Univerzita Karlova v Praze

Pedagogická fakulta

Katedra informačních technologií a technické výchovy

BAKALÁŘSKÁ PRÁCE

Vývoj databázového rozhraní pro počítačový slovník

Development of database application for computer glossary

Michal Dudek

Vedoucí práce:	PhDr. Josef Procházka, PhD.
Studijní program:	B7507 Specializace v pedagogice
Studijní obor:	Informační technologie se zaměřením na vzdělávání

2016

Prohlašuji, že jsem bakalářskou práci na téma Vývoj databázového rozhraní pro počítačový slovník vypracoval pod vedením vedoucího práce samostatně za použití uvedených pramenů a literatury. Dále prohlašuji, že tato práce nebyla využita k získání jiného nebo stejného titulu.

V Praze 2016

.....

Michal Dudek

Poděkování

Tímto bych rád poděkoval PhDr. Josefovi Procházkovi, PhD za odborné a vstřícné vedení, připomínky a cenné rady při zpracování tématu bakalářské práce.

Dále bych rád poděkoval své rodině a přítelkyni za trpělivost a významnou podporu při studiu a řešení této práce.

ANOTACE

Obsahem práce je analyzovat a zhodnotit slovníky a slovníkové aplikace z oblasti terminologie informačních technologií. Dle rozboru nejčastějších problémů, které se u počítačových slovníků nebo slovníkových aplikací vyskytují, je navrženo vlastní řešení databázového rozhraní pro počítačový slovník. Práce předkládá modelový postup vývoje databázového rozhraní od analýzy až po ověření vlastní implementace části aplikace s ohledem na využití moderních technologií a metod k vývoji, které jsou v práci popsány. V závěru práce je část implementovaného rozhraní slovníku ověřena pomocí jednotkových a funkčních testů.

KLÍČOVÁ SLOVA

Slovníková aplikace, databázová aplikace, vývoj aplikace, agilní metodika, framework, vývojové prostředí, jednotkové testy, funkční testy.

ANNOTATION

The aim of the thesis is to analyze and evaluate dictionaries and dictionary applications focusing on terminology concernign information and communication technology. Based on the analysis of issues concerning such applications that are currently available, a database dictionary application is developed. The thesis presents the procedure of development all the way from analysis to the final implementation of the application core using current technologies and development paradigms. Final part of the thesis deals with unit and factory acceptance testing of the application.

KEYWORDS

Glossary application, database application, application development, agile methodologies, framework, development environment, unit test, factory acceptance test.

Obsah

Úvod

V této práci se zabývám analýzou, návrhem a vlastní implementací části webového rozhraní pro počítačový slovník. V analýze sleduji parametry, kterými jsou dostupnost slovníku pro uživatele, orientace ve slovníku a způsob vyhledávání, možnost úprav nalezených výrazů, interakce slovníku s uživatelem aj. Po uvedené analýze následuje definování vlastních funkcí a návrh databázového rozhraní pro počítačový slovník. V závěru práce otestuji vlastní část rozhraní pomocí jednotkových a funkčních testů.

Jako součást práce popisuji postup vývoje databázového rozhraní pro počítačový slovník, výběr nástrojů a metod pro samotný vývoj a ukázky řešení problémů, které se při vývoji objevily. Tento popis slouží jako modelový příklad realizace databázové aplikace a případnému čtenáři nabídne souhrn nástrojů, metod a postupů, kterými se může řídit při realizaci vlastní databázové aplikace. Během realizace databázového rozhraní analyzuji dostupnost technologií a nástrojů, které lze pro vývoj rozhraní využít. Při porovnávání technologií a nástrojů zohledňuji současné trendy a možnosti těchto technologií a nástrojů.

Cíl práce

Cílem práce je vytvoření modelového postupu návrhu databázové aplikace včetně dodržení agilních metodik vývoje. Součástí práce je rovněž nezbytná analýza aktuálně dostupných řešení, nasazení vhodné techniky vývoje, realizace a ověření modelového řešení.

1 Analýza existujících řešení počítačových slovníků

Obsahem analýzy je získat přehled dostupných řešení počítačových slovníků, porovnat jejich klady a zápory z pohledu dostupnosti slovníků, obsažených pojmů a jejich výkladu, poskytovaných uživatelských funkcí a metod zadávání obsahu. Tyto parametry jsou stěžejní charakteristikou pro slovníky nebo slovníkové aplikace. Analýzou, ve které sleduji zmíněné parametry, zjistím, zda jsou na českém trhu dostupné počítačové slovníky nebo slovníkové aplikace, které vyhovují sledovaným parametrům. V případě, že takový slovník neexistuje, vytvořím vlastní návrh s řešením nalezených nedostatků vyplývajících z analýzy.

1.1 Vymezení sledovaných parametrů

Dostupnost slovníku

Dostupností slovníku sleduji, zda je možné slovník využít zdarma, nebo za úplatu, je-li slovník přístupný online, jako aplikace spustitelná na PC, či v tištěné podobě, a zda je tištěná podoba k dostání v prodeji.

Obsah odborných pojmů a jejich výkladu

Obsah odborných pojmů a jejich výkladu je klíčový pro zařazení slovníku do analýzy. Počítačové slovníky, které obsahují méně než 100 odborných pojmů a u nichž je výklad většiny pojmů kratší než pět slov, do své analýzy nezahrnuji. U počítačových slovníků zařazených do analýzy sleduji četnost výskytu doprovodného materiálu v podobě audio-vizuálního obsahu.

Uživatelské funkce

Uživatelskými funkcemi sleduji možnost registrace, přihlášení, hodnocení výkladu pojmů, možnost upozornit na nedostatky ve výkladu, způsob vyhledávání a řazení nalezených výsledků.

Metody zadávání obsahu

U online počítačových slovníků zjišťuji, zda mají uživatelé možnost zadávat obsah do slovníku, případně jakými metodami, a jsou-li uživatelé za přispění ohodnoceni.

1.2 Průběh analýzy

Na českém trhu jsem nalezl počítačové slovníky v podobě tištěných slovníků nebo online webových stránek. Počítačový slovník v podobě aplikace spustitelné na počítači jsem nenalezl. Tištěné a elektronické slovníky ve své analýze odděluji do vlastních kapitol, a to z důvodu rozdílného zacházení se slovníkem, vycházejícím z principu těchto slovníků.

1.2.1 Tištěné slovníky

K analýze tištěných počítačových slovníků jsem si pořídil fyzický slovník a na základě jeho prohlédnutí jsem si zapsal výsledky pozorování. Na českém trhu jsem objevil celkem čtyři solidní počítačové slovníky: *Velký počítačový lexikon* (Winkler, 2009), *Slovník počítačových pojmů a zkratek* (Voráček, 1998), *Slovník počítačové informatiky* (Říha, 2002) a *Velký počítačový slovník* (Nádběla, 2006). Tyto slovníky obsahují alespoň základní výklad pojmů a nejsou zaměřeny na překlad do českého jazyka. Počítačové slovníky, které se zaměřují pouze na překlad, jsem opomenul. Získání fyzických slovníků nebylo vůbec jednoduché vzhledem k jejich poslednímu roku vydání. Nejmladší tištěný slovník byl vydán v roce 2009.

Tištěné slovníky jsou z pohledu vývoje informačních technologií zastaralé, přesto z analýzy vycházejí jako jedny z nejlepších zdrojů informací. Obsah těchto slovníků je bohatý, někdy zahrnují až tisíce odborných pojmů s výkladem, a pocházejí od odborníků z praxe (Winkler, 2009), (Nádběla, 2006), což zaručuje výkladům pojmů důvěryhodnost, na rozdíl od zdrojů elektronických, které jsou často spravovány neznámým autorem. Tištěné slovníky procházejí před vydáním pečlivou korekturou, a proto jsou vhodné jako ověřený pramen informací a následné citace.

Nedostatky tištěných slovníků

Mezi zásadní nedostatky tištěných slovníků řadím způsob vyhledávání. Čtenář nemá možnost využít fulltextové vyhledávání, řazení obsahu s využitím filtrů, označení požadovaného obsahu, např. barevným štítkem, a obsah podle tohoto štítku seskupovat, přehrávání audio-vizuálních informací, atd. Současné tištěné počítačové slovníky nenabízí žádnou formu interaktivní spolupráce se čtenářem a práce se slovníky je časově náročnější než u elektronických variant.

Další nevýhodou tištěných slovníků je pořizovací cena, která se běžně na českém trhu pohybuje mezi 150Kč až 700Kč, záleží na konkrétním slovníku a dostupnosti v prodeji.

U tištěných slovníků postrádám i pohotové vyhledání pojmu, protože s tištěnými slovníky se kvůli jejich váze hůře manipuluje, a tudíž nemohu mít slovník vždy u sebe.

Ze své podstaty mají tištěné slovníky nevýhody především:

a) Ve složitém a zdlouhavém vyhledávání informací, bez možnosti filtrování obsahu.
b) V pasivním přístupu ke čtenáři, tj. čtenář nemá možnost interaktivně vstřebat výklad pojmu.
c) V aktualizaci informací. Pro opravu nebo aktualizaci informací je nutné nechat vytisknout slovník nový.
d) V časovém rozkladu a nenávratném poškození tištěného slovníku.
e) V pohotovém využití slovníku, který není vždy dostupný kvůli obtížné manipulaci.

1.2.2 Elektronické slovníky

Na rozdíl od tištěných slovníků je elektronických počítačových slovníků nepřeberné množství. Při hledání na internetu jsem narazil na desítky slovníků, ale jen málo z nich působilo uceleným dojmem. Mnohé z el. slovníků jsou slovníky firemní, doplňující odborný text, který má firma umístěný na svých webových stránkách. Jiné slovníky jsou plné slangů a akronymů. Takové slovníky slouží spíše pro hráčské komunity než ke sledovanému účelu této práce.

Oproti tištěným slovníkům mohou elektronické slovníky nahradit výše zmíněné nevýhody, jimiž jsou interaktivní práce se čtenářem, filtrování obsahu podle vlastních kritérií a audio-vizuální doprovod obsahu.

Hlavní výhodu el. slovníků shledávám ve snazším a rychlejším vyhledávání oproti slovníkům tištěným. V el. slovníku mohu jednoduše aktualizovat nebo upravovat již zveřejněné informace a přidávat odborné pojmy bez nutnosti dalšího vydání nového slovníku. El. slovník mohu zálohovat, a předejít tak trvalému poškození. El. počítačové slovníky z mé analýzy jsou dostupné online a zdarma, což z nich činí lehce dostupný zdroj informací.

Nedostatky el. slovníků

Mezi výrazné nedostatky el. počítačových slovníků řadím validitu informací a gramatickou nekorektnost. Autoři slovníků jsou většinou anonymní, a proto nelze určit, zda jsou to odborníci, kteří znají správný význam odborných pojmů, a zda neudávají špatný nebo

zkreslující význam pojmu. Ve své analýze jsem narazil pouze na jediný el. počítačový slovník, u kterého byl uveden přímo[1] jeho autor.

Souhrnně mezi nedostatky el. slovníků patří:

a) validita informací
b) gramatická a stylistická nekorektnost obsahu
c) nežádaný obsah, který představují reklamy
d) závislost na internetovém připojení

1.3 Závěr analýzy

Nejnadějnějším zástupcem el. slovníků je *IT-Slovník.cz*[2]. Slovník je dostupný online, zdarma a nabízí uživatelské funkce, jako je vyhledávání a možnost vkládání návrhu dalších nenalezených pojmů ve slovníku, a to zadáním názvu pojmu a krátkého popisu do formuláře. Obsah tohoto slovníku se blíží k počtu 2700 slov, což představuje vzhledem k analýze jednu z nejobsáhlejších databází informací v kategorii el. slovníků, viz tabulka č. 1 na následující stránce. Navíc je tento slovník stále aktivní a nové pojmy s každým týdnem přibývají.

Slovník částečně splňuje sledované parametry analýzy, při procházení slovníku jsem však bohužel neobjevil žádný audio-vizuální materiál, který by doplnil výklad pojmu, ani žádnou formu interaktivního obsahu. Zároveň jsem ve slovníku nenalezl možnost seskupování pojmů dle vlastního označení, nebo hodnocení pojmu. Slovník nabízí pouhé čtení výkladu, které je v některých případech velmi stručné a laikovi pojem dostatečně nevysvětlí[3].

Ve své analýze jsem mezi el. počítačovými slovníky nenalezl zástupce, který by splňoval sledované parametry. Autoři el. verzí počítačových slovníků nevyužívají možného potenciálu, aby čtenáři nabídli interaktivní výklad odborných pojmů, nebo výklad doplnili o audio-vizuální obsah. V některých případech nabízí autoři el. počítačových slovníků hodnocení výkladu komentářem. Tento způsob hodnocení se příliš nevyužívá, to potvrzují i současné trendy na sociálních sítích, kde se používá buď hodnocení vyjádřené v podobě ikon palců, srdíček, hvězdiček aj., nebo hodnocení vyjádřené graficky či číselně, případně procentuálně. Žádný ze slovníků z analýzy nenabízí čtenáři možnost nahlásit případný

[1] Slovník počítačové informatiky a sítí, dostupný na adrese: http://www.svetsiti.cz/slovnik.asp.
[2] IT-Slovník.cz, dostupný na adrese: http://it-slovnik.cz.
[3] Pojem „prohlížeč“, dostupný na adrese: http://it-slovnik.cz/pojem/prohlizec

nedostatek ve výkladu pojmů, čímž se autor ochuzuje o zpětnou vazbu a případné nedostatky musí vyhledat sám.

	Název slovníku	Dostupnost	Počet výrazů	Vyhledávání	Uživatelské funkce a interaktivita
Online	Slovník počítačové informatiky a sítí[4]	Online, zdarma, veřejný bez registrace	5300	Přesný výskyt	Čtení, komentáře
	IT-Slovnik.cz	Online, zdarma, veřejný bez registrace	2700	Přesný výskyt, fulltextové	Čtení, komentáře, přidávání dalších návrhů
	Svět Hardware - Slovník základních pojmů	Online, zdarma, veřejný bez registrace	1100	Fulltextové	Čtení
	ABCLinuxu.cz - Výkladový slovník	Online, zdarma, veřejný bez registrace	590	Fulltextové	Čtení
	PlayGate.cz - Slovník počítačových pojmů	Online, zdarma, veřejný bez registrace	350	Fulltextové	Čtení
	Bezpečný internet.cz - Slovník	Online, zdarma, veřejný bez registrace	150	Fulltextové	Čtení
	PC-IN Plzeň - Slovník ICT pojmů	Online, zdarma, veřejný bez registrace	130	Přesný výskyt	Čtení
	WebSnadno - Slovník počítačových pojmů	Online, zdarma, veřejný bez registrace	114	Žádné	Čtení
Tištěné	Velký počítačový lexikon	V prodeji	5000	Abecedně	Čtení
	Velký počítačový slovník	V prodeji	3500	Abecedně	Čtení
	Slovník počítačové informatiky	Omezený prodej	3000	Abecedně	Čtení
	Slovník počítačových pojmů a zkratek	Omezený prodej	1700	Abecedně	Čtení

Tabulka č. 1 – Souhrn analýzy počítačových slovníků, platné k únoru 2016. Zdroj: Autor

[4] Autor tohoto slovníku uvádí, že základ slovníku je převzat z tištěného slovníku *Slovník počítačové informatiky: Výkladový slovník pro práci s informacemi* (Říha, 2002).

2 Definice vlastního řešení slovníkové aplikace

Vycházím z nedostatků, které uvádím v závěru analýzy pro definici vlastních funkcí a návrh slovníkové aplikace dle sledovaných parametrů. Výčet parametrů zároveň využívám jako rozdělení do jednotlivých kapitol, které samostatně popíšu. V každé z těchto kapitol uvádím možnosti uživatelských funkcí vlastní slovníkové aplikace.

2.1.1 Dostupnost slovníku

Z analýzy v tabulce č. 1 vyplývá, že všechny online počítačové slovníky jsou dostupné zdarma a bez nutné registrace uživatele. Vlastní slovníková aplikace bude rovněž veřejně přístupná zdarma, až na výjimku registrace uživatele. Ta je podmíněna využitím některých uživatelských funkcí, při kterých je třeba evidovat uživatelem nastavené hodnoty, např. štítky u vybraných pojmů a vyhledávání pomocí těchto štítků. Registrace bude rovněž volně přístupná.

