými vrstvami pracuje, je uvedené pri konkrétnom popise stránok, kde sa tak deje. *ElasticSearch* dovoľuje použitie faktoru náhodnosti v mnohých jeho dotazoch. Takýto dotaz by umožnoval pridať nastavenú *fuzzy* hodnotu do procesu a výsledky by mohli bývať aj odlišné po každom spustení. V práci som toto neuvažoval, rozhodol som sa použiť metódu bez náhodnosti, kde sa výsledky pri rovnakých dotazoch dokážu opakovať.

ElasticSearch ako každý podobný systém môže vrátiť chyby. Na to bolo prihliadané a podobné stavy ošetrené. Problém, ktorý som cez snahu nedokázal odstrániť, sa objavuje sporadicky. Napríklad, pri dotaze, ktorý zjednocuje viacej položiek, občas dochádza k vypadnutiu niektorého výsledku. Opakované testovanie ukázalo, že práve *ElasticSearch* nedokázal v jednu dobu dodať potrebné výsledky. Riešenie by som videl v rozdelení dotazu do minimálnych celkov, avšak toto by spôsobilo vyššiu časovú aj pamäťovú náročnosť.

Trohu iný chybový stav predstavuje vypršanie časového limitu pri dotazovaní. V podstate takáto chyba môže nastať ako prípad, kedy sa spojenie neukončí a klient čaká na dáta, ktoré neprichádzajú. Takýto takzvaný *timeout* sa vyskytol napríklad počas testovania zadávaním zložitejších dotazov. Riešenie bolo použitie časového limitu v *curl* volaní. Toto následne vyvolá výnimku, ktorá je zachytená. Demonštračne je toto ošetrené na stránke vyhľadávania, kde sa priamo používa v *Ajax* volaní *JSON* formát dát. Tak je možné určiť, či ide o bežné výsledky, alebo o chyby (nie len *timeout*). Timeout je nastavený na hodnotu 300 sekúnd, čo sa môže zdať až príliš, no v prípade potreby sa dá prestaviť v klientovi.

2.3.5 Administrácia

Ako administračný systém, aj tento musel ponúkať vytvorené prostredie, kde by sa bežný užívateľ nedostal. Každý stránkový objekt má atribút, ktorý určuje, či je stránka normálne klasifikovaná ako privátna. Inicializačný objekt toto preveruje a v prípade, že je stránka privátna, volá príslušnú metódu autentizačného objektu. Inicializačný objekt potom vykoná kroky, ktoré ďalej spracujú požiadavku od používateľa – dá mu možnosť sa prihlásiť alebo ho vpustí do privátnej sekcie.

Celý autentizačný proces funguje za použitia malých dátových položiek, ktoré sú prístupné iba pre danú webovú doménu a sú uložené v prehliadači. Jedná sa o *Cookies*, ktoré sú podobne ako *session* používané k tomuto, a iným účelom. Výhodu oproti *session* majú napríklad v tom, že nie je potrebná ďalšia podporná funkcionalita, pre kontrolu expirácie. Niektorý užívatelia webových prehliadačov majú *Cookies* zablokované. Mnohokrát mu to nedovoluje poskytnúť pokročilejšie užívateľské funkcie ako aj celkový pôžitok z návštevy webovej stránky. Toto rovnako súvisí aj s ďalšou okrajovou otázkou. Konkrétne či podporovať úžívateľov s vypnutým *Javascriptom* alebo nie. Taktiež som to do hĺbky neriešil, ale odpoveď sedí aj na predchádzajúci problém. V tejto dobe vnímam ako obmedzenie úžívateľských možností, ak má niekto zakázanú podporu *Javascriptu* alebo *Cookies*. Pri *Javascripte* sa podľa mnohých štatistík jedná momentálne o celkovo len maximálne 2% ľudí a v Európe ešte menej[1]. Je pravda, že v tejto práci *Javascript* využívam pomerne dosť. Jedná sa hlavne o *Ajax* funkcionalitu. Naďalej to spoločne s *Cookies* ostáva iba v administračnej časti, preto sa to bežného návštevníka portálu týkať nebude.

Autentizačný proces pozostáva klasicky zo zadania užívateľského mena a hesla, prípadne zašktnutím políčka pre použitie *Cookie* s neobmedzenou platnosťou. Heslo sa kôli bezpečnosti ukladá v databázi vo forme *md5 hashu* a voči tejto forme sa aj kontroluje. V prípade, že je v *url* dostupný parameter pre preserovanie a prihlasovanie prebehlo úspešne, tak systém presmeruje užívateľa na konkrétnu stránku danú parametrom. Takáto situácia nastane, ak neprihlásený užívateľ žiada privátnu stránku. Systém ho vtedy presmeruje na prihlasovaciu stránku s parametrom ukazujúcim na jeho pôvodne žiadanú stránku. Doplňujúcim prvkom je odhlásenie, ktoré je implementované ako akcia, odkazovaná ako podstránka v administračnej stránke. Pri odhlasovaní sa jednoducho nastaví platnosť *cookie* na dobu minulú, čím sa znevaliduje.

Administrácia ponúka jednoduché rozhranie pre orientáciu v nej – vrchná časť vo forme menu. Menu ponúka 5 položiek so zameraním na:

- vyhľadávanie
- nastavenia
- kategórie
- prispôsobenie vzhľadu
- užívatelov

Každá z týchto položiek je ďalej v texte rozpísaná. Sú reprezentované jedným stránkovým modelom, ktorý obsahuje potrebnú logiku pre vetvenie podľa dodaných premenných. Celkovo tieto stránky slúžia iba na to, aby naplnili databázu dátami, takže s naplnenou databázou by bolo možné pracovať so systémom bez použitia administrácie.

Vyhľadávanie

Stránka s vyhľadávaním obsahuje formulár, ktorý slúži na vyhľadanie článkov. Správca tak zadá časť textu, ktorý je spojený s článkom a systém mu vyhľadá potrebné články. Toto rozhranie je doplnené o prvky, ktoré je možné vidieť aj v iných vyhľadávacích portáloch. Jedná sa o automatické ponúkanie dokončených výrazov alebo pokročilé vyhľadávanie.

Automatické ponúkanie výrazov nedopĺňa len koniec slov. Zobrazuje celé názvy článkov v príležitostne schovávacom sa boxe. Tento počet je pevne daný v kóde a je nastavený na hodnotu, ktorá sa zdala byť optimálna. Po kliknutí na takúto položku sa názov ponúknutého článku doplní do vyhľádávacieho poľa a pre väčší komfort sa hneď odštartuje vyhľadávanie. Pri implementácií tejto funkcionality som sa pokúšal rozumne vyriešit problém, ktorý vracal v doplňovacom boxe len výsledky z poľa nadpisu (v originále *title*). Nešlo o technický problém ale skôr o praktický. Nakoniec sa toto podľa mojích preferencí ukázalo ako plne postačujúce a nebolo teda nutné upravovať doplňovanie s pomocou dát z ostatných polí.

