VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DOPLNĚK VISUAL STUDIO CODE PRO KONTROLU PRAVOPISU

VISUAL STUDIO CODE SPELL CHECKER EXTENSION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **DENIS GERGURI**
AUTHOR

VEDOUCÍ PRÁCE **Ing. PAVEL SVOBODA,**
SUPERVISOR

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Gerguri Denis**
Obor: Informační technologie
Téma: **Doplněk Visual Studio Code pro kontrolu pravopisu**
Visual Studio Code Spell Checker Extension
Kategorie: Uživatelská rozhraní

Pokyny:

1. Prostudujte rozhraní VS Code pro psaní doplňků
2. Vytvořte přehled dostupných knihoven pro kontrolu pravopisu (aspell, hunspell)
3. Vyberte jednu z knihoven a implementujte jednoduchou kontrolu pravopisu
4. Implementujte možnost nekontrolovat klíčová slova jazyka (C++,Latex, popřípadě dle zvážení)
5. Vytvořte stručnou dokumentaci a návod jak doplněk používat
6. Zamyslete se nad nedostatky a možnosti jejich řešení

Literatura:

- https://code.visualstudio.com/Docs/extensions/overview
- https://github.com/Microsoft/vscode-spell-check
- https://github.com/Jason-Rev/vscode-spell-checker

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Svoboda Pavel, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato bakalářská práce se zabývá vytvořením doplňku kontroly pravopisu pro textový editor zdrojových kódu Microsoft Visual Code. Práce popisuje historii kontroly pravopisu, dnešní volně nejpoužívanější knihovny a samotnou implementaci a publikaci doplňku kontroly pravopisu a jeho případném zlepšení.

Abstract

This bachelor's thesis deals with creation of spell checking extension for source code text editor Microsoft Visual Code. This papper describes the history of the spell checking, today's most widely used libraries and the actual implementation and publication of the spell checker and its possible improvement.

Klíčová slova

kontrola, pravopis, doplňek

Keywords

spell, check, extension

Citace

GERGURI, Denis. *Doplněk Visual Studio Code pro kontrolu pravopisu.* Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Svoboda Pavel.

Doplněk Visual Studio Code pro kontrolu pravopisu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Pavla Svobody. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Denis Gerguri
18. května 2017

Poděkování

Děkuji, panu Ing. Pavlu Svobodovi za vedení mojí bakalářské práce.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Jak všichni jistě víme, lidé dnes využívají počítačové programy, protože zjednodušují náš každodenní život a šetří čas. Jedním z těchto programů může být kontrola pravopisu. Člověk není tvor neomylný, a proto pomoct si počítačem u kontroly při každodenním psaní, může být značně polehčující okolnost. V této bakalářské práci se nejprve seznámíme se základními pojmy jako doplněk a kontrola pravopisu.

Ukážeme si, jak může být tato kontrola složitá a pokusíme se vybrat vhodné nástroje pro implementaci jednoduchého doplňku kontroly pravopisu pro textový editor zdrojových kódu Microsoft Visual Code.

Dále se pokusíme implementovat některé pokročilé funkce a tento výsledný produkt publikovat na repozitář doplňků, kde si jej budou mít možnost stáhnout uživatelé z celého světa.

Budeme se snažit navrhnout myšlenky a postupy pro případné budoucí zlepšení.

Kapitola 2

Základní pojmy

2.1 Program pro kontrolu pravopisu

Neboli anglicky *spell checker* je v počítačové terminologii označován takový program, který v textovém dokumentu označí za chybná ta slova, která nejsou napsaná pravopisně. S postupem času se tyto programy vyvíjely do mnohem sofistikovanějších verzí, které jsou schopné nejenom označit a navrhnout správné slovo, ale také označit celé věty nebo větší části textu na základě špatně použitého slova v kontextu.

2.2 Doplněk

Bývá anglicky označován jako *extension* nebo *plugin*. Je to malý kus programu, který upravuje nebo obohacuje funkcionalitu hlavního programu. Doplněk většinou přímo nebývá součástí hlavního programu, protože účelné využití tohoto programu závisí na uživatelské preferenci a také na problému, který chce uživatel zrovna řešit. K normálnímu chodu hlavního programu není potřeba, neboť by mohl nezpřehledňovat nebo zpomalovat hlavní program.

Možnost vytvářet doplňky k programům byla vytvořena za účelem rozšíření hlavního programu externími vývojáři, a tím urychlení jeho vývoje a také z důvodu oddělení zdrojového kódu doplňku od zdrojového kódu hlavního programu na základě rozdílné licence. V praxi to může znamenat, že hlavní program, který je zcela zdarma, může mít zpoplatněný doplněk.

Doplňky dnes nejčastěji využíváme v internetových prohlížečích a textových editorech. Mezi nejpopulárnější doplňky se řadí právě kontrola pravopisu. Dále to mohou být programy pro blokaci nevyžádaných reklam na internetu, integrovaný emailový klient v prohlížeči nebo například možnost zapnutí nočního režimu, abychom zabránili únavě a poškození zraku. Nejčastěji bývají doplňky stahovány z externího repozitáře k hlavnímu programu.

2.3 Visual Studio Code

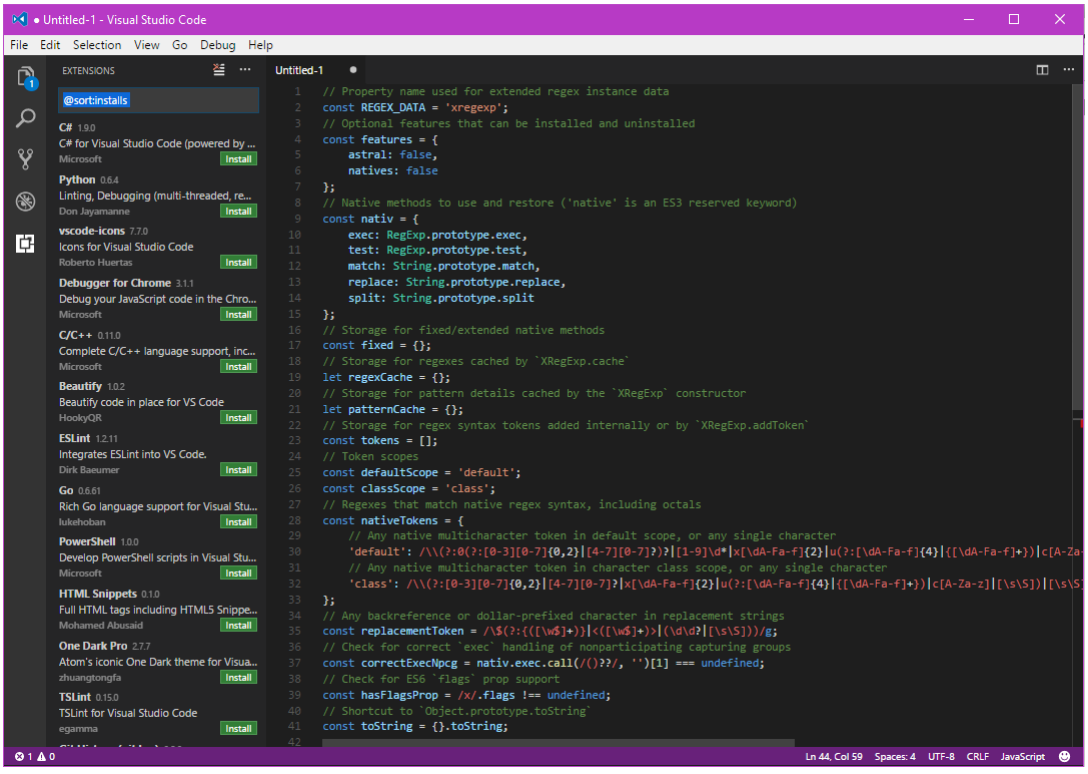
Dále jen *VSCode* je textový editor primárně určený pro editaci zdrojového kódu od firmy *Microsoft*. Je dostupný pro operační systémy *Windows*, *macOS* a *Linux*. Zdrojový kód je veřejně dostupný a licencovaný pod MIT licencí[7], která bez omezení dovoluje nakládat s tímto kódem, jak uživatel uzná za vhodné. To znamená, že tento textový editor můžete