Textový obsah slovníku bude šířen pod licencí *Creative Commons BY-SA*, která umožňuje uživatelům libovolně nakládat s výkladem u pojmů při uvedení původního zdroje a následném zachování stejné licence. Uživatelé tak mohou obsah slovníku libovolně šířit, upravovat, sdílet aj. *„Licence Creative Commons jsou souborem veřejných licencí, které přinášejí nové možnosti v oblasti publikování autorských děl“*. *„Nejsou popřením klasického pojetí autorského práva, jsou jeho nadstavbou a jako takové vycházejí z občanského zákoníku (§ 2358 – 2389 Zákona č. 89/2012 Sb., občanský zákoník)“*. *„Obliba licencí Creative Commons vychází především z jejich mezinárodní srozumitelnosti“* (Creative Commons Česká Republika, 2016). V dolní části stránky s výkladem pojmu pak uživatelé naleznou vygenerovanou citaci, kterou mohou využít pro správné uvedení zdroje.

2.1.2 Uživatelské funkce

Základní uživatelské funkce

Mezi základní uživatelské funkce řadím registraci a přihlášení uživatele. Bez těchto funkcí nelze provozovat mnoho jiných uživatelských funkcí, jež popisuji v následujících odstavcích. Základní funkcí je i kontaktní formulář, přes který může jakýkoliv uživatel odeslat své sdělení.

Vyhledávání

Nejdůležitější uživatelskou funkcí ve slovnících je vyhledávání. Uživatel má možnost vyhledávat zadáním výrazu do textového pole a při potvrzení výrazu se tento výraz vyhodnotí. Jestliže existuje přesná shoda zadaného výrazu v textovém poli s pojmem ve slovníku, je zobrazena stránka s informacemi o pojmu. V případě, že zadaný výraz má více než jednu přesnou shodu[5], zobrazí se stránka se seznamem pojmů shodného názvu a u každého pojmu krátký popis, podle kterého lze pojem identifikovat. Další možností je fulltextové vyhledávání, které prochází krátkými popisy a výklady pojmů ve slovníku a hledá výskyt zadaného výrazu. Při nalezení jednoho a více výskytů fulltextového vyhledávání je rovněž zobrazen seznam s pojmy a popisem, u nichž byl výraz v popisu nebo výkladu nalezen. Poslední možností je nenalezení žádné shody ani v jedné z předchozích možností, v tomto případě je uživatel vyzván, aby výraz přidal jako chybějící pojem do slovníku.

Uživatel může vyhledávat pojmy i dle kategorií, do kterých je pojem zařazen, a to zadáním symbolu křížku a názvu kategorie. Podobně může uživatel vyhledávat i pomocí symbolu zavináče a jména autora, nebo příkazu `/tag` a názvu vlastního štítku. Tyto výrazy nazýváme regulárními výrazy, zkráceně *regexpy* z angl. regular expression. Stejnou metodu využívá i nejznámější vyhledávač Google, u kterého místo zaškrtávacího políčka pro vyhledávání na konkrétních webových stránkách stačí zadat regulární výraz *site:adresa stránek* a klíčový výraz. Pro vyhledávání s využitím názvu štítků musí být uživatel přihlášen a mít nastavené štítky.

Štítky

Štítky jsou další uživatelskou funkcí, která slouží k seskupování pojmů do skupin, jež si uživatel sám nadefinuje podle potřeby. Pojmy lze takto seskupovat napříč více štítky najednou, tzn., že jeden pojem může mít jeden uživatel označen několika štítky zároveň.

Přidávání pojmů

Přidávání pojmů do slovníku je další významná uživatelská funkce, díky které se může slovník rozšiřovat o další pojmy. Přidávání pojmů je umožněno náhodným i přihlášeným uživatelům. Pojmy zadané uživatelem nejsou ihned veřejné a odeslaný návrh musí projít

[5] Shodný výraz může být například ZIP jako souborový formát a ZIP jako nosič dat.

úpravami, aby byla zajištěna kvalita obsahu ve slovníku. Při přidávání pojmu má uživatel možnost k pojmu přiřadit audio-vizuální obsah, a tím rozšířit výklad o prostý text.

Hodnocení pojmů

Mezi další chybějící funkce vyplývající z analýzy patří hodnocení pojmu. Hodnocení slouží jako zpětná vazba od uživatelů vyjadřující kvalitu pojmu. Vyjádření hodnocení je procentuální a hodnotit může pouze přihlášený uživatel, aby nebylo hodnocení zneužito. Každý uživatel může hodnotit pouze jednou. Jakmile vyjádří své hodnocení, zobrazí se jeho hodnocení pod celkovým hodnocením všech uživatelů a uživatel si může porovnat své hodnocení s celkovým průměrem. Jakmile je špatně hodnocený pojem upraven, veškeré hodnocení se automaticky vynuluje a uživatel může opět hodnotit.

Hlášení problému u pojmů

Hlášení problému je další zpětnou vazbou, kterou mohou uživatelé využít. Hlášení slouží k vyjádření závažnějších problémů, jakými jsou např. porušení autorské licence, gramatické chyby ve výkladu, nefunkční mediální obsah aj. Funkce je přístupná pouze pro přihlášené uživatele, a to proto, aby nedošlo ke zneužití. Při větším počtu nahlášených chyb se zobrazí varování pro uživatele, které upozorňuje na problémy.

2.1.3 Metody plnění obsahu

Obsah slovníku tvoří návrhy uživatelů. Přidané návrhy pojmů nejsou automaticky veřejné, aby byla zachována úroveň výkladu, nebo aby nedošlo k porušení autorských práv třetí strany. Neveřejné pojmy upravují editoři, kteří kontrolují validitu informací a gramatickou korektnost. V případě, že u pojmu chybí citace a doplňkový materiál v podobě obrázků, videa, zvuků nebo jiných souborů, editor takový obsah doplní, je-li to možné.

Plnění obsahu je minimálně třífázový proces, který zajistí, aby byl obsah pojmu validní a správný. První fáze je příspěvek od náhodného nebo registrovaného uživatele, který odešle svůj návrh. V tomto návrhu má uživatel možnost zadat název pojmu, kategorii, do které pojem spadá, krátký popis do 255 znaků a odkázat na audio-vizuální materiál nebo jiné doplňkové soubory. Ve druhé fázi editor zkontroluje informace v návrhu odeslaném uživatelem. Shledá-li informace správnými a dostačujícími, doplní editor pojem o jiné kategorie, než jsou kategorie zadané uživatelem v návrhu, a předá pojem k poslední fázi. Objeví-li nedostatky ve výkladu, zavádějící informace nebo stručný výklad, doplní a upraví obsah pojmu sám. V poslední fázi pak editor zkontroluje gramatickou korektnost

výkladu a překlepy, aby byl obsah pojmu na úrovni. Poté označí pojem jako autorizovaný a pojem bude veřejně přístupný.

Celý proces je pro práci editora náročný, nelze však jiným způsobem zaručit, že informace, které budou v obsahu výkladu pojmu, budou zároveň validní a gramaticky korektní. Proces eliminuje předchozí nedostatky vycházející z analýzy u el. slovníků, ve kterých se vyskytují chyby nebo zavádějící výklady pojmů.

2.1.4 Grafické funkce

Grafické funkce jsou funkce pro zobrazení informací, které se nacházejí v databázi a nejsou součástí *uživatelského grafického rozhraní*, zkráceně GUI z angl. Graphical User Interface. Pomocí těchto funkcí se zobrazují informace o pojmu, kategorie, do kterých pojem patří, hodnocení pojmu, autora pojmu, štítky, které má uživatel nastaven a varování pro uživatele.

Zobrazení kategorií slouží k orientaci uživatele, do jakého oboru pojem spadá, ale zároveň může uživatel tyto kategorie procházet a hledat další pojmy z oblasti svého zájmu.

Hodnocení pojmu vyjadřuje spokojenost uživatelů s popisem a výkladem pojmu. V případě nízkého hodnocení je uživatel vyzván ke spolupráci, aby výklad obohatil. Pojmy, které mají nízké hodnocení, lze v databázi filtrovat, a přestože není u pojmu evidován žádný problém, jsou tyto pojmy navrhovány editorům k úpravě.

Informace o autorovi využije uživatel např. při budoucím vyhledávání nebo při cíleném sledování příspěvků svého oblíbeného autora. Jméno autora je uživatelské jméno zadané při registraci.

Štítky jsou u přihlášených uživatelů zobrazeny v postranní části GUI a zároveň u pojmů, které jsou uživatelem ke štítku přiřazeny. Štítky mohou nabývat různých barev, ale název štítku je vždy bílý.

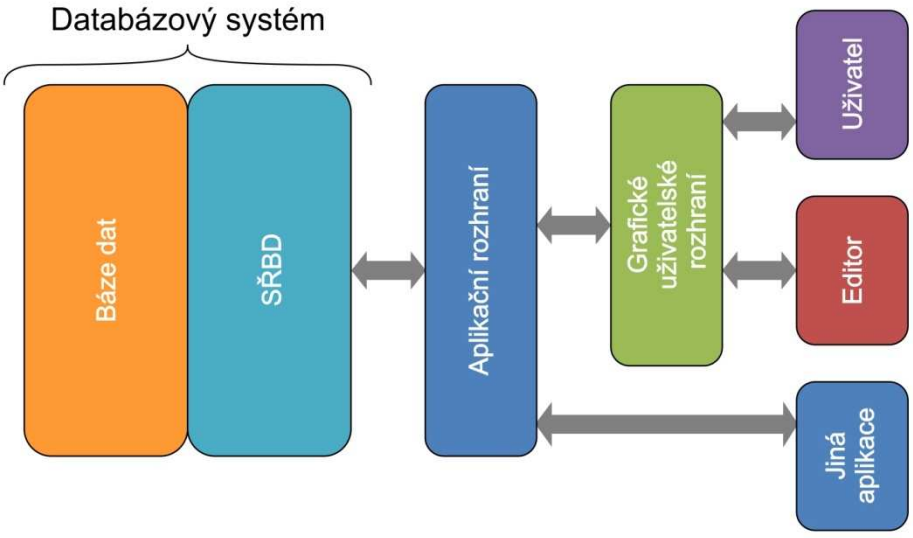
3 Návrh slovníkové aplikace

V této kapitole se zabývám návrhem vlastního řešení databázového rozhraní, které se skládá z návrhu designu databázového rozhraní, analýzy a výběru vhodných technologií, zmapování rolí, procesů a jejich definic a návrhu databázového modelu. Při návrhu vycházím z vlastních definic funkcí v kapitole č. 2.

3.1 Design aplikace a výběr technologií

V kapitole design a výběr technologií se věnuji návrhu struktury databázového rozhraní a tomu, z jakých částí bude rozhraní sestaveno. Pro každou část rozhraní provádím povrchovou analýzu dostupných technologií a volím technologii a prostředí, ve kterém bude část rozhraní vyvíjena.

Zadáním práce je vyvinout databázové rozhraní pro počítačový slovník. Rozhraní vytvořím jako webovou aplikaci, která přistupuje k informacím uloženým v databázi. Webovou aplikaci doplním o GUI pro snadnější práci s formuláři a dalšími prvky, které webovou část rozhraní ovládají. Databázové rozhraní má strukturu znázorněnou na obrázku č. 1. Každou z této části popisuji v samostatné kapitole.



Obrázek č. 1 – Blokové schéma databázového rozhraní slovníkové aplikace. Zdroj: Autor

3.1.1 Databázový systém

Na obrázku č. 1 znázorňuji propojení dat s aplikačním rozhraním. Data jsou zpracována skrze *systém řízení báze dat*, zkráceně SŘBD, *„ten společně s databází tvoří databázový systém“*, zkráceně DBS (Šeda, 2002, str. 4).

Databázové systémy mají mnoho podob. Nejrozšířenějšími jsou relační SQL systémy, po kterých následují NoSQL systémy. NoSQL systémy se dále dělí na dokumentové, Key-value, Search engine, Graph DBMS aj. Statistika a přehled 20 nejrozšířenějších systémů jsou uvedeny v příloze č. 1.

NoSQL systémy

NoSQL, z angl. Not-Only SQL, databáze nemají žádnou jasně vymezenou podobu struktury dat, proto je lze během vývoje aplikace jakkoliv upravovat podle potřeby a není nutné kvůli tomu měnit celé schéma databáze, které představuje velké zásahy. Tyto databáze jsou dynamické, což je významná vlastnost při vývoji aplikace agilní metodou vývoje. Další klíčovou vlastností je ukládání do vlastních datových souborů, umožňující nezávislost na jedné hlavní databázi. NoSQL databáze vykazují i rychlejší zpracování dat, protože nejsou závislé na referenční integritě[6] nebo uživatelském oprávnění. Výkon NoSQL databází je výraznější než u databází typu SQL. Lepší výkon spočívá v práci přímo s objekty, které jsou do datových souborů umístěny. Oproti načítání jednotlivých *záznamů – řádků* v relačních databázích pracují aplikace přímo s informacemi objektu a jeho vlastnostmi, které mají podobu proměnných. Přímý přístup k objektům a jeho vlastnostem usnadňuje vývoj aplikace, protože se programátor nemusí starat o datovou strukturu databázového systému, nýbrž pracuje s objektem přímo (Celko, 2014). K aktuálně nejrozšířenějším NoSQL databázím řadíme MongoDB, Cassandra, Elasticsearch, HBase a Neo4j. Viz příloha č. 1.

I přes všechny uvedené výhody NoSQL databází volím relační SQL SŘBD. Rozhraní úmyslně navrhuji pro využití relační SQL SŘBD, protože tento systém zajišťuje klíčové vlastnosti, jako jsou integrita a konzistence dat, jednoznačná struktura a další užitečné vlastnosti pomocí tzv. *ACID transakcí – „je to akronym pro atomicitu – konzistenci – izolovanost – trvalost.“ „Atomicitou myslíme, že se provede celá transakce nebo nic, transakce je nedělitelná. Konzistence znamená, že jakákoliv transakce v konzistentním stavu se může přenést do jiného konzistentního stavu, tedy není nijak porušena integrita*

[6] Nezkoumají se vztahy mezi entitami a platnost cizích klíčů.

dat. Izolovaností myslíme oddělení transakcí od sebe, dokud není celá transakce provedena. A trvalost se projevuje tím, že jednou provedená transakce v databázi zůstane, a to i v případě, pokud následně selže systém.“ (Date, 2011, str. 180). Tyto transakce už nemusím ošetřit sám ve vlastním kódu aplikace.

Vyloučením relačních SQL databází bych přišel o zmíněné výhody v podobě ACID transakcí, které jsou vyvíjeny profesionálními vývojáři. Snaha docílit stejné rychlosti a co nejlepší bezchybnosti při implementaci vlastního kódu by vývoj aplikace zbytečně protáhla a výsledek vlastního ošetření aplikace by byl nejasný.

Relační SQL systémy

Mezi nejznámější relační databázové systémy patří Oracle, PostgreSQL, MS SQL, SAP, MySQL, SQLite nebo novější MariaDB. Některými z těchto relačních systémů jsou draze placené systémy pro podnikové řešení, u kterých se předpokládá práce se statisíci až milióny záznamů v co nejkratším časovém úseku. I když mají velmi dobré výkonnostní vlastnosti, jsou pro můj účel tyto databázové systémy drahé a implementace těchto systémů by byla zbytečně náročná.

Proto vybírám Open Source relační SQL systémy, jako jsou MySQL, PostgreSQL, SQLite a MariaDB, jež jsou k využití zdarma a postačí nárokům mé aplikace. Z těchto Open Source systémů je těžké najít jednoznačného kandidáta, který svými výhodami převyšuje ostatní.

V bakalářské práci *Výkonnostní srovnání relačních databází* Lukáše Košárka nacházím PostgreSQL nejlepšího kandidáta z Open Source systémů. „*Ten dosáhl vůbec nejlepšího výsledku s jedním připojeným vláknem*“ (Košárek, 2010, str. 36), ale zároveň se projevily potíže při vícenásobném připojení uživatelů. „*PostgreSQL, stejně jako Firebird, spouští pro každé připojení nový proces. Při pokusu o spuštění takto velkého počtu procesů nebylo možné alokovat dostatek operační paměti pro každé připojení a PostgreSQL se ukončil. K ukončení docházelo v okamžiku 100% naplnění operační paměti.*“ (Košárek, 2010, str. 31). Vyšší propustnost v počtu připojených vláken k DBS u Open Source systémů daleko lépe zvládá MySQL, jak je patrné z grafu v příloze č. 2.