Pokročilé vyhľadávanie ponúka možnosť vyhľadávať výraz v určitom poli. Pôvodne bola integrovaná aj podpora pre kontrolu a spracovanie podporných informácií zadaných do vyhľadávacieho poľa. Tieto informácie mali jednoduchú špecifickú syntax a ďalej upresňovali vyhľadávanie. Napríklad bolo možné použiť rôzne filtre alebo vyhľadať jeden podvýraz v inom poli ako druhý podvýraz, ktoré bolo k dispozícií aj v prototype. Z hľadiska toho, čo by teoretický správca naozaj využil, bolo toto rozšírenie odstránené a ostala možnosť len špecifikovať celé vyhľadávacie polia pre celý výraz. Spracovanie vstupného poľa dovoluje použitie sekvencií „AND“, „OR“, prípadne aj znak „*“ pre doplnenie 0 až n ľubovolných znakov. *ElasticSearch* vyhľadáva články, ktoré občas nemajú totožný výraz, ako žiadaný. Je to dané vnútornou logikou vyhľadávania, ktorá sa snaží dodať čo najbližšie výsledky. Pri automatickom doplňovaní sa pri vrátených výsledkoch, ktoré obsahujú presnú formu žiadaného výrazu, potrebná časť výsledku zvýrazňuje. Celá stránka je orientovaná na *Javascript* a taktiež aj získavanie výsledkov funguje s pomocou *Ajaxu*. Po vyhľadání je možné zobraziť náhľad po kliknutí na názov článku. Ten sa znova dotiahne *Ajaxom* vo forme zvýrazneného rámu nad ostatným obsahom. Pokročilejšie riešenie ponúkajú rôzne rozšírenia *Javasriptovej* knižnice *jQuery* pre takzvaný *lightbox*. Článok je taktiež možné rovno pridať do niektorej kategórie, podľa starého označenia do sekcie 1.

Ako bolo spomenute v časti 2.3.2, existujú v štruktúre isté obmedzenia. Napríklad obmedzenie, aby stránka mala unikátny názov. Toto nariadenie podporuje fakt, že sa neočakáva rovnaký článok na viacerých miestach na prezentačnej stránke. Je zabezpecené taktiež, aby podobná situácia nemohla nastať pri hľadaní článku, ktorý je uložený ako návrh. Alternatívne riešenie v oboch prípadoch by mohol predtavovať mechanizmus pridania článku po jeho odstránení zo štruktúr, kde bol pôvodne umiestnený. Tento spôsob je avšak žiadúci možno len pri návrhoch a aj to je vec názoru. Vybral som si možnosť prvú. Pretože mi príde intuitívnejšia.

Niektoré dokumenty, ktoré sa objavujú v *ElasticSearch* nemajú kompletné dáta. Niektoré napríklad neobsahujú vôbec textové pole, prípadne abstrakt. Systém dovoľuje pridávanie aj takýchto článkov, no taktiež je možné nastaviť, aby systém pracoval iba s vybranými poliami, ktoré si zvolené, aby aspoň nejaké dáta obsahovali. Bolo nutné zaistiť, aby sa pri nastaveniach, ktoré požadujú mať zadané polia v dokumente, boli správne nastavené filtre. Pri normálnom *API* na hľadanie to nebol problém. Pri *mlt API* toto nie je priamo dovolené, bolo treba nutné použiť iný spôsob, ktorý vyhľadáva podobnosť v poliach, ktoré majú byť neprázdne. Keďže hľadanie výrazu v prázdnom poli skončí neúspechom, bolo toto riešenie postačujúce. Naďalej je ale sytém prispôsobený pre použitie existujúcich polí.

Nastavenia

Administrácia ponúka niekoľko možností pre nastavenie chovania systému. Na tejto stránke sa dá nájsť nastavenie pre *ElasticSearch*, ako aj globálne nastavenia portálu. Všetky nastavenia čerpajú svoj pôvodný obsah z konfiguračného súboru, kde sú uložené pod konštantami začínajúcimi *prefixom* „RG5_DEF“. To označuje, že takáto hodnota je použitá, ak v databázi ešte nie je použitá iná, novšia. Celé nastavenia sú v rámci jedného formulára. Jeho odoslanie spôsobí teda uloženie všetkých hodnôt. Tieto hodnoty sa uložoa do tabuľky *configs*. Databáza potom v bežnom fungovaní vytvára s konštantami v konfiguračnom súbore spôsob ako sa dostat k aktuálnym nastaveniam.

Kategórie

Pred tým, než správca môže pridávať články, je potrebné mať vytvorenú kategóriu, do ktorej bude článok pridelený. Vytváranie kategórie zahŕňa vyplnenie niekoľkých polí vo formulári. Ide tak o názov kategórie, kľúčové slová, a nastavenia pre plánovanie. Názov slúži na pohodlnejšiu identifikáciu. Kľúčové slová majú svoj význam pri vyhľadávaní relevantných článkov. Táto funkcia je popísaná v ďalšej časti 2.3.8. Ostatné polia vo formulári slúžia pre nastavenie intervalu a dátum poslednej aktualizácie kategórie.

Nastavenie kategórie taktiež umožňuje prezeranie pridaných článkov s možnosťou článok odstrániť, prípadne ho pridať medzi vzory, podľa ktorých bude prebiehať hladanie relevantných článkov.

Taktiež je tu zoznam vyhľadaných článkov, ktoré čakajú na odmietnutie alebo potvrdenie. Ak správca článok odmietne, článok ostané schovaný v tabuľke *suggestions* s nastavenou hodnotou, značiacou odmietnutie. Tieto články sa bežne ďalej neobjavia v zozname ponúknutých. Ak správca potvrdí článok, tak sa navrhnutý článok odstráni celý z tabuľky *suggestions* a pridá sa ako normálny článok.

Pre dodatočnú podporu boli implementované akcie pre možnosti si vynútiť aktualizáciu, prípadne odstrániť záznamy z tabuľky návrhov príslušiacich k danej kategórií. Toto sa hodí, ak sa správca rozhodne akceptovať pri aktualizácií članky, ktoré boli predtým zamietnuté.

Prispôsobenie vzhľadu

Umožnuje meniť vzhľad na prezentačnej stránke. Každá kategória, podobne ako ostatné stránky, má pridelenú šablónu. Kategórie ju dovoľujú meniť priamo v systéme. Pre jednoduchosť sa skenuje príslušný priečinok a hľadajú sa skripty s koncovkou „_tpl.php“. Ostatné je v réžií samotnej šablóny, či si importuje iné súbory. Do oblasti prispôsobenia vzhľadu som začlenil aj moderáciu komentárov. Je možné prezerať komentáre a riešiť ich povolenie alebo zmazanie. Funkčnosť samotných komentárov je riešená v samotej sekcií 2.3.7.

Pre viac možností je dovolené nastavovať priamo *status* stránky. Hodnoty, ktoré sa používajú sú „active“ a „inactive“. Takýto prístup je možné zohľadniť v prezentácií. Konkrétne to znamená, že inaktívna stránka sa chová, ako keby pre návštevníka neexistovala. Platí to pre kategórie a aj pre články. Vnútorne však existujú, je možné k nim pristupovať, meniť ich, prípadne spúštať aktualizácie. Neaktívna kategória sa nezobrazí v zozname a v prípade, že sa jedná o kategóriu, ktorá je zvolená ako hlavná (*homepage*), to systém vyhodnotí, ako keby ani neexistovala. To následne zohľadní pri výpise chyby.

2.3.6 Užívatelia

Systém dovoľuje taktiež pridávať nových užívateľov a spravovať ich. Atribúty, ktoré takýto užívateľ má, sú meno, heslo, email a užívateľský úroveň, prípadne dátum vytvorenia. Z dôvodu bezpečnosti sa heslo ukladá vo formáte *md5 hashu*. Prítomnosť užívateľských úrovní definuje role užívateľov. Sú celkovo tri:

- Administrátor
- Editor
- Prispievateľ

Každú činnosť užívateľa je jednoduché kontrolovať získaním hodnoty jeho úrovne a porovnanie v podmienke. V administrácií je takto zakomponované pravidlo, že jedine administrátor môže vytvárať a mazať užívateľov. Druhé pravidlo dovoľuje upravovať nastavenia systému. K tomuto má prístup ako administrátor, tak aj editor. Prispievateľ je určený hlavne na správu obsahu a to zahŕňa aj vyhľadávanie.