využívat i ke komerčním účelům, nebo například na firemních počítačích zcela zdarma. Jeho první verze byla vydána v roce 2015[9].

Je postaven na *Electron* frameworku[5], což je framework pro vytváření multiplatformních aplikací pomocí *JavaScriptu*. Běhovým prostředím určeném k realizaci *JavaScript* kódu je *Node.js*. *VSCode* disponuje velikou řadou funkcí pro usnadnění editace zdrojového kódu. Jsou to například funkce pro zvýraznění syntaxe snad všech veřejně známých programovacích jazyků nebo chytré doplňování kódu pro jazyky *JavaScript*, *HTML*, *CSS* a také *TypeScript*, což je nadstavba pro *JavaScript*. Další možností rozšíření jeho funkcionality se provádí právě pomocí doplňků.

Doplňky pro *VSCode* se píší v jazyce *JavaScript* nebo *TypeScript*. Doplňky se poté publikují na repozitář, který se jmenuje *VSCode Marketplace*. Doplňky může uživatel vyhledávat pomocí rozhraní přímo v aplikaci *VSCode*, nebo na webové adrese repozitáře. Součástí každého doplňku je i jeho *README* soubor, který se chová jako hlavní strana doplňku a slouží jako dokumentace, jak doplněk funguje a jak ho používat. Registrovaní uživatelé mohou doplňky hodnotit a psát k nim recenze, připomínky a případně nedostatky nebo návrhy na vylepšení. Obrázek 2.1 zobrazuje grafické rozhraní *VSCode*.

Dle mého názoru je *VSCode* rychlý a přehledný textový editor, který obsahuje veškeré nástroje pro účinnou editaci a vývoj zdrojových kódů.



Obrázek 2.1: VSCode ver. 1.12.1

Kapitola 3

Dostupné knihovny a programy pro kontrolu pravopisu

Historie programové kontroly pravopisu sahá až do druhé poloviny minulého století. V následující kapitole srovnáme nejznámější z nich. Ukážeme si rozdíly a vybereme vhodnou knihovnu pro implementaci našeho doplňku. Jak uvidíte, kontrola pravopisu není tak triviální úkol jak by se mohlo zdát, speciálně pokud jde o podporu několika různých jazyků.

3.1 Spell

První veřejně dostupný program pro kontrolu pravopisu na *Unixových* systémech byl napsán v roce 1971. Napsal ho *Ralph Gorin* v *Assembleru* a jmenuje se *spell*. Spell však nebyl korektorem, jaké používáme dnes, nýbrž pouze jakýmsi validátorem, který jednotlivá slova textu porovnal s přiloženým slovníkem a označil ty, které slovník neobsahoval. Neumožňoval tedy návrh správných slov namísto špatně označeného tvaru.

V roce 1975 byl spell přepsán *Douglasem C. Johnsnem* do *jazyka C*, včetně několika úprav na zvýšení rychlosti [2]. Ani tyto úpravy však nebyly dostačující a proto o několik let později bylo načase tento program nahradit. Velkou nevýhodou byla pouze podpora anglického jazyka a právě neschopnost navrhnout pravděpodobné správné výrazy slov namísto překlepů. Jedním z důvodů neschopnosti podpory jiných jazyků než angličtiny byla tehdejší nedostatečně velká jazyková sada. Tehdejší počítače používaly pouze 7-bitové kódování, které umožňovalo pouze zápis malých a velkých písmen anglické abecedy. Drtivá většina dnešních programů pro kontrolu pravopisů vychází právě z programu spell. Program byl vydán pod svobodnou *BSD licencí*[3], což později umožnilo jeho rozvoj. S trochou nadsázky by se dalo říct, že všechny další programy vychází právě z něj.

3.2 Ispell

V roce 1984 byl do *Unixových* distribucí portován program s název *Ispell* (international spell checker)[14]. Jedná se o vylepšenou verzi programu spell. Na jeho tvorbě se podílelo mnoho lidí. Za zmínku stojí jeho nejvýznamnější autor *Geoff Kuening*. Mezi tehdejší přednosti ispellu patřila podpora dalších evropských jazyků díky rozšířené znakové sadě, která umožňovala psát i jiné znaky, než základní anglickou abecedu a schopnost navrhnout správná slova na základě *Damerau-Levenshteinové* délky jedna[4].

Damerau-Levenshteinová délka je taková metrika pro textové řetězce, která udává, kolika nejméně možnými operacemi je možné přetvořit jeden textový řetězec na druhý. Mezi povolené operace patří:

- *Přídání znaku do textového řetězce.*
- *Odebrání znaku z textového řetězce.*
- *Nahrazení znaku textového řetězce za jiný.*
- *Výměna dvou sousedních znaků textového řetězce. (5)*

Od *Levenshteinové* metriky se liší právě přidáním poslední operace. Podle *Dameraua* je jedna z těchto čtyř operací nejčastějším důvodem chybně napsaného slova. Právě díky aplikování této metriky do zdrojového kódu umí ispell navrhovat pravděpodobná správná slova k jejich nesprávné variantě, čímž posunul kontrolu pravopisu na novou úroveň.

Z důvodu velkého počtu slov obsažených ve slovnících jednotlivých jazyků, byl také vymyšlen nový způsob, kdy k vyhledání korektního slova byl kromě slovníku využíván také druhý soubor. Tento soubor obsahoval pravidla pro vytváření přípon slov daného jazyka. V češtině to může být například přípona na základě rodu a pádu podstatného jména. Toto vylepšení značně redukovalo počet slov obsažených ve slovníku. Namísto anglických slov *jump*, *jumped*, *jumping* a *jumps* stačilo, aby slovník obsahoval pouze slovo *jump* a jeho další varianty byly vytvořeny právě pomocí tohoto druhého souboru. Další příklad z tohoto souboru můžete vidět na obrázku 3.1.

```
SFX   y     ied         [^aeiou]y
```

Obrázek 3.1: Výňatek z anglického affix souboru

1. *SFX* Označuje příponu (anglicky suffix).
2. *y* Řetězec, který odstranit před přidáním přípony. V tomto případě pouze znak „y“.
3. *ied* Řetězec, kterým předchozí část nahradit.
4. *[^aeiou]y* Podmínka, která musí být splněna. V tomto případě poslední písmeno slova musí být „y“, a zároveň předposlední písmeno nesmí být ani jedno z těchto písmen „a“,„e“,„i“,„o“,„u“.