Bohužel jsem nenašel odborné publikace, které by se zabývaly srovnáním relačních SQL systémů v současnosti a zahrnovaly by i novou odnož MariaDB. Na odborných webových stránkách věnované MariaDB jsem nalezl výsledky testů při porovnání MySQL a

MariaDB. Z výsledků v článku *Performance evaluation of MariaDB 10.1 and MySQL 5.7.4-labs-tplc* (Lindstrom, 2014) a článku *MariaDB 10.1 and MySQL 5.7 performance on commodity hardware* (Schwenke, 2015) je patrné, že MariaDB má při testování nejaktuálnějších verzí těchto systémů lepší výkon než MySQL.

MariaDB je odnož systému MySQL, který aktuálně patří společnosti Oracle. Na rozdíl od MySQL je MariaDB aktivně vyvíjena a pravidelně aktualizována, systém nabízí použití nových engine, jako je *Percona XtraDB Storage Engine*, které nahrazuje *InnoDB*, a posouvá se za hranice limitů toho engine, jenž spočívá v efektivnějším využití paměti (Percona LLC, 2016), stejně jako *Aria Storage Engine*, kterou nahrazuje MyISAM (MariaDB Corporation, 2010).

Jako databázový systém pro počítačový slovník jsem si zvolil MariaDB, který z analýzy vychází jako nejrychlejší systém a nabízí nové způsoby ukládání dat do databáze.

3.1.2 Aplikační rozhraní

Další částí databázového rozhraní slovníku je aplikační rozhraní, které komunikuje mezi DBS a předává zpracované informace uživateli. Aplikační rozhraní tvoří kombinace webových technologií. V této kapitole se zabývám volbou a popisem webových technologií použitých pro realizaci rozhraní slovníku.

Statické, interaktivní a dynamické stránky

Statické stránky jsou tvořeny v jazyce *HTML*, z angl. Hyper Text Markup Language, kterým vytváříme objekty v blocích pomocí značek. Bloky mají u každého webového prohlížeče jinou grafickou podobu. Pro sjednocení stylu a vytváření grafických efektů jsem využil kaskádové styly, zkráceně *CSS* z angl. Cascade Style Sheet.

Pro vytvoření dynamických a interaktivních stránek využívám knihovnu jQuery, která je psaná v jazyce *JavaScript*, zkráceně JS. JavaScriptové knihovny, jako například již zmíněná jQuery nebo React.js, ulehčují práci při tvorbě dynamických prvků ve stránce.

Podstata těchto knihoven spočívá v již předepsaných blocích kódu JavaScriptu, které nazýváme metodami. Místo dlouhého psaní vlastního kódu JS využívám již napsané *metody*, které mi umožňují efektivněji pracovat s HTML dokumentem v podobě objektů. Objekty se hierarchicky seskupují do stromové struktury HTML dokumentu. Každý prvek má v dokumentu svou přesnou hierarchickou pozici, nějakého předka, případně potomka, např. *formulář* je potomkem *dokumentu* a zároveň rodičem *textového pole*. Této logice

říkáme *Document Object Model*, zkráceně DOM (Rutter, 2011). Pro vytvoření dynamiky zavolám požadovanou metodu a metodě předám název objektu, se kterým má pracovat. Viz ukázka kódu:

```
// Předání hodnoty z posuvníku do textového pole
$(document).ready(function(){
      function displayValSlide(){
      var sliderValue = $("#score-slider").val();
      $("#score-text").val(sliderValue + ' %');
      }
      $("#score-slider").change(displayValSlide);
      displayValSlide();
});
```

V ukázce kódu nejprve definuji funkci `$(document).ready()`, aby se všechny JavaScriptové metody vykonaly, až po načtení celého HTML dokumentu. Následuje definice metody `displayValSlide()`, která načítá hodnotu z objektu s identifikátorem `score-slider` a načtenou hodnotu předá do objektu s identifikátorem `score-text`. Poté přichází na řadu metoda, která vyvolá předchozí metodu `displayValSlide()` při jakékoliv změně v objektu `score-slider`. Jakmile uživatel změní hodnotu na posuvníku, hodnota se automaticky zobrazí v textovém poli.

Knihovna jQuery obsahuje metody pro:

a) manipulaci s objekty
b) CSS manipulaci
c) HTML události
d) efekty a animace
e) AJAX, angl. Asynchronous JavaScript and XML

Na internetu existují stovky až tisíce již hotových projektů, které si stačí pouze stáhnout a přizpůsobit si je k vlastnímu účelu, nebo grafickým vzhledem. Zajímavou a obsáhlou databází takových projektů je *The jQuery Plugin Registry*, dostupná na adrese: http://plugins.jquery.com. Databáze disponuje až stovkami hotových projektů s různým zaměřením na práci s grafickým rozhraním, vytváření fotoalb, interaktivní formuláře aj.

Skriptování na straně serveru

HTML, CSS a JavaScript jsou webové technologie, které pracují na straně uživatele. I když s kombinací těchto technologií dosáhnu graficky zpracovaných a dynamických stránek, stále postrádám technologii, díky které mohu vykonávat úkony na straně serveru, např. komunikovat s SQL SŘBD nebo generovat HTML dokumenty podle potřeby.

Mezi nejrozšířenější jazyky na straně serveru pro webové aplikace patří PHP, .NET a Java. Ze statistiky *Usage of server-side programming languages for websites*[7] uvedené na nezávislém webu, který poskytuje informace o využívaných technologiích, je zřejmé, že nejpoužívanější jazyk pro webové aplikace je PHP (W3Techs, 2016).

Jazyk PHP má solidní dokumentaci, disponuje obrovským množstvím funkcí, je nezávislý na platformě, podporuje velké množství databázových systémů a je standardem pro webhostingové služby. To jsou důvody, proč jsem se rozhodl využít tento jazyk pro vývoj databázového rozhraní.

Frameworky

V jazyce PHP existuje celá řada populárních frameworků. *„Frameworky mají klíčový význam pro vývoj rozsáhlejších objektově orientovaných softwarů. Slibují vyšší produktivitu a kratší vývojový čas a zároveň znovupoužití designu a kódu.“* (Riehle, 2000, str. 1) Vyšší produktivita frameworků spočívá v již předepsaném kódu, který mohu libovolně implementovat do svého projektu, stejně jako v případě knihovny jQuery. Frameworky se dělí podle účelu, např. na frameworky s využitím v architektuře nebo v softwaru. Softwarové frameworky nadále dělíme podle oboru využití (Wikipedia, 2016):

a) application framework
b) content management framework
c) web application framework
d) multimedia framework

Frameworků pro PHP je velké množství. Analýza a porovnání předností a záporů frameworků by vystačila na samostatnou práci. Část analýzy je provedena v práci Michala Sukupčáka *Synopsy PHP Framework* (Sukupčák, 2014, str. 10). Na technickém portálu SitePoint v článku *The Best PHP Framework for 2015: SitePoint Survey Results* jsou

[7] Statistika je součást práce jako příloha č. 3

výsledky hlasování pro PHP frameworky podle oblíbenosti vývojářů ve firemních a osobních projektech (Skvorc, 2015). Výsledky jsou součástí práce jako příloha č. 4.

Na třetím místě se umístil framework Nette, který je dílem českých vývojářů. Nette je oproti jiným frameworkům mladým frameworkem, který se začal vyvíjet v roce 2004, ale veřejně byl k dispozici až v roce 2008 jako Open Source. Framework se aktivně vyvíjí a vycházejí u něj novější a modernější verze tohoto frameworku. K dspozici má vlastní ladící nástroj *Tracy*. V české republice má tento framework velkou podporu ze strany vývojářů i společností. Nabídek pro vývojáře, kteří umí pracovat s Nette, je na českém trhu obrovské množství.

Nevýhodou frameworků je jejich rozmanitost a časová náročnost na orientaci v datové struktuře, v architektuře reprezentace dat a ve zvládnutí syntaxe. PHP frameworky se od sebe značně liší a přechod z jednoho frameworku na jiný je náročný. Kvalitně vyvíjet software ve frameworku zabere mnoho času, pokud nemá vývojář s frameworky žádnou zkušenost.

Souhrn vybraných technologií

Design databázového rozhraní počítačového slovníku je rozdělen do tří samostatných celků, které mezi sebou hierarchicky komunikují. Tzn., že uživatel, který zažádá o informace skrze GUI, předá dotaz aplikačnímu rozhraní a to následně komunikuje s DBS.

a) Databázový systém je realizován pomocí MariaDB.
b) Skripty vykonávané na straně serveru, které jsou součástí databázového rozhraní slovníku, jsou psány v jazyce PHP.
c) Grafické uživatelské rozhraní slovníku je psané webovými technologiemi HTML, CSS a JavaScript.

3.2 Role a procesy

3.2.1 Role

Obecně můžeme role dělit na dvě kategorie, a to na role aktivní a role pasivní. Rozdíl mezi těmito rolemi je v přístupu a práci se samotnou aplikací. Role aktivní mohou přímo ovlivňovat informace v systému, kdežto role pasivní pouze informace ze systému obdrží. (Jalloul, 2004).

Pro stanovení rolí je důležité držet se obecných názvů a nevytvářet duplicitní role pouhým názvoslovím. Čím méně rolí pro aplikaci navrhneme, tím jednodušší a méně náročný systém uživatelů vytvoříme. Zároveň to neznamená, že musíme role co nejvíce generalizovat a snažit se vytvořit univerzální roli s velkým obsahem oprávnění a procesů. Cílem je eliminovat duplicitní role. Jednoduše lze zaměnit roli *redaktora* a *korektora*, přičemž obě role mají v souvislosti s využitím procesů v aplikaci totožný význam. Vytvořením univerzální role *editor* se tak dostatečně vystihne význam obou těchto rolí a nemusíme mapovat procesy pro obě role zvlášť.

U databázového rozhraní slovníku pracuji celkem s pěti rolemi. Aktivní role zastupuje administrátor, editor a přihlášený uživatel. Pasivní roli představuje anonymní uživatel a aplikační rozhraní třetích stran.

Administrátor je role s plným oprávněním k jakékoliv změně v databázi i v samotném rozhraní, není nijak limitován. Roli editora zastupují uživatelé, kteří jsou určeni k úpravám informací u pojmů a na rozdíl od administrátora nemohou dělat úpravy hromadně, schvalovat kategorie a měnit svá oprávnění. Poslední aktivní rolí je přihlášený uživatel, který má možnost nastavit si štítky a hodnotit pojmy. Všechny aktivní role mohou ve slovníku vyhledávat a přidávat nové pojmy.

Mezi role pasivní patří jakýkoliv anonymní uživatel, který není přihlášen. Takový uživatel má možnost vyhledávat ve slovníku a registrovat se. Pro hodnocení a nastavení štítků je anonymní uživatel vyzván k přihlášení. Aplikační rozhraní třetích stran je speciální rolí pro získávání informací z databáze. Tato role se identifikuje na základě svého vygenerovaného *api klíče*, který je uložen v databázi slovníku. Jestliže ověření identity proběhne v pořádku, má třetí strana možnost načítat informace z databáze slovníku.

3.2.2 Procesy

Proces je obecně definován jako souhrn několika dílčích kroků k dosažení stanoveného cíle. Procesy při návrhu rozhraní popisujeme pomocí diagramů v jazyce *UML*, angl. Unified Modeling Language. Syntax UML diagramů je jednoduchá, diagram má podobu grafických primitiv.

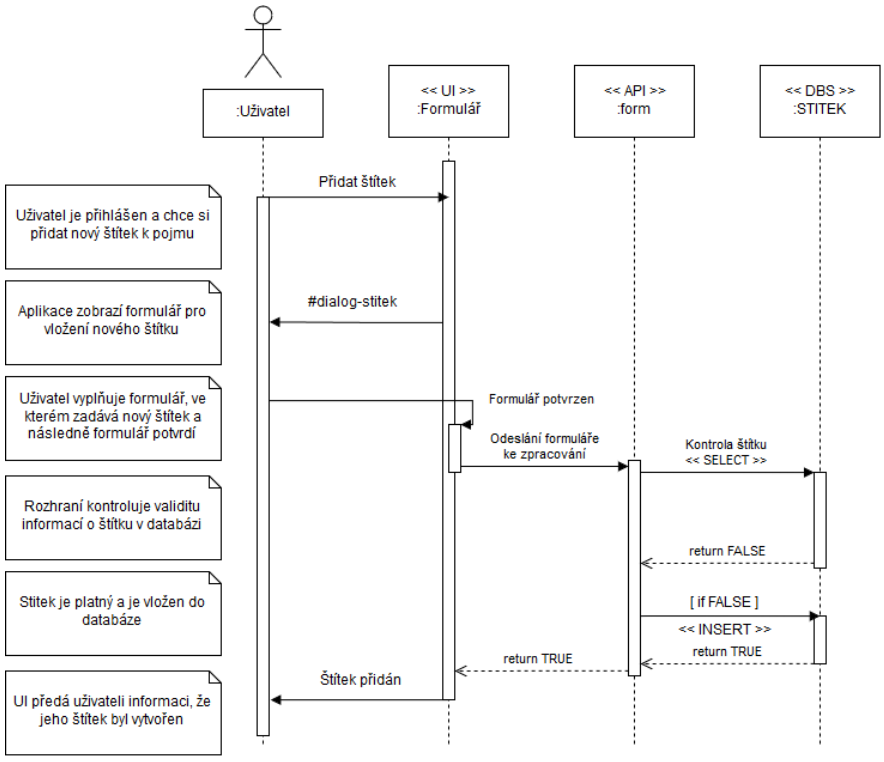
Vizualizace procesu UML diagramem nemusí primárně sloužit pouze pro samotný vývoj, ale i pro potenciálního zákazníka, který z diagramu lépe pochopí celý průběh procesu.

Proces se může skládat z desítek jednotlivých kroků, proto je vhodné pro lepší orientaci znázornit celý proces graficky.

Sekvenční UML diagram

Jedním z druhů UML diagramů jsou sekvenční diagramy, ve kterých jsou přesně popsány jednotlivé kroky v procesu. Stejný sekvenční diagram jsem použil na obrázku č. 2 pro znázornění procesu vkládání nového štítku do databáze přihlášeným uživatelem.

Z obrázku č. 2 je patrné, jak celý proces přidávání štítku uživatelem funguje, a v jakých krocích se zapojují jednotlivé části celého rozhraní. Takto mohu zmapovat všechny procesy pro konkrétní role a objasnit celý design databázového rozhraní.



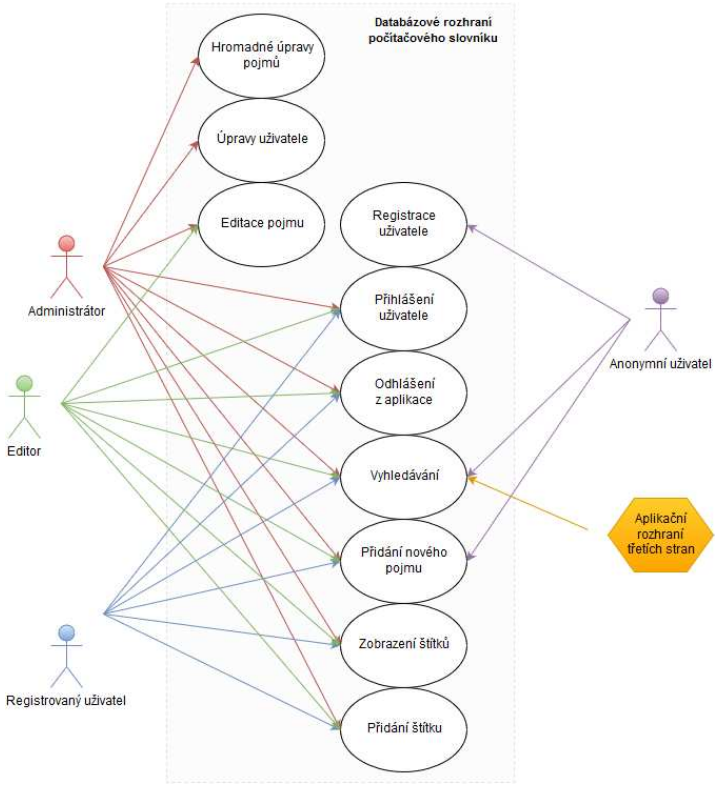
Obrázek č. 2 – Sekvenční UML diagram pro přidání pojmu uživatelem. Zdroj: Autor

V prvním kroku kontroluji, zda je uživatel přihlášen. Uživatelé, kteří nejsou přihlášeni, nemají možnost přidávat štítky – nezobrazí se jim dialog pro přidání štítku. V dalším kroku předá GUI dialog pro přidání štítku, který obsahuje formulář s textovým polem, výběrem barvy a tlačítkem pro potvrzení. Jakmile uživatel odešle formulář, předají se informace ke zpracování databázovým rozhraním. Rozhraní kontaktuje databázový systém a předá získané informace z formuláře. Databázový systém odpoví rozhraní nepravdu – *FALSE*.