2.3.7 Komentáre

Každý článok ponúka možnosť k nemu pridávať komentáre. Implementovaná je podpora jednoduchých komentárov bez hierarchie. Pridávať komentáre môže ľubovolná osoba a to z pohľadu bezpečnosti vytvára problém. Pre takýto prípad je možnosť si vybrať, či budú komentáre moderované alebo nie. Moderovaním sa rozumie potvrdzovanie nejakým správcom pred tým, ako bude komentár vidieť aj na prezentačnej stránke. Toto riešenie nemusí byť obzvlášť pohodlné, preto je vytvorený prvok, ktorý sa stará o použitie *Captchi*. *Captcha* je v skratke nástroj, ktorý dokáže odlíšiť človeka a počítač, napríklad potrebou opísať text z obrázka a dokáže zrušiť funkčnosť robotov, ktorý často pridávajú do diskuzií pripravený obsah. Moja *Captcha* zadá pre užívateľa jednoduchý aritmetický príklad a to často aj postačí. Pri spracovaní výsledku sa musí niekde zobrať správny výsledok. To je vyriešené pomocou skrytého poľa, ktorý obsahuje hodnotu, získanú kombináciou tajného reťazca z konfiguračného súboru, dátumu a správneho výsledku. Validácia výsledku teda len znova prevedie užívateľom zadaný výsledok a skontroluje, či sa zhoduje s hodnotou zo skrytého poľa.

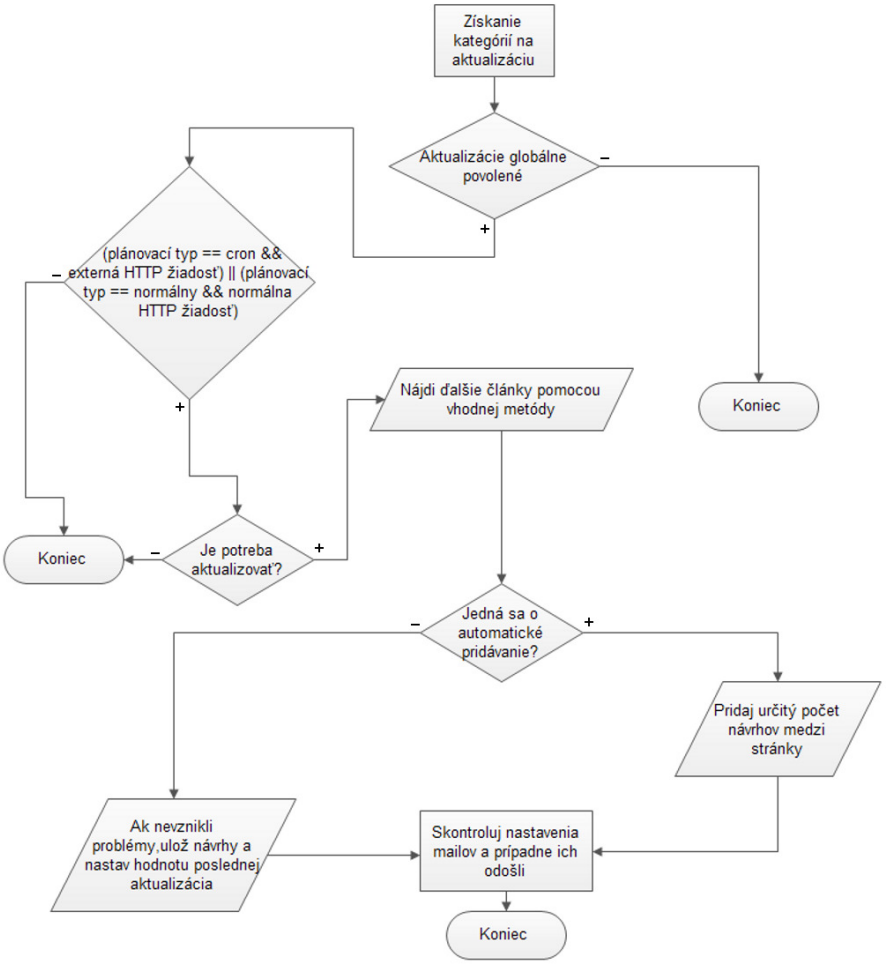
Práve kôli použitému dátumu v kontrolnom reťazci, by bol problém, ak by sa stránka načítala pred polnocou a spracovanie komentára by prebehlo po nej. Preto sa v prípade nezhody použije ešte druhý dátum (v tomto prípade už „všerajší“) a až po ďalšom neúspechu sa vráti správne určená chyba. Kontrola dát prebieha v rámci *Ajax* žiadosti a následne vhodne vracia text označujúci prípadnú chybu, alebo nový pridaný komentár.

2.3.8 Vyhľadávanie relevantných článkov

Relevantné články v problematike tejto práce predstavovali články, ktoré mali niečo spoločné s danou problematikou. Problematika môže byť popísaná textovo vo forme kľúčových slov alebo môže zahŕňať celú množinu už pridaných článkov. V prípade kategórie, ktorá by mala článkov rádovo stoviek sa jedná problém s vyťažením. Snažil som sa tento problém obmedziť a ako dobrá možnosť sa ukázala klasifikovanie ľubovolných článkov ako takzvané „vzory“. Podľa príslušných nastavení sa následne používajú vzory alebo kľučové slová pre vyhľadávanie podobnosti.

Vyhľadávanie sa môže spustiť v rámci plánovanej akcie alebo čisto manuálne. Každá kategória má pre plánovanie príslušné hodnoty v databáze, ktoré označujú dátum poslednej aktualizácie, interval aktualizácií v dňoch a hodnotu, ktorá označuje, či je plánovanie povolené. Ďalšie možnosti sú v stránke s nastaveniami. Konkrétne je možnosť si vybrať, či bude zapnuté plánovanie, odosielanie *emailov* pre správcov (volitelní správcovia), automatické pridávanie bez manuálneho zásahu alebo či bude kontrolované a následné spúšťané pri každej *HTTP* žiadosti, alebo iba pri externej žiadosti. Externá žiadosť je taký typ, ktorý sa dá spúšťať „zvonku“. Na toto je pripravvený prístupový bod v zložke *cron*. Pre správnu funkcionalitu stačí nastaviť správne práva a umožniť prístup, napríklad pre nástroj s rovnakým menom v *Linuxe* (a iných OS). Takýto princíp simuluje normálnu žiadosť, avšak vykoná iba aktualizáciu bez výpisu stránky.

Bohužial nie je na testovacom počítači odosielanie *emailov* povolené, preto táto súčasť ostáva len ukrytá. Testovanie kôli tomuto muselo prebehnúť na inej doméne (samozrejme bez podpory o strany *ElasticSearch*). Každá úspešna aktualizácia kategórie nastaví taktiež aj hodnotu poslednej aktualizácie na aktuálny dátum. Úspešná znamená, že sa nevyskytol problém a našiel sa aspoň jeden článok. S tým je spojené taktiež odosielanie mailov zvoleným správcom.