Takto zapsaný řádek v affix souboru má poté za následek, že k anglickému slovíčku *supply* je vytvořeno k porovnání slovo *supplied*, bez nutnosti obsahovat toto slovo ve slovníku. Mezi další kladné stránky stojí zmínit první snahu o vytvoření programového rozhraní pro tehdejší *Emacs* textové editory[**?**] a možnost nekontrolovat klíčová slova jazyka *TeX*. Jeho předposlední verze byla vydána v roce 2005. Poslední verze byla vydána v roce 2015 a nepřináší nic nového. Proto považuju program již za zastaralý a neudržovaný. Tak jako spell i ispell byl vydán pod *BSD* licencí, což umožnilo jeho další rozvoj.

3.3 MySpelll

MySpell [13] není sám o sobě program. Neobsahuje žádné rozhraní pro interakci. Je tedy pouze knihovnou určenou pro integrování do větších programů. Napsal ho *Kevin Hendricks* v programovacím jazyce *C++*. Byl vytvořen jako snaha implementovat kontrolu pravopisu do veřejně dostupného balíku programů *OpenOffice*. Díky veřejné BSD licenci byla tato knihovna využita například u programu *Aegisub*, což je program pro vytváření filmových titulků. Vychází z *Ispellu* a jeho slovníky a soubory pro vytváření přípon jsou s ním zpětně kompatibilní. Mimo výše uvedené funkce přináší několik vylepšení. Novinkou je například zavedení nahrazovacích tabulek. Tabulku můžete vidět na obrázku 3.2.

```
REP f ph
REP ph f
REP f gh
REP gh f
REP j dg
REP dg j
REP k ch
REP ch k
```

Obrázek 3.2: Příklad anglické nahrazovací tabulky

- *První sloupec* Označuje, že se jedná o nahrazení (anglicky replacement).
- *Druhý sloupec* Označuje, co má být nahrazeno.
- *Třetí sloupec* Označuje, čím má být nahrazeno.

Příkladem může být anglické slovíčko *photo*, které se vyslovuje jako *foto*. Jak vidíte, je zde snadné udělat chybu. Díky tomuto jsou navrhnuta správná slova i u slov, které jsou podle *Damerau-Levenshteinové* metriky vzdálenější než jedna. Odpovídající změna by souhlasila s metrikou vyzdálenou dva, neboť bychom museli jeden znak přidat a druhý zaměnit. Výpočet pro délku dvě by byl mnohem složitější a jeho výpočetní náročnost by převyšovala náročnost těchto nahrazovacích tabulek.

Za největší přínos považuji využití *n-gram modelů*[10]. Tyto modely využívají předchozí znalost textu daného jazyka, na základě kterého jsou schopny určovat pravděpodobnost následujícího písmene nebo slova. Navrhována slova jsou ohodnocena a ty s lepším hodnocením jsou navrhována na prvních pozicích. Jednoduchým příkladem je anglická věta: *The car is red.* V případě nechtěného nahrazení slova *is* za *ix* by mohly mezi navrhovanými slovy být *if*, *is* a *in*. Slovíčko *is* by zde mělo nejvyšší hodnocení, a proto by bylo upřednostněno. Za nevýhody považuji neschopnost přidávat vlastní slova do personalizovaného slovníku a *MySpell* bohužel také nepodporuje různé znakové sady. Od verze *OpenOffice 2.0.2* byl nahrazen za knihovnu *Hunspell*.

3.4 Hunspell

Hunspell je momentálně nejrozšířenější program a knihovna pro kontrolu pravopisu[17]. Vychází z *MySpellu*, byl napsán v roce 2002 a jeho slovníky jsou s ním zpětně kompatibilní. Díky podpoře znakové sady Unicode je možné tuto knihovnu využívat pro většinu světových

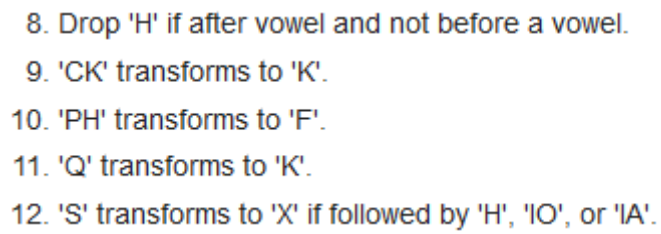
jazyků. K dnešnímu dni jich je přibližně 70. *Hunspell* je napsán v jazyce *C++* panem *László Némethem* a byl navrhnut hlavně pro jazyky s bohatou morfologií, primárně pro jazyk maďarský. Byl portován do většiny nejpoužívanějších programovacích jazyků a vydán pod více licencemi. Konkrétně jde o licence *GPL/LGPL/MPL*, které také umožňují svobodné nakládání se zdrojovými kódy. Díky tomu se stal velice populární. Dnes je používán většinou internetových prohlížečů jako například *Safari*, *Firefox*, *Chrome*, *Opera*, open-source kancelářskými balíky *OpenOffice*, *LibreOffice* a mnoha dalšími hojně využívanými programy. Kromě výše zmíněných praktik je *Hunspell* morfologický analyzér. Podle *doc. RNDr. Aleše Horáka, Ph.D.*[18] je morfologická analýza :

> *„Morfologická analýza je základním prostředkem zkoumání přirozeného jazyka a zabývá se rozlišováním a generováním správných gramatických tvarů slovních výrazů, které vzniknou skloňováním a časováním. Výsledkem je sada značek, které popisují gramatické kategorie daného tvaru, zejména pak základní tvar (lemma) a slovní vzor. Automatické rozlišení tvaru slova ve volném textu lze využít při vývoji gramatického korektoru, jako pomůcka při značkování korpusů nebo při poloautomatickém vytváření slovníků. Největší problém v této oblasti je morfologická desambiguace (zjednoznačňování gramatické značky) - tedy jak automaticky rozlišit, zda slovo jedu označuje sloveso nebo podstatné jméno.“*

Hunspell však nedokáže nalézt lemma (kmen slova), ale pouze kořen pomocí algoritmu *stemmatizace*. Jedná se o sadu jednoduchých pravidel pro odstranění koncovek. Pokud anglické slovo končí příponou *-ed* je pro získaní kořene slova odstraněna. Například slovo *loved* má kořen slovo *love*. Podporuje také skládání slov. Skládání je hojně využívané v německém a maďarském jazyce ale setkáme se s tím i v jazyce českém.