Údaje zadané uživatelem neexistují, to je správně, jinak by nemohl být nový štítek přidán. Rozhraní předá nový dotaz do databázového systému k vložení těchto údajů. V případě úspěchu vrátí databázový systém hodnotu *TRUE*, kterou rozhraní vyhodnotí jako úspěch, a předá GUI informaci, aby zobrazil upozornění pro uživatele, že štítek byl přidán.

Use case diagram

Další variantou vizualizace procesů a rolí je UML *diagram případů užití*, angl. Use case diagram. Diagram případů užití nevyjadřuje jednotlivé kroky procesu tak, jako u sekvenčních diagramů, ale znázorňuje všechny role, procesy a vztahy mezi nimi v jednom diagramu. Vztahy v tomto diagramu znázorňují, jaké procesy má každá role přidělené (Jalloul, 2004).



Obrázek č. 3 – Use Case diagram. Zdroj: Autor

Obrázek č. 3 detailně popisuje, jaké role a procesy jsou pro databázové rozhraní slovníku navrženy. Každá role má přiřazené své procesy, na které směřuje ukazatelem šipky. Na pohled je zřejmé, že např. anonymní uživatel má možnost registrovat se, vyhledávat a přidávat pojmy. Ostatní procesy nemá přidělené, a proto je nemůže využít.

3.3 Konceptuální a logický model databáze

Návrh databáze se skládá z několika kroků, ve kterých je třeba určit, jaká data budou v databázi uložena, jakou budou mít strukturu a jaké jsou vzájemné vztahy mezi daty v databázi. Po těchto krocích mohu vytvořit modely, které vyjadřují podobu datové struktury. Takové modely označujeme jako konceptuální, logický a fyzický model.

Při konceptuálním modelování pracuji s analýzou dat a jejich reprezentací jako s objekty. Každý objekt má své *vlastnosti* - *atributy* a je propojen s jinými objekty pomocí vztahů. Model může mít hierarchickou, Entitně-Relační, zkráceně ER nebo i objektově-orientovanou podobu reprezentace dat. V závěru kapitoly č. 3.1.1 vybírám pro vlastní slovník databázový systém MariaDB, který má charakter relační databáze, tzn., že konceptuální model musí být v souladu s tímto systémem a má podobu Entitně-Relačního schématu.

Návrh konceptuálního modelu provádím ve dvou fázích. Nejprve si nadefinuji seznam entit a následně vytvořím ER diagram, kterým vyjádřím vztahy mezi entitami. Poté k entitám doplním jejich atributy a celé schéma normalizuji. Návrh konceptuálního modelu lze vytvořit v jednom kroku již jako normalizovaný, záleží na osobní preferenci.

Seznam entit a jejich atributů:

Entita	Atributy
STITEK	idStitek, nazev, barva, pismo, idUzivatel
UZIVATEL	idUzivatel, jmeno, prijmeni, uzivjmeno, email, heslo, telefon, avatar, www, narozen, vytvořen, api_key, idRole
ROLE	idRole, nazev
POJEM	idPojem, nazev, zkratka, popis, vyklad, uroven, autor, vytvoren, schvaleno, schvalil, proces, verejne
SYNONYM	idPojem, synonym
KATEGORIE	idKategorie, idNadrazene, nazev, verejne, schvaleno, schvalil
HODNOCENI	idUzivatel, idPojem, hodnota
PROBLEM	idProblem, nazev
OBSAH	idObsah, nazev, odkaz, lokalne, pripona, verejne, idKategorie
OBEZNIK	email, odebirat
CITACE	idCitace, autor, titul, nakladatelstvi, isbn, citace

Tabulka č. 2 – Přehled entit a jejich atributů. Zdroj: Autor

Konceptuální model mohu vytvořit softwarově, nebo např. na papír, v obou případech však musím dodržet stanovenou konvenci podoby diagramu použitím grafických primitiv. Pro modelování konceptuálního modelu využívám nástroj *Draw.io*, který je dostupný na adrese: http://draw.io. Draw.io je nástroj pro tvorbu diagramů, schémat a technických návrhů, jenž spolehlivě nahradí komerční i úzce specializované programy pro tvorbu konkrétního druhu diagramu. Dostupný je zdarma a online, export dat je v podobě PDF, XML, HTML a SVG souboru nebo jako obrázek typu JPEG a PNG. Nástroj podporuje ukládání přímo na Google Drive, Dropbox, OneDrive, nebo na pevný disk. Konceptuální model v podobě ER diagramu pro databázi slovníkové aplikace je součástí přílohy č. 5.

Teprve po vytvoření konceptuálního modelu mohu vytvořit logický model databáze. Logický model, na rozdíl od modelu konceptuálního, řeší už konkrétní podobu atributů. Tzn., že si v logickém modelu stanovím, jakým datovým typem jsou záznamy atributu, zda je atribut primárním nebo cizím klíčem, či nikoliv, definuji indexy aj. Logický model musím navrhnout vždy softwarově. Pro návrh logického modelu volím vývojové prostředí MySQL Workbench Community edition.

MySQL Workbench je ve verzi *Community edition* volně dostupný a zdarma k použití na neomezenou dobu na všech nejznámějších platformách. Tato verze plnohodnotně dostačuje pro funkce, jakými jsou modelování databázového schématu, připojení k databázovému serveru a vytváření SQL skriptů. Dostupné jsou i další varianty nástrojů pro modelování databázového schématu, jako např. Database Workbench, SQLyog, HeidiSQL, dbForge Studio a další. Většina z nich jsou *trialové verze*, jejichž použití je zdarma na omezenou dobu, případně s nutností registrace. Žádný z těchto nástrojů mi při vytváření logického schéma nevyhovoval tak, jako MySQL Workbench.

Při tvorbě logického schématu kontroluje MySQL Workbench různé závislosti, jako jsou cizí klíče, nebo při vztahu s kardinalitou M:N automaticky vytváří další tabulku pro tento vztah. Grafické rozhraní tohoto nástroje je velmi příjemné a intuitivní. Proto jsem se rozhodl zvolit tento nástroj pro tvorbu logického schématu databáze.

Před založením nového logického schématu v MySQL Workbench je třeba provést úpravy nastavení, ve kterém si nadefinuji nové chování programu při vytváření nových diagramů, objektů a vztahů mezi objekty. Každý výchozí model vytvořený ve Workbench nástroji má způsob ukládání tabulek v *InnoDB engine*, výchozí nastavení změním na *XtraDB*, které MariaDB podporuje. O výhodách *databázových engine* u databázového systému MariaDB

pojednávám v kapitole č. 3.1.1. Workbench při vytváření cizích klíčů vytváří název cizího klíče s názvem tabulky, odkud pochází a v níž je primárním klíčem, např. při vytvoření cizího klíče `idUzivatel` do tabulky `STITEK`, se vytvoří název cizího klíče `UZIVATEL_idUzivatel`. Toto nastavení upravuji, aby se vkládal pouze název klíče. Posledním nastavením je změna chování při vytváření vztahů M:N. Tabulka vytvořená vztahem M:N má název složený z názvů tabulek, k nimž vztah patří, a zároveň obsahuje spojení `_has_`, které není potřebné.

Ačkoliv je MySQL Workbench určen pro databáze typu MySQL, mohu tento nástroj využít i pro tvorbu logického schématu pro MariaDB. Rozdíly v datových typech, kontrole cizích klíčů a jiných závislostí ošetřím nastavením chování programu, anebo úpravou SQL skriptu pro vytvoření fyzického schématu databáze. Logické schéma databáze je součástí přílohy č. 6.

4 Vývojové prostředí

V této kapitole se zabývám analýzou a výběrem vhodných softwarových nástrojů pro vývoj databázového rozhraní. Popisuji, co jsou vývojová prostředí a k čemu během vývoje slouží. U každého vybraného nástroje pak uvádím nastavení a základní klávesové zkratky pro efektivnější práci s nástrojem.

Vývojová prostředí, zkráceně *IDE* z angl. Integrated Development Environment, jsou softwarové nástroje, které usnadňují práci vývojáře při realizaci projektu, na kterém pracuje. Usnadnění práce spočívá ve využití široké škály integrovaných nástrojů, kterými vývojové prostředí disponuje. Nástroje mají své specifické vlastnosti a obsahují předpřipravené části modelu databáze nebo zdrojového kódu, které se při vývoji obecně používají.

IDE mají nástroje jako je editor, který představuje prázdné okno pro psaní zdrojového kódu, a kompilátor, jenž překládá napsaný zdrojový kód do strojového kódu. Následně se strojový kód spustí a ověří se funkčnost zdrojového kódu, nebo může prostředí využít přímého interpreta pro jazyk, ve kterém je zdrojový kód napsán, bez nutnosti kód překládat kompilátorem. Některá IDE mají i ladící nástroje pro *debugování,* z angl. debugging. Debugování je proces analýzy a odstranění chyb ve zdrojovém kódu. Chyby mají podobu překlepů, nevalidní syntaxe, volání funkcí nebo proměnných, které nejsou definovány aj. Dalšími nástroji jsou nástroje pro testování a refaktorování kódu. Refaktorování kódu znamená úpravu kódu do čitelnější podoby pro rychlejší a snazší orientaci v kódu, bez zásadních změn samotných algoritmů. Nejcennějším nástrojem ve vývojovém prostředí je našeptávání kódu v editoru vývojového prostředí. Našeptávání efektivně zkracuje dobu psaní kódu a zároveň předchází zbytečným chybám nebo překlepům. Při psaní zdrojového kódu v editoru nástroj našeptávání analyzuje část slova a nabídne mi možnosti kódu, které chci pravděpodobně napsat. Vývojové prostředí za mě automaticky kontroluje, jestli nemám chybu v syntaxi a používám správné názvy tříd, proměnných aj. V případě, že mám chybu v kódu, upozorní prostředí na tuto chybu v různých podobách, jako např. podtržením textu nebo vykřičníkem v řádku, kde se chyba nachází. Rozličnost nástrojů ve vývojovém prostředí je rozdílná a každé prostředí disponuje jinými nástroji s jinými vlastnostmi.

Zvolená vývojová prostředí

K vývoji databázového rozhraní je potřeba kombinace několika vývojových prostředí. Každé prostředí má své specifické použití a teprve kombinací těchto prostředí lze dosáhnout výsledku. Pro vývoj databáze využívám prostředí MySQL Workbench. Dále však potřebuji prostředí pro realizaci zdrojového kódu databázového rozhraní, napsaného v jazyce PHP.

Mezi nejoblíbenější vývojová prostředí pro psaní zdrojového kódu patří PhpStorm, Sublime Text, NetBeans a Zend Studio. To potvrzují i výsledky v článku *Best PHP IDE in 2014 – Survey Results* (Skvorc, 2014). Jasným výhercem je prostředí PhpStorm, viz příloha č. 7, které disponuje všemi zmíněnými nástroji včetně refaktoringu a testování kódu. PhpStorm je dostupný pro studenty v plné verzi zdarma po dobu jednoho roku od data registrace, a proto jsem se rozhodl nástroj využít k realizaci zdrojového kódu databázového rozhraní.

PhpStorm je náročnější IDE na osvojení všech jeho funkcí, nástrojů a klávesových zkratek. Prostředí má obrovské možnosti nastavení a přizpůsobení si chování jednotlivých nástrojů. Základní práce s programem je bez potíží. Mezi nejcharakterističtější rysy PhpStormu patří různé druhy vyhledávání, jako např. *Search everywhere*, které umožňuje vyhledávat v jakékoliv části prostředí, a to i v nastavení programu a jeho vlastních zdrojových souborech. Dalšími možnostmi je vyhledávání tříd, funkcí, proměnných aj. V obou případech je možné vyhledávat pomocí syntaxe CamelCase, tzn., že zadaný výraz pro vyhledávání se skládá z několika slov spojených dohromady a každé počáteční písmeno spojeného slova je velké, viz *NázHleVýraz* – název hledaného výrazu.

Nejužitečnější funkcí, kterou v prostředí PhpStormu využívám, je definování si vlastního kusu zdrojového kódu do nástroje našeptávání pomocí *Live Templates*. Vlastní kus kódu, který opakovaně využívám, vložím do vlastní šablony a pojmenuji ji vhodným názvem, pomocí něhož šablonu v našeptávání vyvolám a následně vložím obsah vlastního kódu v připravené šabloně.

Další funkcí, kterou má editor PhpStormu, je *Complete Statement*. Tato funkce identifikuje blok kódu, v němž se nachází kurzor, a při stisku klávesové zkratky *Ctrl+Shift+Enter* funkce automaticky dokončí celou část daného bloku, např. složené závorky a středník.

5 Metody vývoje

V kapitole metody vývoje se zabývám známými metodami, které se využívají při vývoji software. K vybraným metodám popisuji jejich charakteristiku a princip vývoje. K závěru kapitoly popisuji agilní metodu vývoje, kterou jsem si zvolil pro vývoj databázového rozhraní, a charakterizuji její stěžejní vlastnosti.

Každý vývoj softwaru je vyvíjen nějakými technikami, postupy či pravidly, souhrnně lze všechny tyto části sjednotit a označit za metodu vývoje. Některé metody vývoje jsou natolik populární, že pro ně vznikají frameworky, které jsou rozvíjeny, podporovány a šířeny společnostmi, jako např. *Rational Unified Proces* od společnosti IBM, nebo komunitou vývojářů, jako v případě *Agile Unified Proces*, kteří tvoří alianci agilních metodik.

Obecně se každá metoda vývoje skládá z paradigmatu, které je pro metodu charakteristické, a ze scénáře, který tvoří jednotlivé kroky vývoje. V dnešní době dělíme metody vývoje na dva hlavní proudy, jež jsou rigorózní, a agilní metodiky.

Každá metoda vývoje, ať už spadá do proudu rigorózních, nebo agilních metod, má své klady a zápory vzhledem k vyvíjenému software. Vzhledem ke svým kladům a záporům je každá metodika vývoje vhodnější pro jiný druh vyvíjeného software. Nelze jednoznačně říct, že jedna metoda vývoje je lepší než druhá. Přesto je ze statistiky v článcích *Stats: Project Methodologies 2011 a Stats: Project Methodologies 2013* zřejmé, že podíl agilních metodik roste oproti podílu rigorózních metod, které v článku zastupuje vodopádový model a V-Model (Young, 2012), (Young, 2013). Statistiky jsou součástí přílohy č. 9 a přílohy č. 10.

5.1 Rigorózní metody vývoje

Pro rigorózní metody vývoje je charakteristické, že se striktně drží svých postupů, a jakákoliv odchylka, jež není součástí metody, je nežádoucí. Metody mají přesně a podrobně definovaný proces vývoje, který lze opakovat. Požadavky na výsledný software jsou definovány před počátkem vývoje a dle požadavků je sestaven plán realizace. U rigorózních metod má každý člen týmu přidělenou svoji roli, v níž zastává určené procesy, a proto je u každého člena týmu kladen důraz na specializaci. Pohled na člověka je

sekundární a zastává se názor, že každý je nahraditelný, a to díky rozsáhlé dokumentaci, která se během vývoje vytváří a je rovněž charakteristická pro rigorózní metody vývoje.