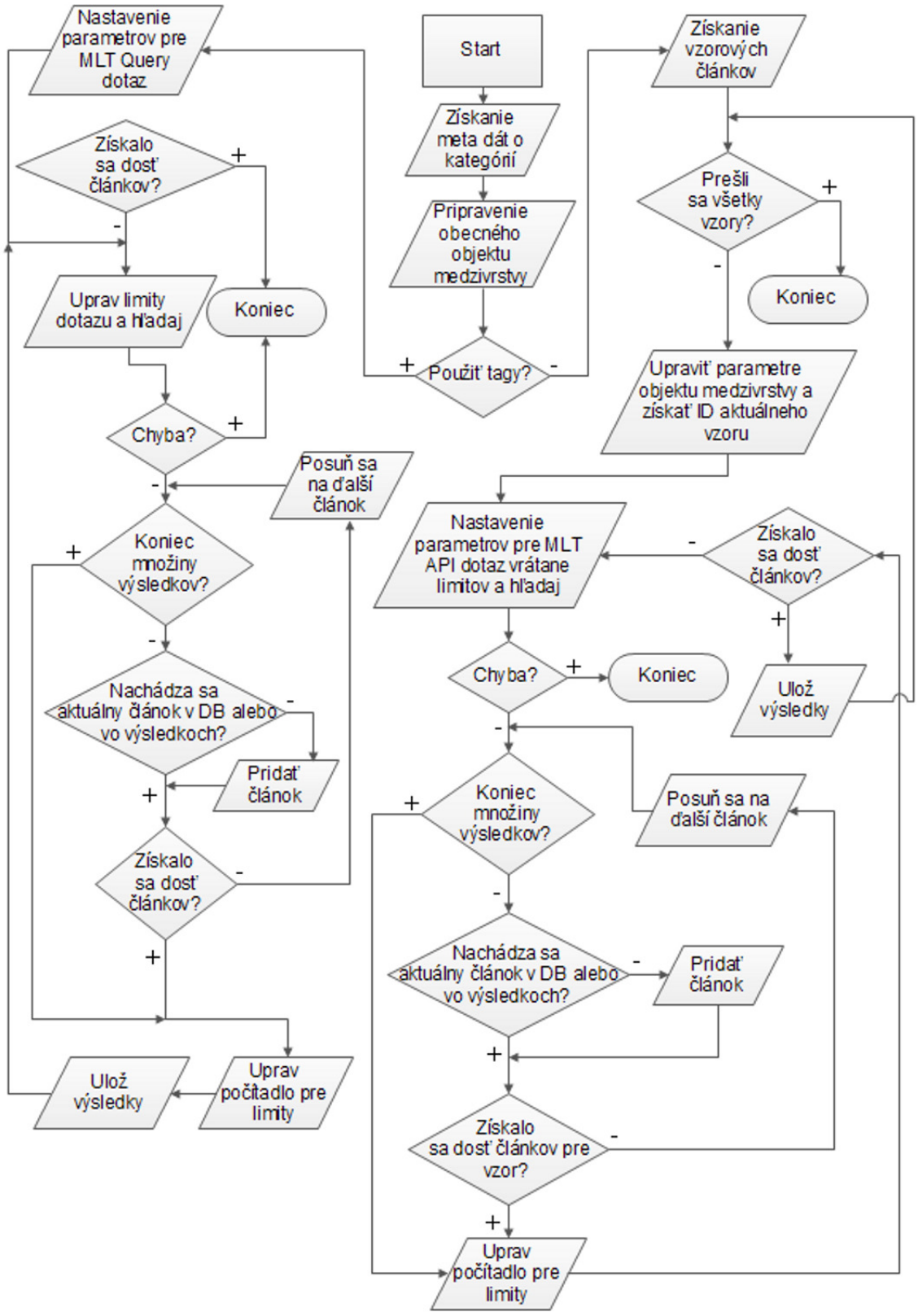
Obrázek 2.9: Zjednodušený pohľad na plánovací algoritmus

Uvedený zjednodušený algoritmus nepopisuje samotný priebeh získavania článkov. Získavanie článkov na základe podobnosti je vec nastavenia a potom sa člení na dva typy – *mlt API* a *mlt Query*. Prvý typ ponúka iné rozhranie ako bežné vyhľadávanie, ktoré používam (2.8) ale poskytuje naďalej podobné možnosti a obmedzenia. Kľúčovým prvkom tohoto typu je poznať *ID* dokumentu. Nejedná sa ale o *ID*, ktoré sa nachádza v poli, ktoré je súčasť dokumentov, ale celkové pridelené *ID*. Rozdiel medzi nimi je v tom, že pole *rrsid* predstavuje parameter jedného dokumentu a to by malo byť rovnaké stále, a *ID* dokumentu je pridelený reťazec znakov, ktorý pred indexáciou nie je známy. Toto predstavuje dôvod, prečo databáza neukladá *ID* dokumentu, ale práve hodnotu z poľa *rrsid* (2.3.4). Pre *mlt API* sa *ID* dá získať normálnym vyhľadávaním, nakoľko je súčasťou množiny vrátených hodnôt. Potom ostáva vhodne zostaviť dotaz a odoslať do *ElasticSearchu*.

Druhý typ vyhľadáva na základe podobnosti vo výraze, ktorý je súčasťou dokumentu. Nie je potrebné poznať *ID*, jedine potrebný výraz. Tento implementuje veľmi podobný prístup ako normálne vyhľadávanie v medzivrste. Jediný rozdiel, ktorý vyčnieva je použitie *More Like This Query* namiesto *Quest String* dotazu. Reťazec pre tento typo dotazu sa bere z vlastnosti *tagov*. Tieto *tagy* môžu byť zapísané v ľubovoľnej forme, *ElasticSearch* bude vyhľadávať podľa celého použitého zápisu.

Z dôvodu praktickosti boli tieto dve metódy oddelené. V podstate samotnej interpretácie týchto dotazov nejde o veľký rozdiel. *mlt API* sa počas spracovania rozvinie do *mlt Query* dotazu.

Problém, ktorý sa dokáže pri takýchto dotazoch objaviť, je nastavenie parametrov pre celú problematiku *More Like This*. Ide o parametre *min_doc_freq* a *min_term_freq*. Tieto dva parametre dokážu vypustiť pojmy alebo slová, ktoré sa nebudú v dokumentoch v istej frekvencií vyskytovať. Efektívna hodnota je 1 pre oba parametre, čo som zohľadnil aj v konfiguračnom súbore. Pri testovaní aj priamo cez *SSH* spojenie k počítaču sa mi nepodarilo okrem malých zmien v hodnotách získať žiadne výsledky. Toto chovanie ale možno bude v určitých prípadoch žiadúce, keďže správca nemusí chcieť za každú cenu pri aktualizácií nájsť nové články. Preto je možnosť hodnoty zmeniť dodatočne aj v administrácií.



Obrázek 2.10: Oba typy *More Like This* dotazov

2.3.9 Inštalácia

Z dôvodu uľacenia prípadného nového nasadenia je k dispozícií administračný modul, ktorý poskytuje možnosť manipulácie s tabuľkami v databáze. Dá sa naň nahliadať ako na stránku, ktorej účel mal byť od začiatku oddelený od hlavnej administrácie. Vznikol pôvodne kôli nahradeniu nástroja *phpMyAdmin* a ponúkal hlavne vývojárske funkcie, ako zobrazenie tabuliek alebo mazanie konkrétnych dát. To čo ostalo v konečnej verzií je možnosť odinštalácie a inštaláce tabuliek, ako aj *export* a *Import* dát. Súčasťou procesu inštalácie je taktiež proces vytvorenia prvého užívateľa. Informácie o novom užívateľovi sú jediné dáta, ktoré sa zapíšu do databázy potom, ako sú tabuľky vytvorené. Tento modul je nastavený pre spoluprácu s objektom databázy. Hlavná činnosť je založená na kontrole, či je databáza v poriadku. Táto kontrola prebieha tak, že sa porovnáva aktuálna štruktúra tabuliek, teda stĺpce v tabuľkách, s pripraveným zápisom v kóde a ten je zapísaný ako pole. Ak je potom databázová štruktúra v poriadku, nová inštalácia je zbytočna, a tým pádom správcovi nie je umožnené vykonať opakovanú inštaláciu. K opakovanej inštalácií môže dôjsť iba po porušení štruktúry, napríklad odinštalovaním.