3.5 Aspell

Další velice používaný program a knihovna je *Aspell*[1]. Napsal ho *Kevin Atkinson* v jazyce *C++*. Je to další z modifikací *Ispellu*. Stejně jako *Hunspell* podporuje znakovou sadu Unicode. Slova, která nejsou obsažena ve slovníku, indexuje pomocí algoritmu zvaného *Metaphone*[8], díky kterému dosahuje excelentních výsledků pro těžce chybně napsaná slova. *Metaphone* je fonetický algoritmus navržen *Lawrencem Philipsem* v roce 1990. V principu tento algoritmus převede nekorektně napsané slovo do podoby přibližné aproximace jeho výslovnosti. K tomuto převodu využívá velkou množinu pravidel, výňatek z ní pro představu můžete vidět na obrázku 3.3.



Obrázek 3.3: Výňatek z množiny pravidel algoritmu Metaphone

Český překlad:

- Odstraň písmeno „H“ v případě, že následuje za samohláskou a zároveň, ne před samohláskou.
- „CK“ nahraď za „K“
- „PH“ nahraď za „F“
- „Q“ nahraď za „K“ item „S“ nahraď za „X“ pokud za ním následuje „H“, „IO“, nebo „IA“.

[1] poté na výsledný tvar aplikuje již výše zmíněnou *Demerau-Levenshteinovou* metriku o délce 1 případně 2. Výsledky porovná se slovníkem, odstraní možné duplikáty a vrátí podobná slova. *Lawrenc Philips* v roce 2000 navrhl modernější verzi toho algoritmu, která podporuje kromě anglického jazyka další evropské jazyky a také čínštinu. Ten samý rok byl tento algoritmus adaptován i do *Aspellu*. V roce 2009 přišel *Lawrenc* s třetí verzí toho algoritmu, který už bohužel není veřejně dostupný, a proto už nemůže být jeho součástí. Poslední verze *Aspellu* vyšla v roce 2011 pod *LGPL* licencí, která taktéž neklade žádné omezení. Dle mého názoru to je jeden z důvodu proč hunspell převzal roli nejpoužívanější knihovny. Mezi programy využívající tuto knihovnu patří *Notepad++*, *Gajim*, *LyX* a starší verze internetového prohlížeče *Opera*. Později byl v *Opeře* nahrazen právě knihovnou *Hunspell*. Podívejme se na jednoduché srovnání *Aspellu* s *Hunspellem*. Obrázek 3.4.

Správný tvar	exercised	photo	neschopný	neschopný
Hledaný tvar	excersized	foto	neschopn	nefrhopný
Aspell-Výsledek	exercised, excised, excesses, exercises, exorcised, exercisers	foot, foo, Soto, Toto	neschopen, neschopna, neschopni, neschopně, neschopný, ...	
Hunspell-Výsledek	undersized, expertized, mercerized, excerpted, exceeding, excessive	photo, foot, goto, Soto, Toto	neschopna, neschopná, neschopen, neschopné, neschopný, ...	neschopný

Obrázek 3.4: Srovnání

1. Jak můžeme vidět v prvním sloupci hledaného výrazu excersized, *Aspell* našel díky fonetickému algoritmu správný výraz exercised i pro těžce špatně napsané slovo. *Hunspell* však nikoliv.
2. V druhém sloupci se situace obrací. *Hunspell* má navrch právě díky algoritmu nahrazovacích tabulek, který převzala z *MySpellu*. Dochází k předem zmiňované záměně *ph* na *f* a hledané slovo je úspěšně nalezeno. *Aspell* je bohužel neúspěšný.

3. Třetí sloupec ukazuje, že pouhé nedopsání posledního znaku nedělá ani jednomu programu problém. Správný výraz je nalezen v obou případech.

4. Čtvrtý sloupec nahrazuje u stejného slova dva znaky za kompletně odlišné. Jak můžeme vidět foneticky algoritmus *Aspellu* si s tímto neumí poradit. Nejspíše jeho předpisy nejsou pro český jazyk vhodné.

3.6 Enchant

Jak už teď jistě víte, kontrola pravopisu není tak triviální úkol. Jednotlivá řešení mají své výhody a nevýhody a udělat knihovnu na stejném principu, která by dosahovala skvělých výsledků pro několik různých jazyků je dosti ztěžující. Některé jazyky jsou od jiných naprosto odlišné, a pokud by chtěl programátor psát sofistikovaný doplněk pro kontrolu pravopisu anglického jazyka a jazyka jidiš, musel by využívat více než jedné knihovny. Z tohoto důvodu vznikl v roce 2003 projekt *Enchant*[15].

Enchant je souhrn většiny dostupných knihoven pro kontrolu pravopisu. Hlavně těch, které jsou specifické pouze jednomu jazyku. Poskytuje *API* jak v jazyce *C*, tak *C++*. *Enchant* obsahuje výše zmiňovaný *Aspell* a *Hunspell*. Dále *Voikko* (finská kontrola pravopisu), *Zemberek* (turecká kontrola pravopisu), *Uspell* (jidiš a hebrejština). *Enchant* využívá například známý volně dostupný textový editor *gedit*. Poslední verze vyšla v roce 2010. Pro jednoduché řešení je *Enchant* zbytečně složitý a pravděpodobně z důsledku nedostatečného financování projekt mrtvý zůstane.

3.7 Nspell

Z hlediska psaní doplňku pro *VSCode* je pro nás nejzajímavější knihovna *Nspell*[19]. Autorem je *Titus Wormer* a je publikována pod *MIT* licencí, což mi umožnuje knihovnu neomezeně používat pro další vývoj mého doplňku.

Je napsána v *JavaScriptu* a obsahuje většinu přepsaného jádra knihovny *Hunspell*. Většinu znamená, že neobsahuje například výše zmíněný algoritmus pro nalezení kořene slova, který je stejně pro naše účely zcela zbytečný. Taktéž je kompatibilní se všemi slovníky knihovny *Hunspell* a jejich soubory pro vytváření předpon a přípon. Díky tomu si můžu být jistý, že doplněk bude pracovat se stejnými jazyky, které podporuje knihovna *Hunspel*l a jejíž slovníky jsou hojně a volně publikovány na internetu.

Kapitola 4

Implementace a publikace doplňku

Implementace a publikace doplňku se dělí na několik částí. Nejprve si navrhneme, jak by takový doplněk měl vypadat, a co je z hlediska jeho funkcionality a přehlednosti nezbytné.

4.1 Návrh

1. Doplněk by měl obsahovat grafické rozhraní navázané na editor *VSCode*. Toto grafické rozhraní by mělo obsahovat položky pro jednoduchou interakci s doplňkem, a to ovládací tlačítko zapnout/vypnout a ovládací tlačítko pro změnu jazyku na základě nainstalovaných slovníků.

2. Kontrolovat slova obsažené v editovaném textovém dokumentu. Tyto slova kontrolovat jednotlivě a bez interpunkce, neboť VSCode je hlavně určený k editování zdrojového kódu. Tyto chybně napsaná slova jasně a výrazně označit uživateli *VSCode* a nabídnout k nim alternativu správně napsaného slova.