Mezi nejznámější rigorózní metody patří:

a) vodopádový model
b) prototypový model
c) iterativní a inkrementální model
d) V-Model
e) spirálový model

5.2 Agilní metody vývoje

Oproti rigorózním metodám vývoje je princip agilních metod opačný, tzn., že jsou agilní metody flexibilní, snáze reagují na chybu nebo podnět od zákazníka a nemají problém se ihned přizpůsobit.

Pojem a princip agilních metod je poprvé sjednocen od roku 2001. V tomto roce je skupinou softwarových inženýrů a vývojářů sepsán *manifest agilního programování*. Manifest vychází jako kritika rigorózních metod vývoje a zaměřuje se na čtyři základní hodnoty agilního vývoje software (Agille Alliance, 2001):

a) „Jednotlivci a interakce před procesy a nástroji
b) Fungující software před vyčerpávající dokumentací
c) Spolupráce se zákazníkem před vyjednáváním o smlouvě
d) Reagování na změny před dodržováním plánu“

U agilních metod vývoje je kladen důraz na častější dodávání funkčního softwaru a komunikaci se zákazníkem. Osobní setkání mezi členy týmu je považováno za nejefektivnější formu komunikace. Jedinec v týmu je cennější než nástroj, neboť může při analýze problému inspirovat ostatní a přispět k rychlému nalezení řešení.

Agilní metody jsou spíše sadou doporučení, jak při vývoji postupovat, než aby popisovaly jednotlivé kroky vývoje software, ty naopak agilní metody popírají. Postup vývoje agilní metodou probíhá v krátkých časových úsecích. Ve vývoji agilní metodou je kladen větší důraz na vyvíjený software, než na psaní dokumentace, a je vhodný pro vývoj v týmu, nežli pro jednotlivce.

Nejznámější agilní metody vývoje jsou:

a) extrémní programování
b) Scrum
c) testy řízený vývoj
d) vlastnostmi řízený vývoj
e) Crystal

5.2.1 Extrémní programování

Extrémní programování, zkráceně XP, je nejrozšířenější agilní metodou vývoje. Princip extrémního programování spočívá v co největší jednoduchosti. Programátor píše jen takovou část softwaru, která je právě vyžadována. XP předpokládá, že požadavky na software se budou během vývoje měnit, a proto nemá význam vyvíjet části software do budoucna. *„XP dodává nejjednodušší možné verze, které jsou schopné provozu, verze dodává po velmi krátkých iteracích.“* (Popelka, 2014)

Pro XP je zásadní komunikace mezi všemi zainteresovanými subjekty, tzv. *stakeholdery*. V případě, že není zajištěna dobrá komunikace např. se zákazníkem, může vzniknout produkt, který nebyl očekáván. Významným rysem pro XP je časté testování za pomocí automatizovaných testů, programátoři během vývojového cyklu neustále vytvářejí a provádí jednotkové testy.

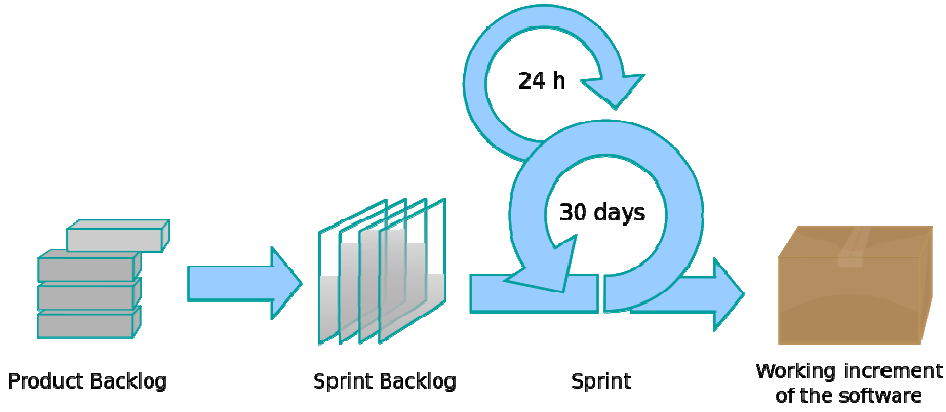
Extrémnost v této metodě spočívá v neustálém opakování činností, které se prokázaly jako přínosné pro vývoj, např. již zmíněné testování, jednoduchost, velmi krátké iterace aj. Autor metody XP Kent Beck popisuje, že při prvním vyslovení XP má vizi knoflíků s ovládacím panelem. *„Každý knoflík byl proces a ze zkušenosti jsem věděl, že to funguje dobře. Obrátil jsem všechny knoflíky na 10 a sledoval, co se stane. Byl jsem překvapen, když jsem zjistil, že celý balík procesů fungoval stabilně, předvídatelně a flexibilně.“* (Beck, 1999).

5.2.2 Scrum

Scrum, česky skrumáž, je agilní metoda vývoje, jež se orientuje na vývojový tým, který tvoří jeden celek k dosažení stejného cíle a je jako celek odpovědný za dodávané části produktu. Vývoj metodou Scrum probíhá v krátkých iteracích, jež nazýváme sprinty.

Sprint se skládá ze dvou částí, které představuje *plánování sprintu* a *denní scrum*. Při plánování sprintu vývojový tým vybírá, jaké práce se za následující sprint udělají, tvoří

backlog a připraví úkoly sprintu s časovým odhadem, jak dlouho bude trvat jejich splnění. Další částí sprintu je denní scrum, který spočívá v každodenním setkávání členů týmu. Denní scrum je někdy označován jako *stand-up* a smyslem tohoto setkávání je report dokončených úkolů nebo řešení problémů, na které člen týmu při řešení úkolu narazil. Celý sprint trvá v rozmezí jednoho až tří týdnů a během něj jsou vyvíjeny části produktu. Každý člen vývojového týmu si sám vybírá, na jaké části bude pracovat. Grafická podoba procesu vývoje metodou Scrum je na obrázku č. 4.



Obrázek č. 4, Scrum process. Zdroj: Autor: Lakeworks
https://commons.wikimedia.org/w/index.php?curid=3526338

6 Implementace databázového rozhraní

V kapitole implementace databázového rozhraní popisuji postup implementace jednotlivých částí, ze kterých se skládá celé databázové rozhraní pro slovník. V jednotlivých pasážích textu se zabývám řešením problémů, které se během implementace vyskytly, a text doplňuji vlastním řešením.

6.1 Příprava prostředí k implementaci

Pro vývoj a následné otestování databázového rozhraní si připravím prostředí, ve kterém poběží části rozhraní slovníku. Prostředí se skládá z databázového systému, v němž je uložena báze dat s informacemi, jež tvoří slovník, a z webového serveru s podporou PHP, který přeloží skripty v jazyce PHP a předá vygenerovanou webovou stránku.

Takové prostředí lze realizovat s instalovaným software MariaDB a IIS, které podporuje PHP skripty skrze modul CGI. V linuxovém prostředí lze instalovat kombinaci software MariaDB, Apache a modul PHP. Instalace a konfigurace těchto softwarových částí je zdlouhavá a k vývoji databázového rozhraní volím již připravené řešení, jež je známé pod zkratkou *XAMPP*, která znamená, že je balík nezávislý na platformě a obsahuje webový server Apache, databázový server MariaDB, modul pro zpracování skriptů v PHP a Perl.

Připravený balík aplikací XAMPP volím z toho důvodu, že obsahuje databázi MariaDB, kterou jiné, známé řešení s názvem WAMPServer nemá. XAMPP se dobře ovládá přes kontrolní panel, který nabízí sadu tlačítek, pomocí nichž mohu přímo přistupovat k editaci konfiguračních souborů, nebo každou část celého balíku postupně zapnout a ovládat. XAMPP zároveň disponuje i logovacím oknem ve spodní části kontrolního panelu, které hlásí stavy celého prostředí a jednotlivých částí.

Vlastní nastavení jednotlivých částí prostředí XAMPP není nutné, protože balík je ve výchozím stavu připraven ihned k použití. Výchozí hodnoty jsou nastaveny pro lokální použití, jež právě pro vývoj rozhraní využívám.

6.2 Propojení prostředí s nástroji pro vývoj

Pro vkládání SQL dotazů, přes které komunikuji se SŘBD, se musím připojit k databázovému serveru pomocí MySQL Workbench. Spojení MySQL Workbench zvolím na úvodní obrazovce programu výběrem *MySQL Connections* a vyplním základní údaje o připojení k databázovému serveru, který má při lokálním použití standardně adresu

`127.0.0.1` s portem `3306`. Po připojení k databázovému serveru mám možnost psát SQL dotazy a komunikovat se serverem.

Prostředí XAMPP musím propojit i s PhpStormem a to hlavně kvůli absenci interpreta jazyku PHP v PhpStormu. Bez spojení PhpStormu s webovým serverem nemám možnost ověřit funkčnost svých PHP skriptů. V nastavení programu vyhledám položku s názvem *Build, Execution, Deployment,* poté následuje položka *Deployment*. V této položce provedu požadované nastavení. Přidám si nové spojení, které odkazuje na místní složku s projektem databázového rozhraní slovníku, ale soubory budu procházet skrze webový server na adrese `http://localhost`, jelikož využívám lokální server Apache, který je součástí XAMPP.

6.3 Implementace databáze

Při vytvoření fyzického schématu databáze vycházím z návrhu logického schématu, realizovaného v MySQL Workbench. Nástroj Workbench má možnost vygenerovat SQL skript pro vytvoření databáze. Skript je vygenerován na základě vztahů, objektů a jejich atributů vytvořených v logickém schématu.

U generování skriptu procházím dialogovým průvodcem, ve kterém mám možnost nastavit další parametry skriptu. Zde si zvolím parametr, aby se mi před každým vytvořením nové tabulky vytvořil příkaz na odstranění tabulky stejného názvu, pokud již existuje. Workbench někdy vytvoří ve vygenerovaném skriptu dvě shodné tabulky se stejnými názvy a atributy. V dalším kroku využívám filtr, kterým určuji, jaké objekty chci zahrnout do vygenerování skriptu, a následně nechám skript vygenerovat.

Vytvořením fyzického schématu databáze pomocí vygenerovaného skriptu z MySQL Workbench narážím na potíže s kontrolou výchozích hodnot u atributů, které jsou u MySQL povoleny. MariaDB je však nepodporuje, proto jsem výchozí hodnoty u atributů ze skriptu odstranil. Dalším problémem je při použití SQL módu `TRADITIONAL`, jenž v mém případě blokuje vytvoření záznamů s nulovým datem. Proto tento parametr odstraňuji a ponechávám parametr `ALLOW_INVALID_DATES` (MariaDB, 2012).

6.4 Problémy spojené s implementací

Ve vlastní implementaci rozhraní slovníku využívám funkce jako registrace uživatele, přihlášení k odběru novinek nebo odeslání kontaktního formuláře. Tyto funkce jsou vázané na PHP funkci `mail()`, která odesílá uživateli informace podle toho, jakou z uvedených

funkcí právě použil. Při implementaci rozhraní na lokálním prostředí nemohu PHP funkci `mail()` použít, neboť nemám dostupný mailserver.

Problém jsem se rozhodl vyřešit využitím webhostingu s vlastní doménou, který je dostupný na adrese `http://pocitacovyslovnik.cz`. Webhosting má stejný typ databáze a verzi PHP modulu jako prostředí XAMPP, proto se nevyskytl žádný problém při přenesení souborů na webhosting.

7 Testování vlastní aplikace

V kapitole testování vlastní aplikace popisuji důvody, proč je testování aplikací důležité a jakými způsoby lze aplikaci testovat. V kapitolách, jež se zaměřují na konkrétní způsob testování, popisuji problematiku podrobněji a přikládám vlastní vzor testu, který jsem aplikoval na implementované části databázového rozhraní.

Testování je nedílnou součástí jakéhokoliv vývoje aplikací a slouží k odhalování chyb, které se během vývoje neúmyslně vytvoří. Některé druhy testů se provádí i u softwaru, jenž je již po nějakou dobu využíván v praxi, a to proto, že s vývojem technologií se nachází nové metody, jak lez původně bezpečný kód obejít.

Alespoň základní testování aplikace vždy provádí během vývoje programátor, který postupně zkouší, zda aplikace funguje správně a zobrazuje očekávané výsledky. Tato metoda testování je nejzákladnější součástí procesu testování aplikace.

Celý proces testování je komplexní a rozděluje se do několika etap, ve kterých provádí testování pokaždé jiný subjekt. Celý proces důkladného testování je časově náročný a může trvat i déle než samostatný vývoj aplikace. Z toho důvodu se v některých projektech určité etapy testování úmyslně vynechají, aby nebyl výsledný produkt levnější. Zároveň to znamená vyšší procento chyb v aplikaci, protože vynechané etapy testování tyto chyby neodhalí, a v důsledku může v provozu neotestovaná aplikace napáchat větší škody, nežli vyšší cena samotného produktu s provedenými testy.

Cílem testování je odhalit co největší počet chyb a ty následně opravit, aby mohl být výsledný produkt považován za spolehlivý a použitelný. Chyba znamená jakékoliv neočekávané chování aplikace, které vede k jinému než očekávanému stavu. Ten může představovat pád samostatné aplikace nebo poškození zdrojových dat. Z tohoto důvodu je důležité provádět testování aplikace ve zvláštním testovacím prostředí, které je kopií vyvíjeného produktu, aby nedošlo k nevratným změnám. Zároveň je vhodné testované části aplikace verzovat.

Verzování aplikace znamená uchovávat historii provedených změn ve zdrojovém kódu nebo sadě souborů aplikace či databáze. Pro verzování využíváme *systém správy verzí*, zkráceně VCS z angl. Version Control System. VCS systém může mít podobu lokální pro verzování na lokální úrovni, centralizovanou pro verzování např. v týmu, nebo

distribuovanou, kterou je možné rovněž použít pro verzování v týmu s tím rozdílem, že každý napojený uživatel u sebe uchovává kompletní kopii celého repozitáře. Známé VCS systémy jsou Git, Mercurial, Subversion, Perforce aj (Chacon a Straub, 2016). Verzování vlastního projektu je možné přímo z vývojového prostředí PhpStorm, které podporuje zmíněné VCS systémy.

Vlastní testovací prostředí není jen z důvodu předcházení poškození nebo ztráty dat, ale i kvůli možnosti simulace různých druhů stavu systému, jako např. úmyslně chybná data, čímž testujeme, jak se aplikace k takové chybě zachová a zda správně projeví chybu v té části aplikace, která je právě testována, nebo zda se chyba například neprojeví vůbec. Jestliže se chyba vůbec neprojeví, přestože aplikaci chybí data, můžeme takový stav rovněž označit za chybný.

Přípravu takového prostředí má na starost určený tester, který musí být schopen takové prostředí vytvořit a umět mimořádné stavy nasimulovat. Tester musí být technicky zdatný, aby si dokázal prostředí vytvořit, uměl pracovat s doprovodnými částmi aplikace, jako např. databázový systém, aby v něm mohl provést např. záměnu dat a zároveň být zdatný v programování, aby porozuměl kódu aplikace. Nejcennější vlastností testera je silné analytické myšlení a grafické a funkční cítění.

Je patrné, že takových testerů nebude mnoho, a to je rovněž jeden z důvodů, proč se celý proces testování dělí do několika částí, v nichž každý tester, který je součástí většího týmu, plní vlastní podstatnou úlohu. Celý *„tento proces se nazývá řízení jakosti neboli kvality“* (Míka, 2013, str. 3).

Testování tedy můžeme rozdělit do 7 samostatných částí podle druhu prováděných testů (Hlava, 2011):

a) testování programátorem
b) jednotkové testování
c) funkční testování
d) integrační testování
e) systémové testování
f) akceptační testování
g) penetrační testování

Testování programátorem probíhá ve chvíli, kdy programátor sestaví část zdrojového kódu a automaticky ověřuje, že daná část funguje podle předpokladů. Tomáš Hlava uvádí ve svém článku *Fáze a úrovně provádění testů*, že může jít o tzv. *test čtyř očí*, kdy programátor netestuje sám sebou vytvořený kód, ale předá kód ke kontrole jinému programátorovi, který hledá chyby. Zároveň uvádí, že *„v praxi je bohužel tento stupeň testování často podceňován. Přitom oprava chyb v této části testování software je nejméně nákladná.“* (Hlava, 2011). Takový způsob testování, resp. kontroly, můžeme aplikovat v případě, kdy se na vývoji aplikace podílí více než jeden programátor.