V súvislosti s modulom inštalácie sa objavila otázka — ako zabrániť prístup osobe bez poverenia a zároveň umožniť prístup správcovi v prípade, že ešte žiadny užívateľ neexistuje (napríklad pri inštalácií nanovo). Kedže každý stránkový objekt, čo je aj inštalácia, obsahuje vlastnosť označujúcu, či je stránka privátna alebo nie, bolo ju vhodné dynamicky meniť podľa štruktúry databázy. Ak nie je štruktúra v poriadku, privátnosť stránkového objektu sa zruší a taktiež naopak. Ak by však nastala taká situácia, že by sa štruktúra porušila, napríklad by z nejakého dôvodu zmizla niektorá tabuľka, tak toto predstavuje isté riziko. Konkrétne také, že by ľubovoľný návštevník, ktorý sa dostane na stránku s inštalačnými nástrojmi, mohol zmazať kompletne celú databázu. Toto je ale nepravdepodobné a ďalej som sa s tým nezaoberal — svojemu účelu to postačuje. V prípade nutnosti ošetrenia takýchto výnimiek ide iba o niekoľko vnorených podmienok.

Export a import

Z dôvodu lepšej prenositeľnosti som začlenil do projektu aj možnosť *importu* a *exportu* dát. Nejedná sa o celkový výber dát, aký ponúka napríklad *phpMyAdmin* vo forme takzvaného *SQL dumpu*, ale o samostatný export tabuliek vo vhodnej forme. Pre takýto export som nahliadal aj tu na dáta v databáze ako na objekty. Nakoniec, aj obslužné funkcie v databázovom objekte, ktoré vracajú dáta, vracajú polia objektov. Tieto objekty potom obsahujú atribúty, ktoré odpovedajú štruktúre tabuliek, z ktorej boli vytiahnuté. Vhodný formát bol preto viacmenej jasný, no pre vhodnejšiu manipuláciu a ďalšie spracovanie sa použila náhrada. Spomínané objekty sa podobne ako bolo spomínané v popise databázy 2.3.2, ľahko dajú transformovať do podoby reťazca v *JSON* zápise. Namiesto zápisu všetkých dát v jedinom objekte, som sa rozhodol vyskúšať čisto sekvenčný prístup zápisu dát do súboru ako aj čítania z neho. Pre jednoduchosť som uvažoval iba jednoduchý textový súbor, rozhodol som sa použiť obyčajný *MIME*[2] typ súboru — *text/plain*. Tento typ je vhodný pre jednoduché texty, ktorým som exportovaný súbor chcel mať.

Zrejmä z dôvodu chýbajúcej podpory pre *MIME type PHP* normálne nastavuje hlavičku pre tento súbor ako implicitnú hodnotu[4]. Táto hodnota sa kôli spomínanému problému nakoniec použila, a taktiež sa aj dodatočne explicitne nastavuje. Potreba vyčistenia *cache* spolu s ďalším nastavením je taktiež dôležitá, hlavne u niektorých prehliadačoch. Pre od-

[4] Application/octet-stream, prípadne Application/octetstream

lišnosť oproti iným formátom súborov som použil vlastný, hoci sa stále jedná o obyčajný textový súbor s postupnosťou znakov. Formát takéhoto súboru som nazval príznačne podľa formy *JSON* a *dump*(výpis), preto teda „.jsdu“. Štruktúra takéhoto súboru je definovaná ako postupnosť objektov v *JSON* zápise, ohraničených znakmi „\r\n“. Tieto znaky sú normálne určené pre označenie prechodu na nový riadok. Toto následne podporuje *import* dát zo súboru, sekvenčným načítaním jednotlivých riadkov. Pre ďalšiu funkcionalitu pri *importe* sa používa jednoduchý *stavový automat*.

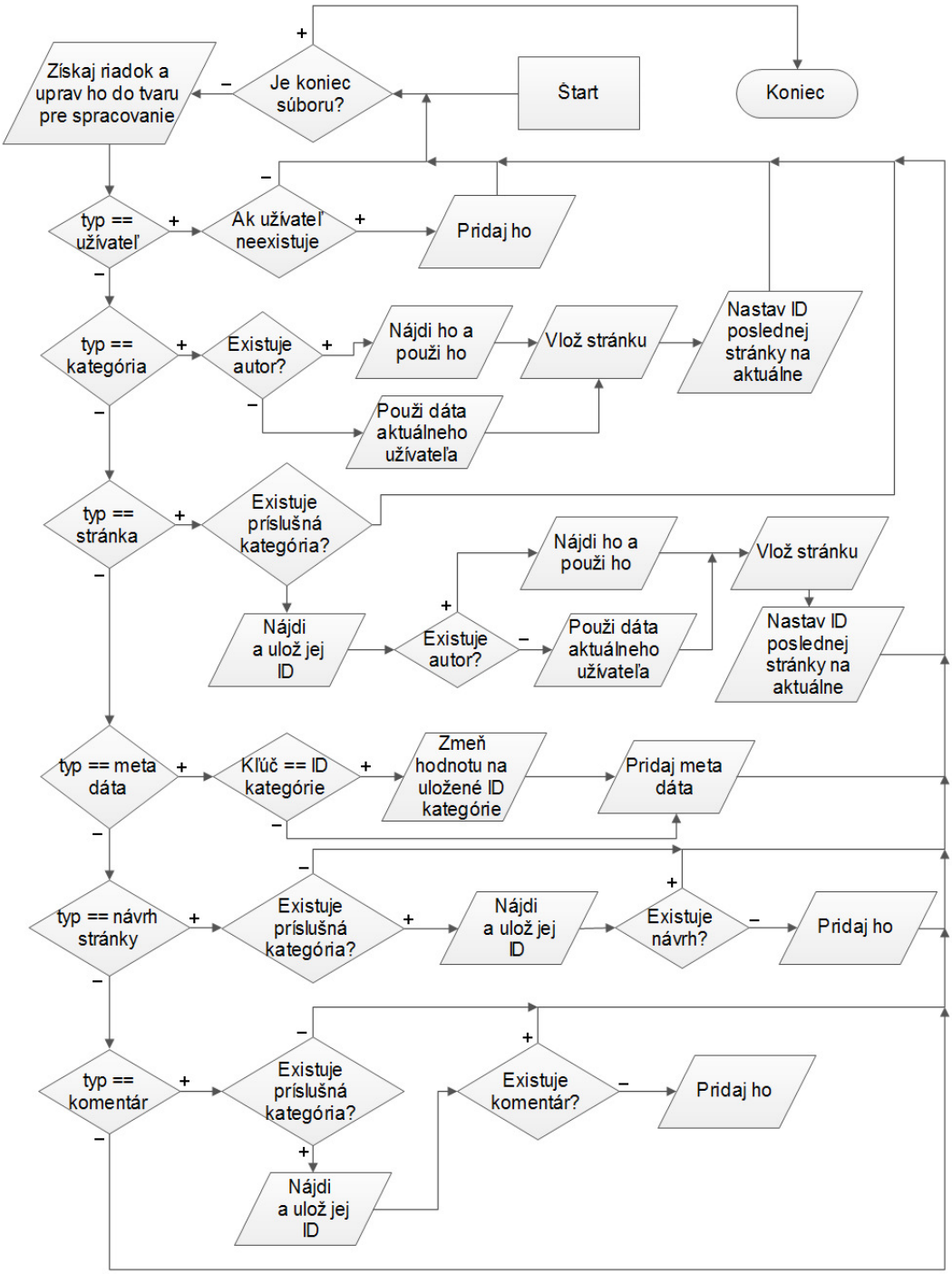
Pre indikáciu chýb sa kumulatívne logicky zhŕňa logický výsledok súčasnej chyby a výsledok aktuálnej operácie. Pre bezchybový výsledný stav importu je potrebné, aby logický výsledok každej operácie mal pravdivú hodnotu („true“). Výsledok *importu* sa po ukončení operácie zobrazí vo forme správy. Vznik chyby neznamená automaticky prerušenie ďalšieho *importu*, iba prerušenie *importu* aktuálneho objektu. Takýto výsledok môže v istej situácií spôsobiť prerušenie *importu* návädzujúcich objektov. Pre príklad sa pri nepridanej kategórií nepridajú ani odpovedajúce články. Sú tu aj prípady, kedy sa pri zistení, že *import* nezaistí odpovedajúci obsah databázy, ako sa požaduje, nenastaví indikácia chyby. V praxi to potom znamená, že ak sa nenájde skutočný úžívateľ, ktorý vytvoril stránku, tak sa použijú dáta aktuálneho užívateľa. Chybou je taktiež chápaný prípad pridávania existujúcej kategórie. Proces to vyhodnotí ako nedovolené pridanie kategórie, čo vyústi v chybu.