3. Možnost kontrolovat zvlášť víceslovné fráze označované jako *CamelCase*, neboť je to jeden z běžných postupů při pojmenovávání proměnných v programovacích jazycích.

4. Možnost přidat chybně označená slova do slovníku a tím je označit za správná, neboť se stává, že slovník není nikdy dokonalý a nemusí i správně napsané slovo obsahovat.

5. Možnost přidat chybně označená slova do seznamu klíčových slov konkrétního typu dokumentu, aby nedocházelo ke kontrole klíčových slov programovacích jazyků. Pro typ dokumentu jazyka *JavaScript* by to například mohly být klíčové slova while a for, které sice patří mezi správná slova jazyka anglického, českého však nikoliv. Tyto úpravy provádět i manuálně bez nutnosti interakce s takto špatně označeným slovem v programu *VSCode*.

4.2 Generování základních souborů

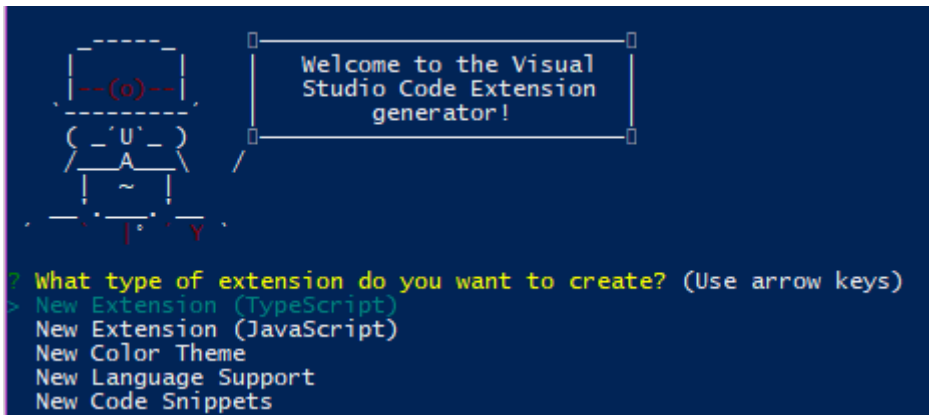
Abychom mohli doplněk vytvářet a debuggovat v prostředí *VSCode* je nutné mít nainstalovaný předem zmiňovaný *Node.js*. Všechny doplňky *VSCode* jsou publikovány formou balíčků, které jsou v prostředí *Node.js* zcela běžné. Balíčky je možné stahovat pomocí programu *NPM* (anglicky *Node Package Manager*), který je implicitně součástí *Node.js*. Jednotlivé balíčky můžou ke svému fungování vyžadovat další balíčky, které jsou automaticky instalovány z *NPM* repozitáře na základě deklarovaných závislostí. Jeden balíček může být

závislý na několika dalších a ty zase na několika dalších atd. Tento systém je zavedený jednak, aby se zabránilo zbytečnému znovu implementování stejného algoritmu nebo funkce, ale také kvůli přehlednosti.

Pro vytvoření balíčku doplňku se základní strukturou pro interakci s aplikačním rozhraním *VSCode*, použijeme Yeoman generator 4.1[12], což je program napsaný firmou *Microsoft* právě pro automatizované vytvoření šablony pro psaní doplňku. Dosáhneme toho pomocí příkazu v příkazové řádce:

```
npm install -g yo generator-code
yo code
```

Poté si zvolíme, jestli budeme náš doplněk psát v jazyce *TypeScript*, nebo v našem případě, jazyce *JavaScript*. Doplníme patřičné náležitosti jako název doplňku, identifikátor, popis doplňku a jméno autora. Program se poté automaticky postará o vytvoření patřičné adresářové struktury a stáhnutí potřebných balíčků.



Obrázek 4.1: Yeoman Extension Generator

4.3 Adresářová Struktura

V následujícím odstavci si stručně popíšeme adresářovou strukturu doplňku (4.2).

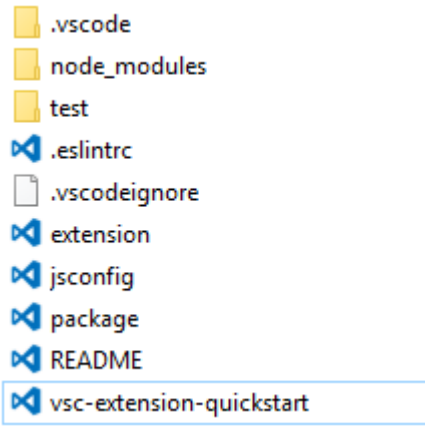
Mezi nejdůležitější položky patří package.json. Jedná se o programový manifest celého doplňku. Jsou zde uvedeny nejdůležitější informace, jako je název doplňku, popis, jméno autora, kategorie, do které doplněk spadá, rozsah verzí *VScode* se kterýma je doplněk kompatibilní a za jakých podmínek bude aktivován. Dále jsou zde například příkazy, které se mají při instalaci doplňku aktivovat a také balíčky, na kterých je doplněk závislý.

Dalším důležitým souborem je *README.md*. Je to vlastně kompletní manuál k doplňku a je vyobrazen na hlavní straně doplňku, jak na webovém depozitáři, tak při procházení doplňky v grafickém rozhraní VSCode.

Adresář *node_modules* obsahuje balíčky, na kterých je doplněk závislý. Tyto balíčky se instalují na základě informací obsažených v souboru *package.json*.

Posledním důležitým souborem je *extension.js*. Tento soubor je načten při aktivaci doplňku a obsahuje jeho zdrojový kód. Obsahuje metodu *activate*, která je volána na základě parametrů nastavených v *package.json*. Aktivace může nastat automaticky při zapnutí *VSCode* nebo například pomocí příkazu později.

Soubor *.vscodeignore* zaznamenává, jaké soubory mají být ignorovány při publikaci doplňku. Jsou zde uvedeny všechny ostatní nezmíněné adresáře a soubory. Jejich účel je zde pouze k debuggování našeho programu ve *VSCode* a jejich přítomnost nemá vliv na následnou publikaci doplňku.