V následujících kapitolách se věnuji z mého pohledu nejdůležitějším částem celého procesu testování s ohledem k vlastnímu vývoji databázového rozhraní slovníku, který má podobu webové aplikace.

7.1 Jednotkové testy

V této části testování jde o sestavení *jednotkového testu*, angl. Unit test, který kontroluje tzv. jednotky. Ty představují část zdrojového kódu aplikace, kterou např. v případě objektově orientovaného programování reprezentuje dle rozsahu zdrojového kódu samostatná třída nebo metoda. Test má podobu programového kódu, proto tuto část testování nejčastěji vykonává programátor. Příprava takových testů může dokonce trvat i déle než samotný vývoj aplikace.

„Jednotkový test je tedy v podstatě část kódu, který aktivuje jednotku a kontroluje výsledek jednoho specifického konce testované jednotky. Pokud je předpokládaný konec výsledku špatný, test jednotky selhal.“ (Osherove, 2014, str. 5) Tzn., že poté musí programátor nebo vývojář najít chybnou část jednotky, tu následně opravit a spustit test znovu.

Vytvoření kvalitního jednotkového testu je náročné na čas. Vzhledem k těmto nárokům, které jsou pro vytvoření jednotkových testů vyžadovány, je efektivní provádět tento typ testu již v počátku vývoje aplikace, protože *„testy jednotek se velmi špatně aplikují na již zaběhlých projektech. U již vytvořených aplikací se většinou musí provést kompletní refaktoring kódu, či dokonce mnohem hlubší úpravy.“* (Hlava, 2011).

Právě tento test je nejvhodnější metodou ověření funkčnosti refaktorovaného kódu. V principu je nejprve sestaven jednotkový test, který ověří bezchybné fungování funkční jednotky aplikace. Poté přejde programátor po malých krocích k refaktorování kódu

jednotky, a následně testem opět ověří, jestli se funkčnost stejné jednotky aplikace nezměnila a je stále bezchybná.

Pro ověření vlastní části rozhraní slovníku rovněž využívám jednotkový test. Test je sestaven pomocí vývojového prostředí PhpStorm, které umožňuje sestavit si a asociovat jednotkový test ke konkrétní jednotce. PhpStorm disponuje frameworkem *PHPUnit* pro testování jednotek a nabízí možnost vkládání předepsaných bloků pro sestavení testu.

Stejně jako v případě vývoje aplikace, i pro jednotkové testy se nabízí řady frameworků. Pro jazyk PHP je dostupný framework PHPUnit. Účel frameworku je stejný jako v případě vývoje aplikace.

Jednotkový test se obecně skládá ze tří částí (Osherove, 2014, str. 24):

a) Aranžování objektů, jejich vytvoření a nastavení, což znamená, že si vytvoříme objekt, na kterém provedeme test, a nastavíme si pro něj vstupní a výstupní proměnné.
b) Vykonání testu – provedeme samostatný test s metodou, kterou právě testujeme.
c) Tvrzení nějakého výsledku, jenž je očekáván – vytvoříme platné tvrzení, které je bezchybné pro testovanou metodu, a porovnáme tvrzení s výsledkem testované metody.

V následující ukázce je podoba jednotkového testu vytvořeného s pomocí testovacího frameworku PHPUnit. Framework PHPUnit je již součástí PhpStormu.

```
<?php
include 'Passgen.php';
class PassgenTest extends PHPUnit_Framework_TestCase
{
   // Nastavení vstupních a výstupní proměnných
   protected $delkahesla = 9;
   protected $vysledek;

   // Aranžování testu
   public function setUp()
   {
       $this->vysledek = new Passgen($this->delkahesla,$this->vysledek);
   }

   // Provedení testu a tvrzení předpokladu
   public function testDelkyHesla()
   {
       return $this->assertSame($this->delkahesla,
       strlen($this->vysledek->passgen($this->delkahesla)));
   }
}
?>
```

Test je zaměřen na otestování metody `passgen()`, která slouží ke generování textového řetězce, jenž představuje heslo. Vstupní hodnota pro metodu `passgen()` je celé číslo, které

určuje délku výstupního řetězce. Cílem testu je ověřit, že metoda vrací textový řetězec a ve stejné délce, v jaké je požadován. Nejprve si z třídy Passgen, která obsahuje testovanou metodu, vygeneruji novou třídu TestPassgen, do níž si importuji třídu Passgen, jinak test nemůže ověřit funkčnost metody v této třídě. Poté si nastavím vstupní a výstupní proměnné s názvem `$delkahesla` a `$vysledek`, naaranžuji test metodou `setUp()`, a následně vytvořím testovací metodu s názvem `testDelkyHesla()`, ve které aplikuji tvrzení, že proměnná `$delkahesla` se rovná délce vytvořeného řetězce metodou `passgen()`. Podoba metody `passgen()` je součástí přílohy č. 8.

Správný výsledek takového testu vypadá takto:

```
Testing started at 15:33 ...
PHPUnit 3.7.21 by Sebastian Bergmann.
Time: 0 seconds, Memory: 1.75Mb
OK (1 test, 1 assertion)
Process finished with exit code 0
```

Kompletní problematice jednotkových testů se věnuje autor Roy Osherove ve své publikaci *the Art of Unit Testing* (Osherove, 2014). Kniha je velmi čtivá, zábavná a obsahuje praktické příklady a pohled na testování jednotek.

7.2 Funkční testy

Oproti jednotkovým testům, ve kterých se postupně testují samostatné třídy a jejich metody, mají *funkční testy*, zkráceně FAT z angl. Factory Acceptance Test, mají předepsané sady vstupů a na základě těchto vstupů ověřují, že aplikace vrací správné výstupy. *„Tester předkládá softwaru připravená vstupní data s tím, že dle specifikace předem zná výstupní data, a ověřuje, že software tato výstupní data opravdu vytvořil“* (Míka, 2013, str. 7). Specifikace je zadání zákazníka s výčtem funkcí, kterými má výsledný software disponovat, a jaké jsou očekávané výsledky těchto funkcí.

Testerem připravená vstupní data jsou vytvářena na základě kombinací, se kterými se aplikace může setkat. Počtů kombinací vstupních dat, které lze aplikaci předložit, je obrovské množství, proto není možné v této fázi testování ověřit všechny kombinace vstupních dat a musíme vybrat jen takovou množinu testů, která adekvátně odpovídá pravděpodobným vstupům při běžném používání aplikace. Z tohoto důvodu jsou funkční

testy nejrozsáhlejší a nejnáročnější fází celého procesu testování a v této fázi se odhalí největší množství chyb.

Netestuje se pouze správné fungování implementovaných funkcí samotné aplikace, ale testuje se také, zda má aplikace opravdu i funkce, které jsou vyžadovány zákazníkem.

V případě vlastního databázového rozhraní slovníkové aplikace se zaměřuji na testování:

a) validity zdrojového kódu
b) kontrole funkčních odkazů
c) kontrole formulářových polí

Cílem ověření validity zdrojového kódu je prokázat, že se v rozhraní slovníku nevyskytuje syntaktická chyba nebo nechybí část zdrojového kódu, protože v případě chybějící párové značky HTML bude rozhraní slovníku fungovat. Pro ověření využívám nejznámější validační službu *Markup Validation Service* vytvořenou World Wide Web konsorciem, známým pod zkratkou W3C. Služba je dostupná online na adrese https://validator.w3.org a umožňuje validaci HTML, CSS, SVG a jiných dokumentů v nejběžnějších verzích.

Ke kontrole funkčních odkazů lze rovněž využít online nástroj *Online Broken Link Checker*, dostupný na adrese www.brokenlinkcheck.com, nebo sofistikovanější nástroj *Xenu*, který nekontroluje pouze aktivní odkazy, jež se zobrazují uživateli, ale i validitu odkazů na externí zdroje a soubory, jimiž mohou být styly, knihovny, obrázky nebo videa, atp. Xenu test odhalí, zda jsou všechny odkazy správně nadefinované, aktivní a funkční. V případě pozitivního výsledku je jisté, že se uživatel aplikace vždy dostane tam, kam vede odkaz. Nástroj Xenu je freeware, dostupný zdarma na dobu neurčitou a bez nutné registrace.

Další částí funkčního testování je kontrola formulářových polí. Kontrola spočívá v ověření správné reakce formulářových polí na vstupní data od uživatele, např., že formulář správně reaguje na špatně vyplněný tvar pole jako v případě emailu. Test je zaměřen i na ověření funkčnosti formuláře, zda proběhne validace formulářových polí a odeslání. K tomuto typu testu existují softwarové nástroje v podobě doplňků pro prohlížeč. Nejznámějším doplňkem je *Firefox Web Developer Toolbar*. Nástroj mě příliš nenadchnul, přestože má mnoho zajímavých funkcí, např. nabízí zdrojový kód k formulářovým polím, jehož je vhodný doplnit, aby byla zaručena stoprocentní funkčnost formulářového prvku s prohlížečem.

Dalším nástrojem pro realizaci funkčních testů je *Selenium IDE*, také v podobě doplňku pro prohlížeč Firefox. Nástroj umožňuje sestavení testu pomocí nahrávání vlastního postupu při procházení databázovým rozhraním. Po ukončení nahrávání se zapíší jednotlivé kroky jako itinerář celého testu. Takto sestavený test se následně spustí s hodnotami, které byly zaznamenány při nahrávání testu, čímž se zjistí, zda rozhraní slovníku vykazuje očekávané hodnoty při změně vizuální, nebo funkční části aplikace.

Sestavený funkční test je součástí přílohy č. 11. V ukázce testu ověřuji, jestli probíhá kontrola zadaných hodnot do formuláře určeného pro registraci uživatele a zda je v případě zadání neplatných hodnot uživatel vyzván k opravě údajů.

7.3 Penetrační testy

Poslední etapou procesu testování jsou penetrační testy. Smysl těchto testů spočívá v testování stability a odhalení neošetřených chyb ve zdrojovém kódu aplikace, aby případný útočník nemohl implementovat vlastní část kódu, tzv. *exploit*. Exploit jsou programy nebo příkazy, kterými útočník vykonává jinou než původně zamýšlenou činnost napadené aplikace.

Tyto testy předchází potenciálnímu úniku informací v podobě osobních údajů, know-how a dalších citlivých, důvěrných a zneužitelných informací. Testy zároveň zvyšují důvěryhodnost zákazníků nebo uživatelů, kteří aplikaci využívají. Realizace testů je prováděna externím subjektem, jenž se snaží do aplikace proniknout různými způsoby.

Testy mohou mít podobu:

a) Zadávání falešných přihlašovacích údajů – zjišťujeme, zda je aplikace chráněna proti *botům*[8] a zda jsou nastaveny maximální limity pro opakované přihlašování.
b) Odposlechu informací v případě nezabezpečeného přenosu – takové informace lze analyzovat a získat přihlašovací údaje uživatelů nebo ke správě aplikace.
c) Simulace botů, které se snaží proniknout skrze formulářová pole – jestliže nejsou formulářová pole maximálně zabezpečena, může být zákazník cílem spamu.
d) Získání informací pomocí *SQL injection* – test spočívá v zadávání SQL dotazů do formulářových polí, čímž se testuje, zda jsou formulářová pole dostatečně ošetřena, a útočník nemůže získat citlivé informace z databáze vlastním SQL příkazem.

[8] Boti jsou robotické programy, které mají jistou míru autonomie, prozkoumávají internet a získávají obsah, ke kterému by za normálních okolností neměly mít přístup.

e) Zadávání částí JavaScriptového kódu do formulářových polí – test je nazýván *XSS* z angl. cross site scripting. Útočník zkouší vložit vlastní kód do aplikace.

Penetrační testy se doporučují opakovat v rozumném časovém horizontu, neboť s rychlým vývojem technologií se objevují nové metody, jak zneužít bezpečnostní chyby v aplikaci, která již penetračními testy prošla.

K otestování vlastního databázového rozhraní penetračními testy jsem využil online nástroj *Detectify: Go Hack Yourself*, který je dostupný na adrese http://detectify.com. Nástroj nabízí po dobu 21dní monitoring databázového rozhraní na vlastním webhostingu přístupném veřejně. Webhosting je nejprve třeba ověřit, to lze například pomocí DNS záznamy, meta značkou v hlavičce HTML dokumentu, nebo zkopírováním vygenerovaného souboru do webhostingu. Po ověření je dostupný test, který se skládá celkem ze 7 fází. Počáteční fáze slouží k získávání co nejvíce možných informací o zadané doméně a tyto informace jsou následně použity pro předposlední fázi, ve které se aplikují penetrační testy a odhaluje se zranitelnost aplikace. V poslední fázi se nachází dostupný přehled analýzy se seznamem zranitelných míst s označením závažnosti. Celý test trvá několik hodin, záleží na rozsáhlosti aplikace.

Pokud jsem si jist, že má aplikace nemá zranitelná místa, mohu nechat své rozhraní otestovat zkušenými programátory v *Bounty programu*. Princip Bounty programu spočívá v umístění nabídky pro programátory, kteří se snaží v aplikaci odhalit slabá místa. V případě, že se to žádnému z programátorů nepodaří, je aplikace bezpečná. V jiném případě programátor popíše zranitelné místo, které využil, a získá svou odměnu.

8 Závěr

Cílem práce bylo analyzovat existující řešení počítačových slovníků a dle analýzy navrhnout vlastní řešení, které doplní zjištěné nedostatky, následně vlastní řešení implementovat a popsat průběh implementace, otestovat a vysvětlit význam těchto testů. Dílčím cílem práce bylo popsat, v jakých krocích probíhá vývoj databázového rozhraní, jaké technologie lze k vývoji použít a pomocí jakých metod rozhraní vyvíjet.

8.1 Zhodnocení dosažení cílů

Analýza prokázala, že žádný z dostupných řešení počítačových slovníků nevyhovuje sledovaným parametrům uvedeným v kapitole č. 1.1, a proto jsem navrhnul vlastní řešení, které přináší uživatelům více funkcí a možností, jak s počítačovým slovníkem pracovat.

Vlastní návrh rozhraní je obohacen o funkce, registraci uživatele, přidávání pojmů do databáze slovníku, využití štítků k seskupování pojmů podle vlastních kritérií, hodnocení pojmu, hlášení chyb a rozšířené možnosti vyhledávání, u kterého lze vyhledávat podle názvu pojmu, kategorií, štítku a jména autora. Vyhledávání je kombinováno v nalezení přesného výskytu výrazu, případně mnohonásobného výskytu s fulltextovým vyhledáváním. Vlastní návrh rozhraní slovníku tedy splňuje sledované parametry.

Práce je strukturovaná do kapitol, které čtenáři prezentují jako hlavní body vývoje databázového rozhraní. V každé z kapitol popisuji základní problematiku, analyzuji dostupné možnosti a na závěr předkládám vlastní rozhodnutí. Práce tak nabízí čtenáři úvodní vhled do problematiky vývoje databázového rozhraní s možnostmi, které může využít ve vlastním vývoji rozhraní.

8.2 Výstupy práce

Hlavním výstupem práce je funkční databázové rozhraní pro počítačový slovník, které bylo vyvíjeno v souladu s touto prací. Rozhraní je dostupné online na adrese `http://pocitacovyslovnik.cz`. K prozkoumání funkcí je vytvořen testovací uživatel, který má přihlašovací údaje: `test@slovnik.cz` a heslo `Kolecko123`.

Vzhledem k použitým technologiím pro vývoj slovníkové aplikace má tato aplikace potenciál se dál vyvíjet. Rozhraní může být rozšířeno o další možnosti např. o vyhledávání podle času vytvoření pojmu, nebo může být slovník dostupný jako mobilní aplikace.

Databázové rozhraní pro počítačový slovník bude i nadále vyvíjeno a rozšiřováno o další funkce pro uživatele, budou přidávány další pojmy s výkladem, aby online slovník splnil své cíle stát se nejvyhledávanější a největší databází pojmů v oboru informačních a komunikačních technologií.

8.3 Závěrečné shrnutí práce

V této práci jsem měl možnost rozšířit a prohloubit si své znalosti týkající se problematiky vývoje databázového rozhraní, a navázat tak na znalosti získané během studia. Při řešení vývoje a implementace databázového rozhraní jsem získal praktické zkušenosti s nastavením a prácí s vývojovým prostředím PhpStorm, které jsem si po chvilce oblíbil. Zkušenosti jsem získal i při řešení potíží s implementací databáze a databázového rozhraní, kdy jsem musel nastudovat dokumentaci a problém vyřešit. Naštěstí se nevyskytl žádný problém, který by byl pro implementaci vlastního rozhraní fatální.