Samotne by však použitie zápisu objektov nebolo dostatočné. Niektoré tabuľky v databázovej štruktúre obsahujú stĺpce, ktoré sú cudzím kľúčom pre inú zabuľku. Následné objekty vytiahnuté z databázy potom neobsahujú potrebné dáta, s ktorými je možné jednoznačne možné určiť referované hodnoty v iných tabuľkách. Ako príklad môže poslúžiť stránka, ktorá má ako atribút s hodnotou *ID* autora. Mohla by nastať situácia, pri ktorej užívateľ neexistuje, prípadne existuje pod iným *ID*. Výsledný *import* by potom vyústil vo výsledok, ktorý by sa nezhodoval s predlohou, ktorú popiuje *export*. Kedže užívatelia sú, podobne ako stránky, samostatné objekty z abstraktného pohľadu na databázu, dá sa to odstrániť zmenou štruktúry databázy. Takéto riešenie by bolo ale nekorektné, mohlo by porušiť normalizovanú formu a taktiež by mohlo spôsobiť ďalšie problémy. Ako riešenie som zvolil pridanie ďalších atribútov do exitujúcich exporovaných objektov, ktoré uľahčujú prepojenie na referované položky. V prípade spomínanej kategória a jej autora, sa do objektu s kategóriou pridáva meno autora. Štruktúra tabuľky pre užívateľov nedovoluje duplicitnú existenciu užívateľov s rovnakým menom, preto je dohľadanie potrebného užívateľa jednoduché. Podobný prístup sa používa aj v iných prípadoch. Narastá tak objem dát v súbore, ale je to zanedbateľné.

V prípade meta dát pre stránky sa zvolil trochu odlišný prístup. Každá stránka môže mať ľubovolne veľa popisných dát, ktoré budú prepojené na túto konkrétnu stránku. Namiesto pridávania dát pre špecifikáciu potrebnej stránky sa používa pravidlo pre zápis dát v súbore, a konkrétne ide o pravidlo, ktoré nariaďuje zápis meta dáta priamo za zápisom stránkového objektu. V prípade porušenia tohoto pravidla sa musí očakávať podobné chovanie ako v iných systémoch, kde sa poruší predpísaná štruktúra súboru, určenom sa spracovanie. Meta dáta sa v takomto prípade proste neuložia a nastaví sa logická hodnota pre indikáciu chyby.

Stránka pre *export* obsahuje prepínače, ktoré špecifikujú výstup. Je možné teda zahrnúť exportovanie návrhov článkov, užívateľov a tak podobne. Princíp exportovania je potom len kontrolovanie týchto prepínačov, načítanie dát z databázy a vhodne ich doplniť ďalšími dátami. Poradie v ktorom sa kontrolujú prepínače, je dané štruktúrou databázy. Ak jedna tabuľka obsahuje vzdialený kľúc do inej tabuľky, tak odkazovaná tabuľka musí byť prvá.

Import, ako bolo spomenuté je riešený len kontrolou súboru na vhodný formát a ďalej sa jedná o jednoduchý stavový automat.



Obrázek 2.11: Algoritmus *importu* dát

2.3.10 Podporná funkcionalita

Okrem rôznych zápisov tried, ktoré pôsobia ako objekty systému z pohľadu architektúry, je tu aj pomocný skript, obsahujúci funkcie, ktoré príkladne uľahčujú zápis tým, že sú

určené iba na jednu činnosť a ich telo obsahuje pevnú sekvenciu príkazov. Ďalej sú tu aj funkcie, ktoré sú natoľko obecné, že by nemali byť implementované ako nejaká metóda konkrétnej triedy. Má to výhody, ak je niektorá funkcionalita v pôvodnom znení volaná ako sled niekoľkých metód, tak s použítím tohoto prístupu sa použije zápis kratší a aj lahší na zapametanie a navyše tieto funkcie pôsobia ako globálne. Je teda možné ich volať odvšadiaľ. Nachádza sa tu konkrétne napríklad funkcia na obsluhu vzniknutej chyby. Nejedná sa o konštrukciu typu *try – catch*, ide o spôsob obsluhy chyby pri očakávaných udalostiach. V inicializačnom súbore sa tento pomocný súbor klasicky iba prikladá k spracovaniu.

2.3.11 Jazyková podpora

Ďalšou funkčnosťou, ktoré často *CMS* systémy oplývajú, je podpora viacerých jazykov, či už v administrácií alebo aj ako preklad samotných textov v článku. Do mojej práce som integroval podobnú podporu. Ide iba o normálny zápis v *PHP* skriptoch, zatiaľ čo možnú alternatívu predstavuje napríklad *gettext*, ktorý ako lokalizačný nástroj je dostupný aj v php v podobe knižnice. Tieto skripty, ktoré používam, sú uložené v zložke *lang* a každý z nich je určený pre jednu lokalizáciu. Každý ďalej obsahuje rovnaký zápis funkcie so statickým polom s dvojicami výrazov. Tieto výrazy sa viažu na vzorový kľúč, ktorý je normálne aj výrazom v pôvodnej lokalizácií – tou je angličtina.Ak sa nájde položka pod žiadaným kľúčom, vráti sa preložený výraz. Ak nie, tak podporná funkcia vráti pôvodný reťazec. Ako ďalšie podporované jazyky som integroval podporu pre češtinu a slovenčinu. Jazyková podpora neobsahuje žiadnu pokročilu logikú, ktorá sa dá nájsť v rôznych prekladačoch. Vzorová lokalizácia je priamo vpísaná do kódu. Pre identifikáciu tejto pôvodnej lokalizácie sa používa hodnota v konfiguračnom súbore. V administrácií je možné vybrať inú lokalizáciu zo zoznamu dostupných, vrátane pôvodného z konfiguračného súboru.

Obsluha jazykov je integrovaná ako jedna funkcia, ktorá prijíma ako vstupný parameter reťazec na prípadne preloženie. Kedže pri pôvodnej lokalizácií sa nepoužíva žiadný lokalizačný súbor a tieto súbory obsahujú jednu rovnakú funkciu, je možné skombinovať kontrolu absencie potrebného súboru s kontrolou nastavenia pôvodnej lokalizácie. Zhodný výstup potom nastane v prípade neexistujúcej a pôvodnej lokalizácie.