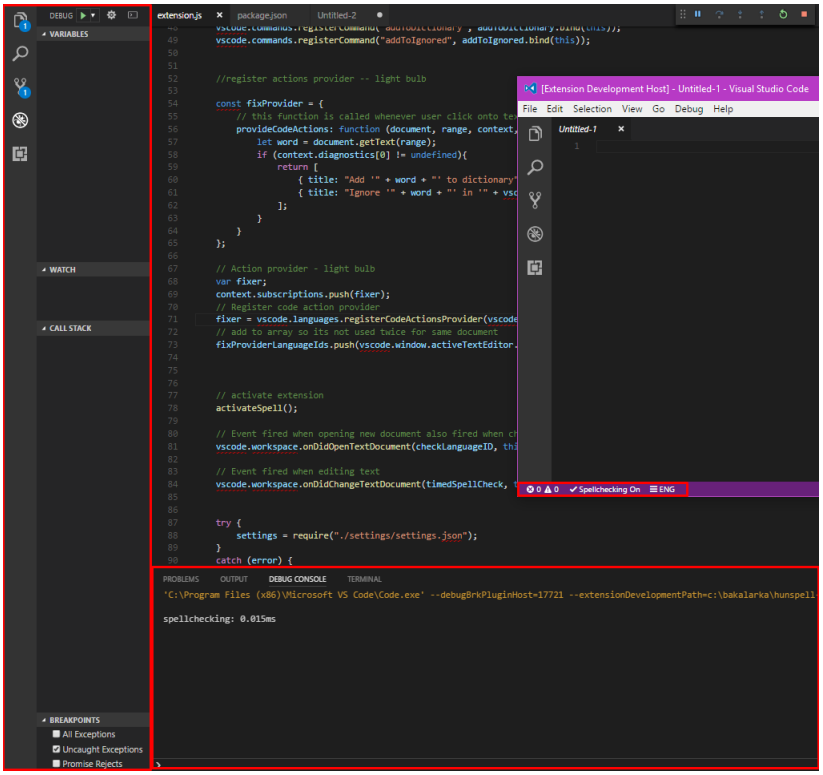


Obrázek 4.2: Adresář doplňku

4.4 Ladění doplňku

Pokud si takto vytvořený adresář otevřeme ve VSCode, můžeme svůj doplněk zároveň i ladit. Při stisknutí klávesy F5, dojde k vytvoření nového okna VSCode s aktivovaným doplňkem. Předchozí okno se přepne do režimu ladění 4.3. Integrovaný debugger, značně usnadňuje vývoj, neboť má programátor vše po ruce a je jednoduché opravit chyby.

Tento ladící program je dle mého názoru velmi přehledný. Disponuje užitečnými funkcemi jako *breakpointy*, což jsou značky, které určují, ve kterém místě zdrojového kódu se má program zastavit. Díky tomuto může programátor sledovat aktuální stav, zásobníku, proměnných a dalších prvků, které mu usnadní nalézt chybu nebo pochopit jak program funguje. Mezi další funkce patří podmíněné *breakpointy* a možnost procházení programu po jednotlivých krocích. Díky klávesovým zkratkám je ladění ještě jednodušší a rychlejší.

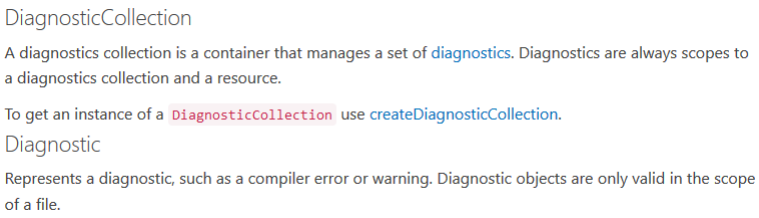


Obrázek 4.3: Debugger a okno s aktivním doplňkem

4.5 API Visual Studio Code

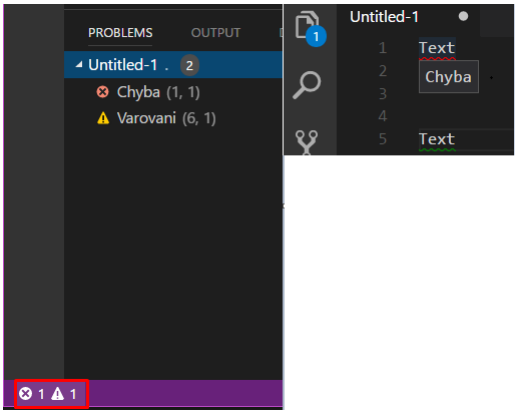
API (z anglického *Application Programming Interface*) je programové rozhraní, která nám říká jakým způsobem přistupovat a ovládat prvky daného programu. *VSCode* má rozsáhlé rozhraní, které je popsáno na jeho webových stránkách[11].

V době psaní tohoto dokumentu bylo toto rozhraní popsané velice špatně, s nedostatkem příkladů a nedostatečným popisem. Vůbec jsem neměl představu, pomocí čeho bych dosáhl interakce s prvky ve *VSCode* jaké bych potřeboval. Inspiraci jsem proto hledal v doplňcích jiných autorů, abych pochopil, co je vlastně co. Jelikož je *VSCode* stále ještě mladý projekt doufám, že se to do budoucna změní a lepší popis povede ke zkvalitnění a nárůstu počtu publikovaných doplňků. Jako příklad uvedu jeden z prvků a jeho nedostatečný popis, který budu dále ve svém doplňku využívat. *DiagnosticCollection*, což je kolekce *Diagnostic* (4.4). Jak můžete vidět *DiagnosticCollection* obsahuje pouze informaci, že je jakýsi kontejner, který spravuje kolekci *Diagnostic*. *Diagnostic* je poté popsána jako chyba nebo varování, například překladače. Není zde uvedena žádná zmínka o tom, jak se interakce s danými prvky ve *VSCode* projeví.



Obrázek 4.4: Popis API

V praxi to znamená, že *Diagnostic* je určitá část textu, která je podtržená vlnovkou zbarvenou zeleně nebo červeně podle toho, jestli se jedná o varování nebo chybu. Tato část textu poté může obsahovat dodatečné informace (4.5). *DiagnosticCollection* je poté kontejner, který je vyvolaný stisknutím myši do levého spodního rohu a obsahuje veškeré *Diagnostic* k upravovanému textovému dokumentu (4.5). Myslím si, že by neškodilo doplnit o praktické příklady zdrojového kódu a vizuální demonstraci, jako jsem to provedl já.

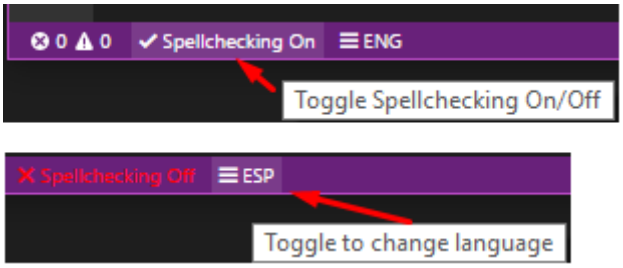


Obrázek 4.5: DiagnosticCollection a Diagnostic

Jak můžeme vidět podtržení textu červenou vlnovkou je skvělý způsob, jak splnit část 2. bodu našeho návrhu, a to jasně a výrazně označit nekorektní slovo.

4.6 Grafické rozhraní doplňku

Grafické rozhraní není nic složitého. Opět zde byla pouze překážka nedostatečně příkladné dokumentace, která se však táhne s celým projektem, a proto jí již nebudu zmiňovat.



Obrázek 4.6: Grafické rozhraní

Jak jsem si navrhl v 1. bodě návrhu, doplněk by měl pro jeho ovládání využívat dvě tlačítka. Jedno pro přepnutí stavu zapnout/vypnout a druhé pro změnu jazyka. Rozhodl jsem se tyto dvě tlačítka umístit na spodní lištu, do levého rohu (4.7), neboť je zde dost místa a uživatel nemusí nic hledat ve skrytých nastaveních.