Jako nejzajímavější část práce shledávám kapitolu věnovanou testování, ve které jsem si nastudoval a následně prakticky vyzkoušel, jak se testy vytvářejí a jak funguje jejich aplikace v praxi.

9 Seznam použitých informačních zdrojů

1. WINKLER, Petr. *Velký počítačový lexikon.* 1. vyd. Praha: Computer Press a.s., 2009. ISBN: 978-80-251-2331-7
2. VORÁČEK, Rudolf. *Slovník počítačových pojmů a zkratek.* 2. vyd. Praha: Fortuna, 1998. ISBN: 80-7168-590-9
3. ŘÍHA, Petr. *Slovník počítačové informatiky: Výkladový slovník pro práci s informacemi.* 1. vyd. Ostrava: MONTANEX a.s., 2002. ISBN: 80-7225-083-3
4. NÁDBĚLA, Josef. *Velký počítačový slovník.* 2. vyd. Kralice na Hané: Computer Media s.r.o., 2006. ISBN: 80-86686-56-6
5. Creative Commons Česká republika. *Úvod do CC* [Online]. © 2016 [cit. 2016-02-22] Dostupné z: <http://www.creativecommons.cz/uvod/>
6. ŠEDA, Miloš. *Databázové systémy*. Brno: VUT, 2002. Dostupné z: <http://www.uai.fme.vutbr.cz/~mseda/DBS02_BS.pdf>
7. CELKO, Joe. *Complete guide to NoSQL: What every SQL professional Leeds to know about Nonrelational databses.* 1. st. Waltham: Elsevier Inc., 2014. ISBN: 978-0-12-407192-6
8. DATE, J. Christopher. *SQL and Relational Theory: How to Write Accurate SQL Code.* 2. nd. Sebastopol: O'Reilly Media Inc., 2011. ISBN: 978-1-449-31640-2
9. KOŠÁREK, Lukáš. *Výkonnostní srovnání relačních databází.* Bakalářská práce. Masarykova Univerzita, Fakulta informatiky, Brno 2010.
10. LINDSTROM, Jan. *Performance evaluation of MariaDB 10.1 and MySQL 5.7.4-labs-tplc* [Online]. © 2014 MariaDB Foundation [cit. 206-02-20] Dostupné z: <https://mariadb.org/performance-evaluation-of-mariadb-10-1-and-mysql-5-7-4-labs-tplc/>
11. SCHWENKE, Axel. *MariaDB 10.1 and MySQL 5.7 performance on commodity hardware* [Online]. © 2015 MariaDB Foundation [cit. 2016-02-20] Dostupné z: <https://mariadb.org/maria-10-1-mysql-5-7-commodity-hardware/>
12. Percona LLC. *The Percona XtraDB Storage Engine* [Online]. © 2016 [cit. 2016-02-20] Dostupné z: <https://www.percona.com/doc/percona-server/5.5/percona_xtradb.html?id=Percona-XtraDB>
13. MariaDB Corporation. *Aria Storage Engine* [Online]. © 2010 [cit. 2016-02-20] Dostupné z: <https://mariadb.com/kb/en/mariadb/aria-storage-engine/>

14. RUTTER, Jake. *SMASHING jQuery*. 1 st. West Sussex: John Wiley & Sons Ltd., 2011. ISBN: 978-0-470-97723-1
15. DAVID, Matthew. *HTML5: Design Rich Internet Applications*. 2. nd. Indiana: Focal Press, 2013. ISBN: 978-0-240-82076-7
16. RIEHLE, Dirk. *Framework Design: A Role Modeling Approach.* Ph.D. Thesis, No. 13509. Zürich, Switzerland, ETH Zürich, 2000. Dostupné z: <http://dirkriehle.com/computer-science/research/dissertation/diss-a4.pdf>
17. WIKIPEDIA CONTRIBUTORS. *Framework* [online]. © 2016 [cit. 2016-02-21]. Dostupné z: <https://en.wikipedia.org/wiki/Framework>
18. W3Techs. *Usage of server-side programming languages for websites* [Online]. © 2016 Q-Success [cit. 2016-02-27]. Dostupné z: <http://w3techs.com/technologies/overview/programming_language/all>
19. SKVORC, Bruno. *The Best PHP Framework for 2015: SitePoint Survey Results* [Online]. © 2015 SitePoint Pty. Ltd. [cit. 2016-02-21] Dostupné z: <http://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/>
20. SUKUPČÁK, Michal. *Synopsy PHP Framework.* Diplomová práce. Masarykova Univerzita, Fakulta informatiky, Brno 2014.
21. JALLOUL, Ghinwa. *ULM by Example.* 1. st. Cambridge: CAMBRIDGE UNIVERSITY PRESS, 2004. ISBN: 0-521-00881-6
22. MariaDB. *Knowledge Base: SQL_MODE* [Online] © 2012 MariaDB [cit. 2016-04-12] Dostupné z: <https://mariadb.com/kb/en/mariadb/sql_mode/>
23. CHACON, Scott a STRAUB, Ben. Pro Git [Online]. © 2016 Apress [cit. 2016-04-12] Dostupné z: < https://progit2.s3.amazonaws.com/en/2016-03-22-f3531/progit-en.1084.pdf>
24. YOUNG, Blake. *Industry Stats: Project Methodologies 2011* [Online]. © 2012 Planit [cit. 2016-04-12] Dostupné z: <https://www.planittesting.com/au/Insights/2012/Industry-Stats-Project-Methodologies-2011>
25. YOUNG, Blake. *Industry Stats: Project Methodologies 2013* [Online]. © 2013 Planit [cit. 2016-04-12] Dostupné z: <https://www.planittesting.com/us/Insights/2013/Industry-Stats-Project-Methodologies-2013>

26. PADDA, Sheilly. *Review of Software Development Methodologies Used in Software Design* [Online]. © 2014 Warse [cit. 2016-04-12] ISSN: 2278-3091 Dostupné z: < http://warse.org/pdfs/2014/ijatcse02352014.pdf>
27. Agile Alliance. Manifesto for Agile Software Development [Online]. © 2001 Agile Alliance [cit. 2016-04-12]. Dostupné z: <http://agilemanifesto.org/>
28. POPELKA, Vladimír. *Srovnávací analýza metodik vývoje software*. Bakalářská práce. Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky, Praha 2014.
29. BECK, Kent. Extreme Programming Explained [Online]. © 1999 Ken Beck [cit. 2016-04-12] ISBN: 0201616416 Dostupné online z: <http://software2012team23.googlecode.com/git-history/5127389d21813c2bd955c53999f66cede994578b/docs/literature/Extreme_Programming_Explained_Kent_Beck_1999.pdf>
30. OSHEROVE, Roy. *The Art of UNIT TESTING*. 2. nd. Shelter Island: Manning Publications Co., 2014. ISBN: 978-1-617290-89-3
31. SKVORC, Bruno. *Best PHP IDE in 2014 – Survey Results* [Online]. © 2014 SitePoint Pty. Ltd. [cit. 2016-03-18] Dostupné z: <http://www.sitepoint.com/best-php-ide-2014-survey-results/>
32. MÍKA, Tomáš. *Návrh nového přístupu k řízení kvality při vývoji software*. Diplomová práce. Brno: Masarykova univerzita, Fakulta informatiky, 2013.
33. HLAVA, Tomáš. *Fáze a úrovně provádění testů* [Online]. © 2011 Tomáš Hlava. [cit. 2016-03-19] Dostupné z: <http://testovanisoftwaru.cz/category/metodika-testovani/>

10 Seznam příloh

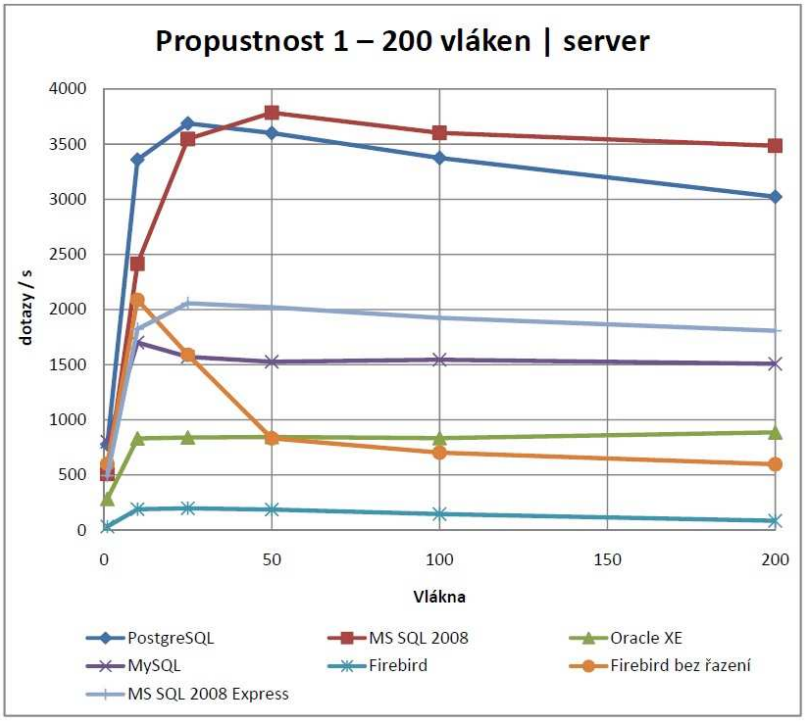
1. *DB-Engines Ranking.* Zdroj: http://db-engines.com/en/rating
2. *Propustnost relačních SQL systémů 1 – 200 vláken na serveru.* Zdroj Košárek, 2010
3. *Usage of server-side programming languages for websites.* Zdroj: W3Techs: http://w3techs.com/technologies/overview/programming_language/all
4. *PHP Framework Popularity in Personal Projects.* Zdroj: SitePoint: http://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/
5. *Konceptuální model – ER diagram databáze slovníkové aplikace.* Zdroj: Autor
6. *Logický model databáze slovníkové aplikace.* Zdroj: Autor
7. *Best PHP IDE in 2014 – Survey Results: Personal choice.* Zdroj: SitePoint http://www.sitepoint.com/best-php-ide-2014-survey-results/
8. *Podoba třídy Passgen s metodou passgen().* Zdroj: Autor
9. *Planit Testing Index 2011: Project Methodologies*. Zdroj: Planit https://www.planittesting.com/au/Insights/2012/Industry-Stats-Project-Methodologies-2011
10. *Planit Testing Index 2013: Software Development Projects by Methodology*. Zdroj: Planit https://www.planittesting.com/us/Insights/2013/Industry-Stats-Project-Methodologies-2013
11. *Funkční test formuláře k registraci uživatele provedený v Selenium IDE.* Zdroj: Autor

11 Přílohy

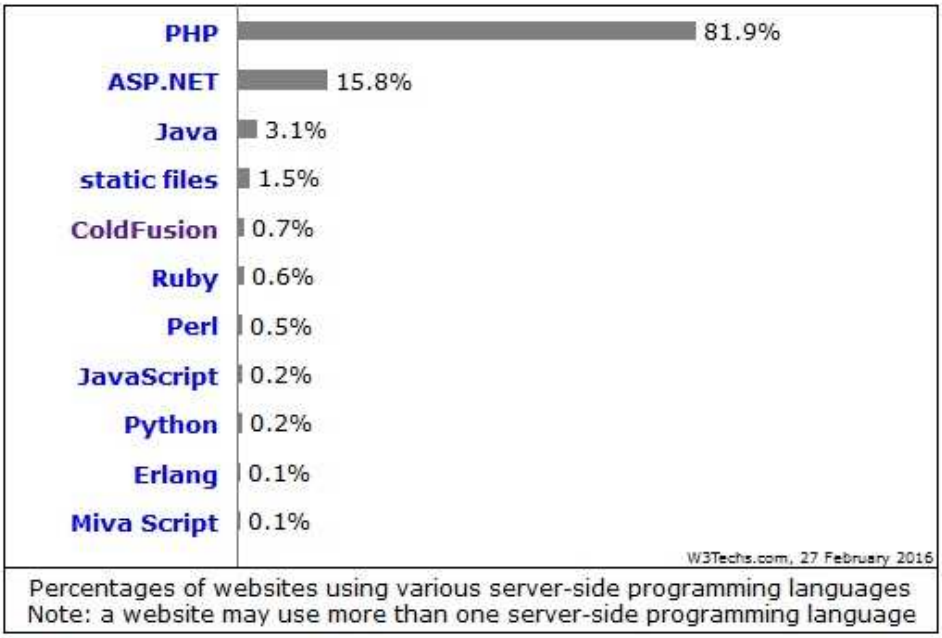
303 systems in ranking, April 2016

Rank Apr 2016	Rank Mar 2016	Rank Apr 2015	DBMS	Database Model	Score Apr 2016	Score Mar 2016	Score Apr 2015
1.	1.	1.	Oracle	Relational DBMS	1467.53	-4.48	+21.40
2.	2.	2.	MySQL	Relational DBMS	1370.11	+22.39	+85.53
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1135.05	-1.45	-14.07
4.	4.	4.	MongoDB	Document store	312.44	+7.11	+33.85
5.	5.	5.	PostgreSQL	Relational DBMS	303.73	+4.10	+35.41
6.	6.	6.	DB2	Relational DBMS	184.08	-3.85	-13.56
7.	7.	7.	Microsoft Access	Relational DBMS	131.97	-3.06	-10.22
8.	8.	8.	Cassandra	Wide column store	129.67	-0.66	+24.78
9.	9.	↑ 10.	Redis	Key-value store	111.24	+5.02	+16.69
10.	10.	↓ 9.	SQLite	Relational DBMS	107.96	+2.19	+5.67
11.	11.	↑ 14.	Elasticsearch	Search engine	82.58	+2.41	+17.92
12.	12.	↓ 11.	SAP Adaptive Server	Relational DBMS	73.32	-3.33	-13.37
13.	13.	13.	Teradata	Relational DBMS	72.26	-1.81	+2.00
14.	14.	↓ 12.	Solr	Search engine	66.02	-3.35	-15.98
15.	15.	15.	HBase	Wide column store	51.49	-0.92	-9.65
16.	16.	↑ 17.	Hive	Relational DBMS	49.08	-1.43	+6.33
17.	17.	↓ 16.	FileMaker	Relational DBMS	46.10	-1.83	-5.72
18.	18.	18.	Splunk	Search engine	42.35	-1.38	+4.32
19.	19.	↑ 21.	SAP HANA	Relational DBMS	40.35	+0.36	+7.01
20.	20.	↑ 22.	Neo4j	Graph DBMS	31.91	-0.44	+3.50
21.	↑ 22.	↑ 25.	MariaDB	Relational DBMS	31.58	+1.70	+9.19

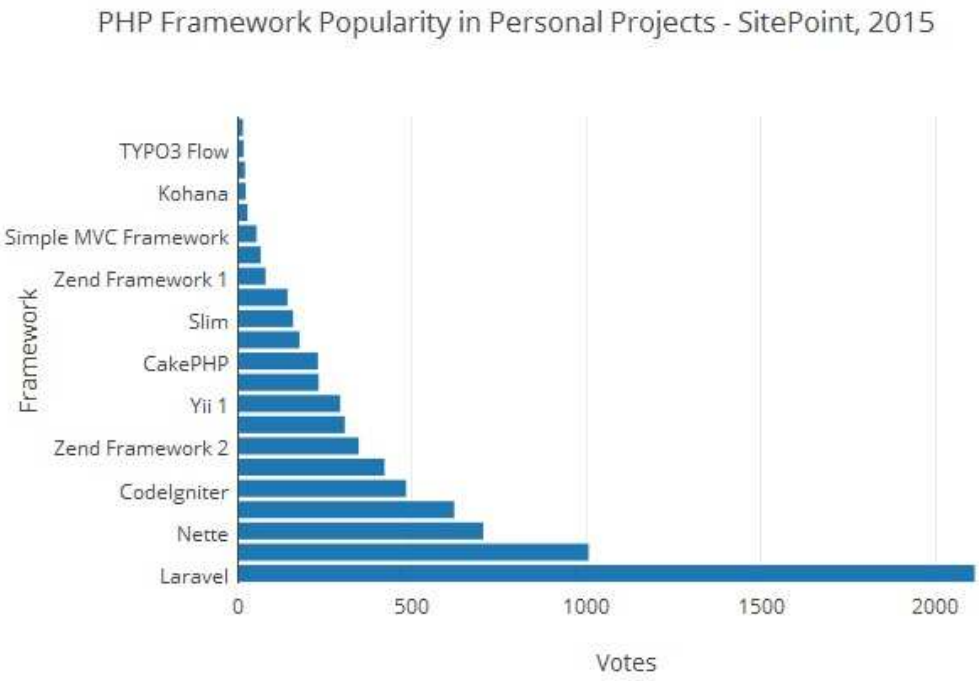
Příloha č. 1 DB-Engines Ranking [cit. 2016-04-10]. Zdroj: http://db-engines.com/en/rating