2.3.12 Prezentačná stránka

Pre prezentáciu bola vytvorená jednoduchá šablóna bez prvkov, ktoré sa dajú nájsť na reálnych webových stránkach. Takéto prvky sú orientované mnohokrát na zdieľanie na sociálnych sieťach, prípadne stiahnutie ako dokument (*pdf*). Stránky, kde sú články všeobecne publikované, sú obsahovo plnšie, často obsahujú stránky priamo nesúvisiace so zámerom webu. V šablóne, ktorú používam je jednoduché logo, zoznam kategórií a samotný obsah, či už vo forme náhľadu na články alebo text samotného článku s komentármi.

Ako bolo spomínané v časti 2.1.1, text článku je uložený v poli ako jeden veľký reťazec. Prezentácia takéhoto textu aj cez vhodné typografické úpravy (písmo, rozostupy medzi riadkami [3]) je ručená jednoliatym textom. Pre trochu lepšiu orientáciu môže pomôcť delenie textu do blokov vytváraním odstavcov. Bez hlbšej známosti obsahu článku sa obecne dá v administrácií nastaviť vytváranie takýchto blokov. Zosníma sa koniec riadkov a doplní sa zápisom pre ukončenie riadku. Je taktiež možné nastaviť počet pázdnych riadkov, čo simuluje odsadenie.

Jedno ďalšie nastavenie zaoberajúce sa prezentáciou umožnuje, aby sa na náhľade článkov zobrazovali iba názvy alebo názvy súčasne s krátkym textom z obsahu článku (dané štruktúrou databázy – stĺpec pre tento text dovoľuje uložit 100 znakov). V prípade nutnosti použiť iba dáta určené pre prezentáciu, ale použiť ich niekde inde, je dostupná metóda v inicializačnej triede, ktorá nespúšťa vypísanie obsahu, ale vrácá dáta, ktoré by inak použila šablóna, ako návratovú hodnotu. Takáto návratová hodnota má podobu poľa, takže je následne jednoduché z neho vybrať potrebné dáta. Tým je zabezpečené prípadné použitie práce ako jeden samostatný modul v inom systéme.

Šablóna je akýsi podklad, do ktorej sa vpíšu potrebné dáta. Prakticky je to súčasť stránkového objektu. Má teda prístup k všetkým atribútom, ktorý daný objekt má. Pre prístup k nim sa používa obslužná funkcia, ktorá v prípade, že je pod daným kľúčom uložená informácia, ju vráti a v opačnom prípade vráti prázdny reťazec. To znamená, že v prípade nekompatibilnej schémy sa nedefinované kľúče nezobrazia a tie, ktoré sú definované mimo požiadavkov šablóny budú jednoducho ignorované. Je taktiež možné šablónu kedykoľvek nahradiť, napríklad v rámci splnenia podmienok, inak je nastavená priamo v konštruktore stránkového objektu.

Kapitola 3

Záver

3.1 Dosiahnuté výsledky

Systém bol vytvorený s použitím skriptovacieho jazyka *PHP*, verzie 5.3.10, databázy *MySQL*, verzie 5.5.24 a *search enginu ElasticSearch*, verzie 0.19.11. Využil som poznatky, ktoré boli k dispozícií, pre vytvorenie plne funkčného administračného systému. Pre vyhľadávanie článkov bol použitý nástroj *ElasticSearch* v kombinácií s *PHP* klientom *Elastica*, ktorý som vhodne zakomponoval do formy systémového objektu. Prezentačná časť, ako *frontend* je základná a venuje sa čisto iba zobrazovaniu článkov s prídavkom písať komentáre. Práca je doplnená rôznymi voliteľnými prídavkami, v neposlednom rade *exportom* a *importom* dát. Toto umožnuje prenositeľnosť na iný server. Vytvorený systém ponúka podporu pre vytvorenie oblastí výskumu a následne jeho automatických aktualizácií. Doplnený je o rôzne nastavenia, ktoré jednak dovoľujú celý portál personalizovať a za ďalšie rôzne manipulovať s dátami, ktoré využije na prezentačnej stránke. Systém je navrhnutý, ako samostatný celok, ktorého architektúra dovoľuje rozšíriteľnosť, prípadne vstavanie do existujúceho riešenia. Toto podporuje vytvorenie frameworku pre ďalší rozvoj.

Bol odskúšaný na fakultnom testovacom počítači *knot08*. Pri tejto konfigurácií práca spĺňala požiadavky a chovala sa podľa predpísaných pravidiel, ktoré boli stanovené pri jej implementácií a taktiež splnila požiadavky plynúce zo zadania práce. Pre testovanie boli pripravené vhodné dáta v niekoľkých kategóriách spadajucich do rôznych oborov výskumu. Testovanie funkčnosti plánovania malo jednoduchý charakter — vyskúšat rôzne metódy hľadania relevantných článkov s rôznymi prístupmi. Išlo o preverenie či systém hľadá správne aj pri externej žiadosti a aj pri normálnej aktualizácií spustenej po uplynutí doby intervalu. Súčasťou testovania bolo taktiež overenie stability systému pri rôznych situáciách, napríklad pri probléme s databázou alebo chýbajúcou šablónou. Idea, aby systém sám dával vedieť správcovi o zmenách bola prakticky vyriešená odoslaním mailu. Nie je teda nutné byť prítomný na portáli pre jeho činnosť. Pri vhodných nastaveniach je možný samostatný chod systému bez vonkajších zásahov. Pri práci sa dbalo na dobré zásady programovania, komentovanie funkcií / metód. Kód je okomentovaný v štýle podobnom pre nástroj *Doxygene*, prípadne iné, ktoré pracujú s podobným zápisom. Neboli použité existujúce *frameworky*, ako napríklad *Zend* alebo *Nette*, ktoré sú známe celosvetovo.

Práca je orientovaná na spracovanie článkov z databázy *ReReSearch* s použitím spomínaného *ElasticSearch*. Téma okolo tejto databázy je pomerne obsiahla a zahŕňa mnohé ďalšie projekty, ktoré pracujú s týmito dátami. Jedná sa tak skôr o celý koncept, ktorý sa snaží vybudovať znalostný systém.

3.1.1 Možné rozšírenia

Táto práca má niektoré spoločné rysy s informačným systémom. Ten, ako taký, máva mnoho smerov, ktorými by mohol vývoj pokračovať. Ako možnosti rozšírenia do budúcnosti by som spomenul napríklad:

- rozšírenie systému na plnohodnotné *CMS*
- rozšírenie systému o podporu editácie dát v *search engine*
- podpora používania viacej *search enginov*
- optimalizáciu zdrojov
- + iné . . .

Samozrejme je možné ísť opačnou cestou a nesnažiť sa daný systém mať ako samostatný, ale upraviť ho do podoby *pluginu*. Keďže mnohé vlastnosti systému sú priamo závislé na implementácií *ElasticSearchu*, moc by som sa s nimi ďalej nezaoberal. Napríklad sa jedná o *More Like This* dotazy, ktoré nemusia ale vždy vracať potrebné dokumenty. Je to ale vec nástroja na správu dokumentov a počíta sa, že ak skutočne nejde o zámer, tak bude jeho vývoj pokračovať.

Prílohy

[A] Manuál
[B] Popis konfiguračného súboru
[C] Popis testovacích dát

Literatúra

[1] Clark, S.: JavaScript Disabled by 1.5% - Yahoo! [online]. http://stevenclark.com.au/2011/02/24/javascript-disabled-by-1-5-yahoo/, 2011-24-02 [cit. 2013-04-10].

[2] N. Borenstein: MIME (Multipurpose Internet Mail Extensions) Part One [online]. http://www.mit.edu/afs/athena/reference/rfc/rfc1521.txt/, September 1993 [cit. 2013-04-22].

[3] Ott, V.: Základní typografické pojmy [online]. http://www.scribus.cz/zakladni-typograficke-pojmy-co-byste-meli-vedet-o-pismu/, 2010-28-02 [cit. 2013-03-22].