Implementace poté vypadá následovně:

```
1  var statusBarItemSpell = vscode.window.createStatusBarItem(vscode.
       StatusBarAlignment.Left, 2);
2  var statusBarItemLanguage = vscode.window.createStatusBarItem(vscode.
       StatusBarAlignment.Left, 1);
3
4  statusBarItemSpell.command = "activateSpell";
5  statusBarItemSpell.tooltip = "Toggle Spellchecking On/Off";
6  statusBarItemSpell.show();
7
8  statusBarItemLanguage.command = "changeLanguage";
9  statusBarItemLanguage.tooltip = "Toggle to change language";
10 statusBarItemLanguage.show();
11 statusBarItemLanguage.text = "$(three-bars) " + settings.languages[0].id;
```

Pomocí metody *createStatusBarItem* jsou vytvořeny dvě tlačítka, která jsou zarovnána doleva s určitou prioritou, čímž zařídíme, aby se jedno tlačítko zobrazilo vždy na levé straně od tlačítka druhého. Dále doplníme text tlačítka, text, který se zobrazí při najetí myší, a aktivujeme zobrazení tlačítek pomocí metody show. *Command* určuje, jaká funkce bude zavolána v případě stisknutí tohoto tlačítka. Tímto by jsem uzavřel první bod návrhu za vyřešený.

4.7 Podmínky aktivace kontroly pravopisu

Samotný doplněk při spuštění *VSCode* kromě zobrazení ovládacích tlačítek a nahrání slovníku do paměti žádnou práci nevykonává. Vyčkává, dokud nenastane jedna z podmínek, která by zapříčinila aktivaci kontroly pravopisu. Toto paradigma nazýváme anglicky *event-driven programming*. Případem takové podmínky může být například kliknutí myší, zmáčknutí kterékoliv klávesy, nebo situace, která v programu nastane. Jedná se o nejpoužívanější způsob v programech s grafickým rozhraním. Pro aktivaci kontroly pravopisu jsem zvolil tyto dva *Eventy*.

Prvním *eventem* je *onDidOpenTextDocument*, ke kterému dochází kdykoliv uživatel otevře nový textový dokument nebo změní typ textového dokumentu. Za změnu typu se považuje například změna z dokumentu typu *JavaScript* na typ dokumentu *C++*.

Druhým *eventem* je *onDidChangeTextDocument*. Tento *event* nastane v případě, že dojde k jakékoliv změně v obsahu upravovaného textového dokumentu. Tady to začíná být trošku složitější. Funkce příslušná danému eventu je volána pokaždé, kdy k němu dojde. V případě prvního *eventu* to není žádný problém. Uživatel otevře textový dokument a tím provede kontrolu pravopisu. K aktivaci tohoto *eventu* dojde pouze jednou pro jeden editovaný soubor. U druhého *eventu* by k aktivaci docházelo pokaždé, kdy by uživatel stiskl jakýkoliv znak, který by upravil obsah editovaného dokumentu. Při obyčejném psaní by se tak stalo několikrát za vteřinu. Toto chování je nežádoucí, neboť by zbytečně zatěžovalo procesor. Proto je nutné zajistit, aby se kontrola prováděla pouze v případě, kdy nedošlo k mnohonásobné aktivaci *eventu* za určitý časový interval. Pokud ano, kontrolovat pouze pro poslední aktivaci tohoto *eventu*. Zabránit se tomu dá metodou podobnou, která se využívá v internetových našeptávačích.

```
1 function timedSpellCheck(){
2
3 clearTimeout(timeout);
4 timeout = setTimeout(spellCheck, 1000);
5 }
```

Při první aktivaci *eventu* dojde k pokusu o zrušení *timeoutu*. Tento *timeout* ještě nebyl vytvořen, proto se nic nestane. Poté dojde k vytvoření toho *timeoutu*, který nám říká, že dojde k volání funkce *spellCheck*, což je kontrola pravopisu, za dobu 1000 milisekund. Při druhém volání je tento *timeout* zrušen a opět nastaven. Pokračuje se tak dlouho, dokud uživatel nepřestane provádět kontinuální změny v intervalu 1 sekundy. Poté je funkce kontroly pravopisu zavolána a provedena.

4.8 Kontrola pravopisu

Samotná kontrola pravopisu je srdcem celého doplňku. V případě že je tlačítko doplňku ve stavu zapnuto, může proběhnout kontrola pravopisu. Nejprve je nutné získat text aktuálně editovaného dokumentu. Toho dosáhneme pomocí:

```
1 var origText = vscode.window.activeTextEditor.document.getText();
```

Poté musíme takto získaný obsah upravit, než budeme moci provést samotnou kontrolu. Úpravu textu provedeme pomocí metody *replace*. Metoda *replace* nalezne hledaný výraz pomocí regulárního výrazu, který poté nahradí za námi zvolený textový řetězec. Regulérní výraz je speciální textový řetězec, který slouží pro popsání vyhledávacího vzoru. Doplňující značka *g* zařídí, aby k těmto změnám došlo v celém rozsahu kontrolovaného řetězce.

```
//Sjednouti znacky konce radku z ruznych systemu
origText = origText.replace(/\r?\n/g, '\n');
// Nahradi tabulatory mezerou
origText = origText.replace(/\t/g, ' ');
//Nahradi posloupnost cislic mezerou
origText = origText.replace(/ [0-9]+/g, ' ');
//Nahradi interpunkci a jine neslovni znaky mezerou
origText = origText.replace(/[\"!#$%&()*+,.\/:;<=>?@\[\]\\^_{|}\n\r\-~]/g, '
    ');
```

Po aplikování prvního regulérního výrazu je nutné uchovat kopii tohoto textu rozdělenou po jednotlivých řádcích. Tímto získáme pole, jehož každou položkou je text jednoho řádku upravovaného textu.

Po aplikování posledního regulérního výrazu text rozdělíme po jednotlivých slovech, kdy každé slovo bude uloženo zvlášť opět do pole.