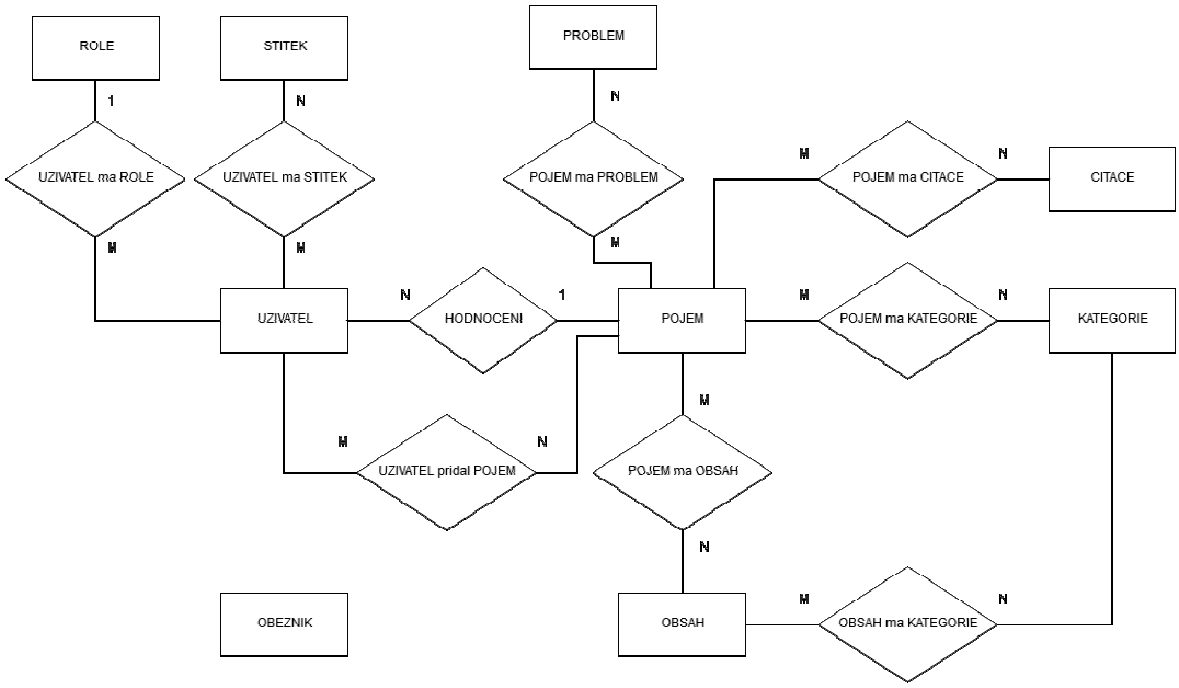
Příloha č. 2, Propustnost relačních SQL systémů 1 – 200 vláken na serveru. Zdroj Košárek, 2010



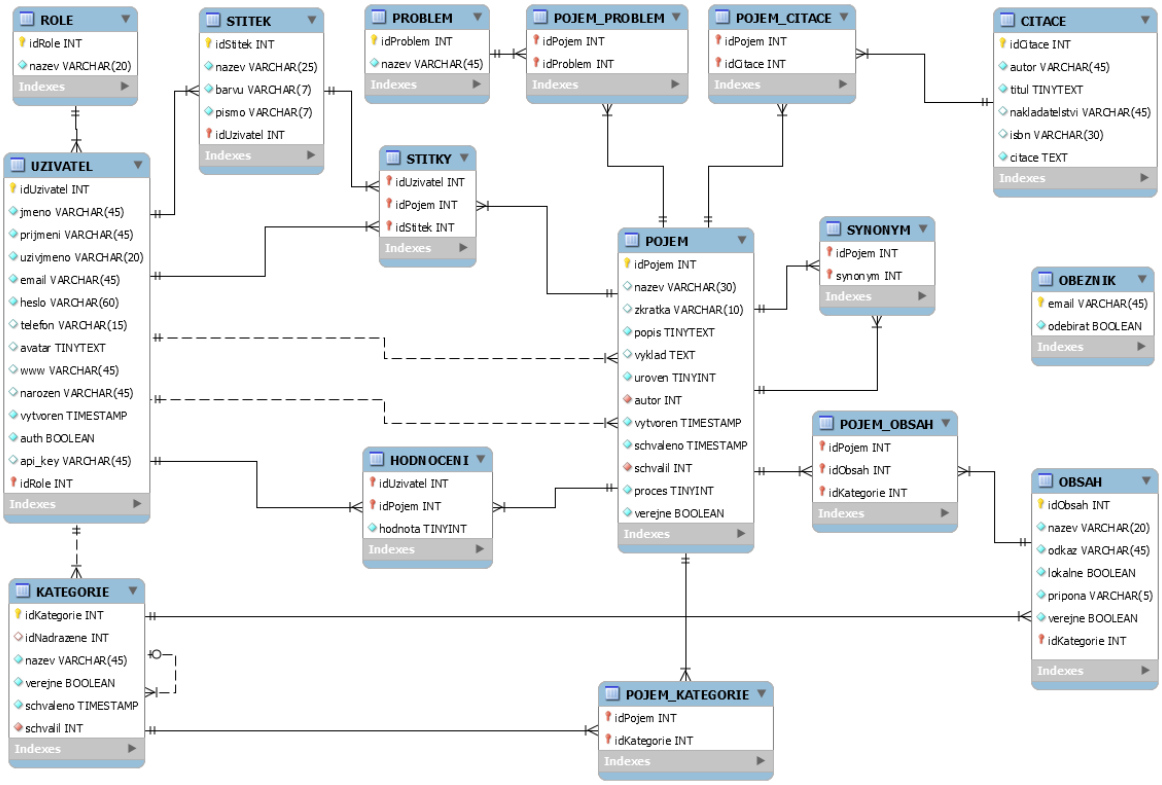
Příloha č. 3, Usage of server-side programming languages for websites [cit, 2016-02-27]. Zdroj: W3Techs: http://w3techs.com/technologies/overview/programming_language/all



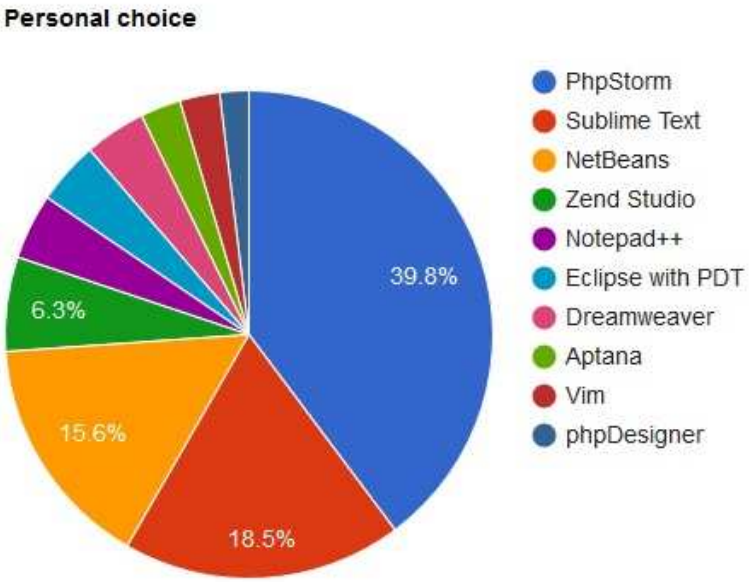
Příloha č. 4, PHP Framework Popularity in Personal Projects [cit. 2016-02-22]. Zdroj: SitePoint: http://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/



Příloha č. 5, Konceptuální model – ER diagram databáze slovníkové aplikace. Zdroj: Autor



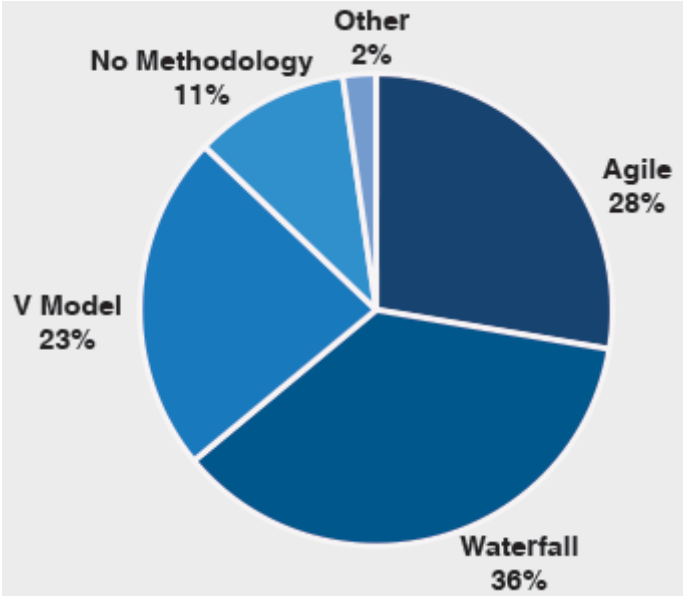
Příloha č. 6, Logický model databáze slovníkové aplikace. Zdroj: Autor



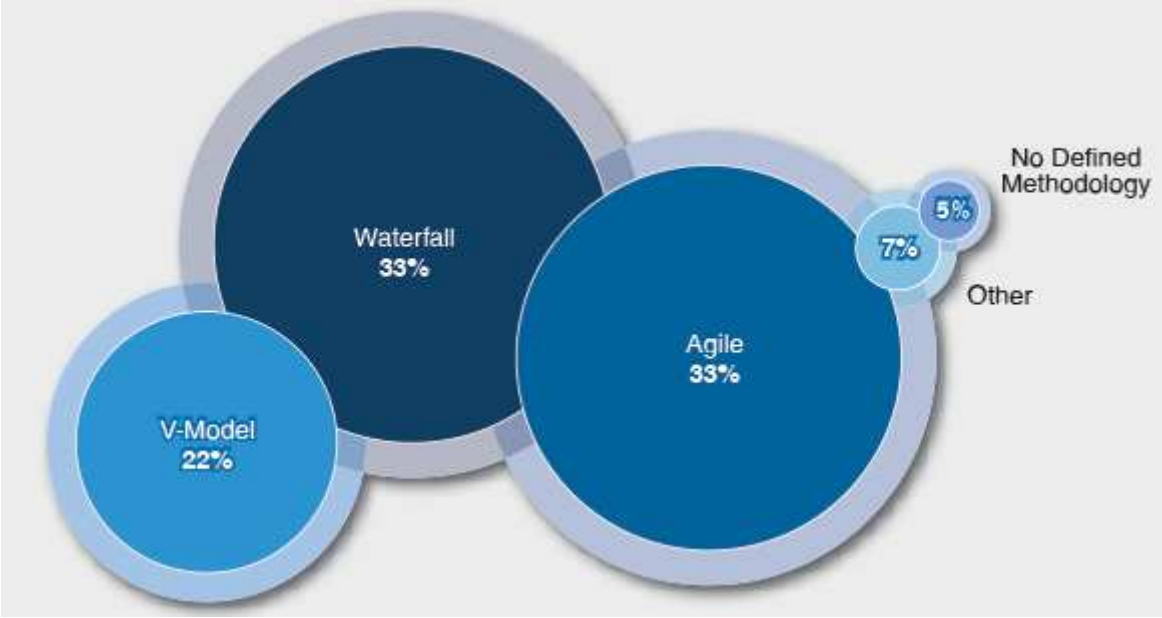
Příloha č. 7, Best PHP IDE in 2014 – Survey Results: Personal choice [cit. 2016-03-18]. Zdroj: SitePoint http://www.sitepoint.com/best-php-ide-2014-survey-results/

```php
<?php
class Passgen
{
    public function passgen($length, $keyspace =
'.,/?&#@+0123456789abcdefghijkmnopqrstuvwxABCDEFGHJKLMNOPQRSTUVWX')
    {
        $str = '';
        $max = mb_strlen($keyspace, '8bit');
        for ($i = 0; $i < $length; ++$i) {
        $str .= $keyspace[mt_rand(0, $max)];
        }
        return $str;
    }
}
?>
```

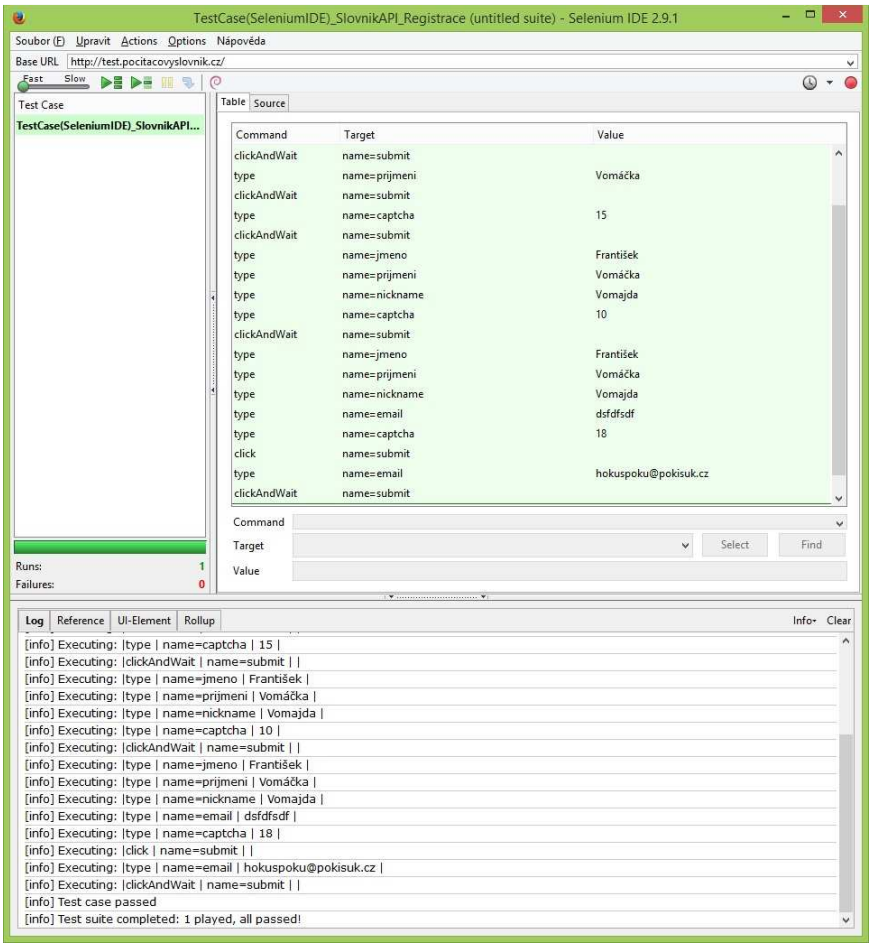
Příloha č. 8, Pohoda třídy Passgen a její metody passgen(). Zdroj Autor



Příloha č. 9, Planit Testing Index 2011: Project Methodologies. [cit. 2016-04-12] Zdroj: Planit https://www.planittesting.com/au/Insights/2012/Industry-Stats-Project-Methodologies-2011



Příloha č. 10, Planit Testing Index 2013: Software Development Projects by Methodology. [cit. 2016-04-12] Zdroj: Planit https://www.planittesting.com/us/Insights/2013/Industry-Stats-Project-Methodologies-2013



Příloha č. 11, Funkční test formuláře k registraci uživatele provedený v Selenium IDE. Zdroj: Autor