[4] Roach, J.: Why mysql_real_escape_string() isn't enough to stop SQL injection attacks! [online]. http://johnroach.info/2011/02/17/why-mysql_real_escape_string-isnt-enough-to-stop-sql-injection-attacks/, 2011-17-02 [cit. 2013-03-02].

[5] WWW stránky: ElasticSearch Documentation [online]. http://www.elasticsearch.org/guide/.

[6] WWW stránky: CMS Architecture in 1.5 and 1.6/ [online]. http://docs.joomla.org/CMS_Architecture_in_1.5_and_1.6/, 2012-24-8 [cit. 2013-05-05].

[7] WWW stránky: Installing Multiple Blogs [online]. http://codex.wordpress.org/Installing_Multiple_Blogs/, 2013-05-04 [cit. 2013-04-22].

[8] WWW stránky: What is inverted index? [online]. http://www.quora.com/Information-Retrieval/What-is-inverted-index/, 2013-22-04 [cit. 2013-05-05].

[9] WWW stránky: API Documentation [online]. http://api.jquery.com/jQuery.ajax/, 2013 [cit. 2013-05-05].

[10] WWW stránky: System Properties Comparison Elasticsearch vs. MySQL [online]. http://db-engines.com/en/system/MySQL%3BSolr/, 2013 [cit. 2013-05-05].

[11] WWW stránky: Elastica API Documentation [online]. http://elastica.io/api/packages/Elastica.html/, Poslední úpravy 2013-24-2.

Príloha A

Manuál

Na priloženom *CD* nosiči je kompletný systém, ktorý stačí vhodne umiestniť do zložky. Systém pobeží aj na *localhoste* v rámci aplikácie, ktorá obsahuje server s podporou, ktorá je špecifikovaná, v časti 3. Predpokladá sa ale testovanie na rovnakom systéme, nakoľko je dostupná databáza a *ElasticSearch*. Tieto dva prvky sú podľa konfiguračného súboru nastavené práve na fakultné počítače*knot08* a *athena2*. Pri použití inej *MySQL* databázy je možné použiť exportované dáta vo forme súboru, ktorý je uložený v príslušnej zložke. Bez *importu* a s novou databázou je nutné spustit taktiež inštaláciu, ku ktorej sa pristupi buď presmerovaním (systém ďalej nepustí) alebo priamo zadaním *url* „/admin?page=install“. Bez inštalácie je dostupný užívateľský účet s údajmi „admin“ a „password“. Administrácia je dostupná cez *url* „/admin“[1]. Uvedené adresy je nutné chápať ako relatívne ku koreňovej zložke.

[1] Je taktiež po nejakú dobu možné používať url http://knot08.fit.vutbr.cz/xmerte01/rrs_gen5

Príloha B

Popis konfiguračného súboru

Konfiguračný súbor obsahuje v prvom rade nastavenia pre pripojenie k databáze. Ďalej tu je sekcia pre konfiguráciu samotného portálu a následne sekcia pre nastavenia *ElasticSearchu*. Položky ktoré obsahujú časť „DEF“, sú určené k zahodeniu a nahradeniu hodnotami v databáze. Jedná sa tak o hodnoty vhodné ako referenčné po inštalácií. Neuvádzam tu niektoré položky, ako napríklad konštanty pre názvy zložiek

Tabuľka položiek pre nastavenie portálu:

RG5_DEF_LANG	Pre lokalizáciu
RG5_DEF_ENABLE_COMMENTS	Povolenie komentárov — ich schovanie
RG5_DEF_NOTIF_ON_UPDATE	Dovoľuje odosielať maily po aktualizácií
RG5_DEF_ARTICLES_SHOW_LIMIT	Počet článkov na stránku v prezentácií
RG5_DEF_AUTO_SEARCH_ENABLE	Povolenie automatických aktualizácií
RG5_DEF_AUTO_SEARCH_TYPE	Dovoľuje samostatne povoľovať návrhy
RG5_DEF_AUTO_SEARCH_PLAN	Typ, s ktorým sa budú vykonávať aktualizácie
RG5_DEF_COMMENT_APPROVING	Možnosť pridávať komentáre bez potvrdenia
RG5_DEF_SPLIT_TEXT	Rozdeľuje prezentačný text do blokov
RG5_DEF_ITEMS_COUNT	Počet položiek na stránke v administrácií
RG5_DEF_SHOW_COMPACT_FORM	Zobrazenie malého náhľadu textu v prezentácií
RG5_DEF_BR_AFTER_SPLIT	Po rozdelení textu do blokov pridá prázdny
RG5_DEF_CATEGORY_TEMPLATE	Implicitná šablóna
RG5_SECURE_HASH	Reťazec pre bezpečnosť použitia *captchi*
RG5_ERRORS	Zapnutie vypisovania varovaní

Tabuľka položiek pre nastavenie *ElasticSearchu*:

RG5_DEF_ES_INDEX	Nastavenie indexu
RG5_DEF_ES_TYPE	Nastavenie typu
RG5_DEF_ES_MUST_HAVE_FIELDS	Polia, ktoré článok musí obsahovať
RG5_DEF_ES_FIELDS_ARRAY	Referencia na všetky dostupné polia
RG5_DEF_ES_RESULTS_COUNT	Pre získanie určitého počtu výsledkov
RG5_DEF_ES_DEF_OPERATOR	Pre rozdelenie slov pri dotazovaní
RG5_DEF_ES_MLT_COUNT	Pre získanie podobných článkov
RG5_DEF_ES_AUTO_UPDATE_COUNT	Pre automatické potvrdzovanie návrhov
RG5_DEF_ES_MLT_MIN_TERM_FREQ	Nastavenie pre parameter do *mlt*
RG5_DEF_ES_MLT_MIN_DOC_FREQ	Nastavenie pre parameter do *mlt*

Príloha C

Popis testovacích dát

Pre demonštráciu bola zvolená kategória s názvom „Engine“. *Tagy*, ktoré som použil pre popis kategórie boli „search, text, data, automatization“. Ostatné atribúty je nepodstatné teraz popisovat, nakoľko som ich v priebehu testovania menil. Nasledovalo pridanie prvotných článkov z administračnej stránky.

- Performance Analysis and Optimization on Lucene
- Integration of a Text Search Engine with a Java Messaging Service

Ako vzor som zvolil prvý článok. Následne som pristúpil k prvej aktualizácií, ktorá bola *vynútená* a použil sa spôsob s *mlt Query*, to znamená použitie *tagov*. Vzniknuté návrhy po prvej aktualizácií:

Milos: A Multimedia Content Management System	Pridané
-3- the relationship of the tables to be more accurate...	-
”Differentiating between data-mining and text...	-
Functionalities of a Content Management System	-
THEORETICAL NOTE	-

Druhá aktualizácia je vo forme externej žiadosti avšak už bez použitia *tagov*.

Navigate and Options	Pridané
Fast Realistic Rendering of Global Worlds using...	-
Optimizing Sorting with Machine Learning Algorithms	-
Design and evaluation of grammar checkers in multiple languages	-
Design of an integrated data retrieval, analysis...	Pridané

Pre testovacie účely bol zmenený vzor na položku „Milos: A Multimedia Content Management System“ a následne sa vykonala tretia aktualizácia vo forme vynútenia bez *tagov*.

Navigate and Options	Pridané
Fast Realistic Rendering of Global Worlds using...	-
Optimizing Sorting with Machine Learning Algorithms	-
Design and evaluation of grammar checkers in multiple languages	-
Design of an integrated data retrieval, analysis...	Pridané

Tento výpis je priamo ako exportovaný súbor pribalený v zdrojových kódoch na CD nosiči.