Získali jsme tedy dvě pole. Jedno obsahuje text rozdělený po řádcích, druhé rozdělený text po slovech. Tuto metodu aplikujeme z důvodu uchování informace, na kterém řádku se dané slovo nachází. Podívejme se na následující kód:

```
for (var i in words) {

  // If word is not in ignored words, converted to lowercase Ahoj,ahoj
      equals ahoj in ignoredWords
  if (ignoredWords.indexOf(words[i].toLocaleLowerCase()) == -1) {
            //If wrong
            if (!dict.correct(words[i]) && words[i].length > 0) {
                // Find position of wrong spelled word
                position = lines[lineNumber].indexOf(words[i], lastPosition);
                // If not found on first line
                while (position < 0) {
                   //Reset lastPosition and search next line
                   lastPosition = 0;
                   lineNumber++;
                   // When found
                   if (lineNumber < lines.length) {
                          position = lines[lineNumber].indexOf(words[i],
                              lastPosition);
                   }

                 }
                 // Next word in array must be after the previous word so we
                     can set new start position from the old one
                 colNumber = position;
                 lastPosition = position + words[i].length;
                 var lineRange = new vscode.Range(lineNumber, colNumber,
                     lineNumber, colNumber + words[i].length);
                 var diag = new vscode.Diagnostic(lineRange,"Suggested word/s
                     : " + dict.suggest(words[i]).toString(), vscode.
                     DiagnosticSeverity.Error);
                 diagnostics.push(diag);

            }
   }
}
```

Jednotlivá slova jsou nejprve porovnána s polem, které uchovává klíčové slova jednotlivých typů textových dokumentů. V případě, že slovo nepatří mezi klíčová slova, je porovnáno se slovníkem pomocí knihovny *Nspell*. Pokud je slovo označeno za chybné, je nutné zjistit, na které pozici v textu se slovo nachází. Tady přichází na řadu pole, ve kterém jsme si uchovali jednotlivé řádky. Slovo se snažíme na daném řádku nalézt. V případě neúspěchu pokračujeme na další řádky tak dlouho, dokud dané slovo nenalezneme. Poté si označíme číslo řádku, pozici na řádku a délku slova. Další slovo v pořadí vyhledává od bodu nalezení předchozího slova. Tímto zabráníme znovu označovaní stejných slov. Tyto údaje nám stačí k označení špatně zapsaného slova. Toho dosáhneme právě pomocí vytvoření *Diagnostic*.

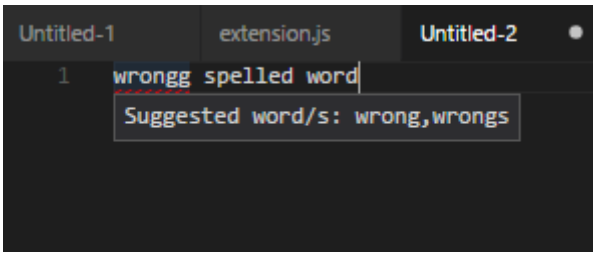
```
1 var lineRange = new vscode.Range(lineNumber, colNumber, lineNumber,
      colNumber + words[i].length);
2 var diag = new vscode.Diagnostic(lineRange,"Suggested word/s: " + dict.
      suggest(words[i]).toString(), vscode.DiagnosticSeverity.Error);
3 diagnostics.push(diag);
```

Nejprve si vytvoříme *VSCode* objekt typu rozsah. Jako parametry konstruktoru vložíme uložené údaje o pozici slova. Číslo řádku, na kterém slovo začíná. Číslo počátečního znaku. Číslo řádku, na kterém slovo končí (tento parametr se bude vždy shodovat s prvním), číslo koncového znaku slova.

Poté vytvoříme *VSCode* objekt typu *Diagnostic*. Jako parametry konstruktoru vložíme rozsah z předchozího bodu, informativní zprávu, která se zobrazí při najetí myší přes dané slovo. V našem případě se jedná o pole navrhovaných správných slov k nesprávnému tvaru. Posledním parametrem je typ barvy, kterou se má slovo podtrhnout.

1. *vscode.DiagnosticSeverity.Error* podtrhne slovo červeně.
2. *vscode.DiagnosticSeverity.Warning* podtrhne slovo zeleně.

Takto vytvořený objekt vložíme do pole typu *DiagnosticCollection*, aby byl správně zaregistrován programem *VSCode*. Ukázku můžeme vidět na obrázku 4.7. Tímto bychom vyřešili druhý bod našeho implementačního návrhu.



Obrázek 4.7: Kontrola pravopisu

4.9 CamelCase

Označuje způsob psaní víceslovných frází formou bez mezery. Takto napsaná fráze musí mít každé druhé a další slovo zapsané tak, že jeho první písmeno je velké. Jelikož se jedná

o standartní konvenci označování proměnných v některých programovacích jazycích, nemůžeme tento problém ignorovat. Na první pohled by se mohlo zdát, že pomocí jednoduchého regulárního výrazu bychom slova mohli rozdělit jako v předchozích případech.

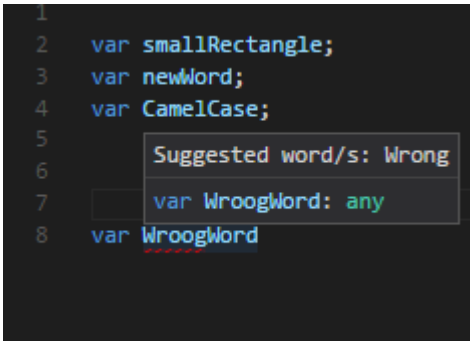
```
1 origText = origText.replace(/([a-z])([A-Z])/g, '$1 $2');
```

Tento výraz vloží mezeru mezi veškeré znaky, které odpovídají sekvenci, kdy velké písmeno následuje za písmenem malým. To je sice pravda, ale základní regulární výrazy jazyka JavaScript bohužel neumí pracovat s rozšířenou znakovou sadou. České pojmenování proměnné technikou *CamelCase* například *malýČtverec* by proto nebylo vůbec rozděleno. To by znamenalo, že toto slovo bude označené za chybné, i když je touto technikou napsané spisovně správně.

Ještěže je tady skvělý balíček *XRegExp*[16] napsaný *Stevnem Levithanem* a vydaný pod *MIT* licencí. *XRegExp* umí pracovat s rozšířenou znakovou sadou, proto je ideálním řešením pro rozdělení slov i jiných jazyků, než angličtiny. Regulární výraz poté vypadá následovně:

```
1 var camelCase = xregexp('(\\p{Ll})(\\p{Lu})');
2 origText = xregexp.replace(origText,camelCase, '$1 $2', 'all');
```

Tímto jsme vyřešili 3 bod našeho návrhu. Takto kontrolovaný kód napsaný formou *CamelCase* můžete vidět na obrázku 4.8.



Obrázek 4.8: CamelCase

4.10 Instalace dalších jazyků

Vzhledem k tomu, že doplněk umí pracovat s různými jazyky na základě přiloženého slovníku a souborů přípon, je potřeba si ukázat, jak toho bylo dosaženo a jakým způsobem lze nové jazyky do doplňku přidávat.

Adresář doplňku obsahuje adresář s název languages, která implicitně obsahuje anglický a španělský slovník a jemu přidružený slovník přípon. Do této složky je tedy nutné nový jazyk vložit. Slovníky a jejich soubory přípon jsou volně dostupné ke stažení například na stránkách balíku *OpenOffice*. Toto ale není všechno. Aby doplněk registroval, že byl nový jazyk přidán, musí být doplněn do souboru *settings.json*. Ten se nachází v adresáři settings. Vypadá následovně:

```
{
  "languages": [
    {
      "id":"ENG",
      "name":"English",
      "aff_file":"en_US.aff",
      "dic_file":"en_US.dic"
    },
    {
      "id":"ESP",
      "name":"Spanish",
      "aff_file":"Spanish.aff",
      "dic_file":"Spanish.dic"
    }
    ]
}
```

Pro dokončení instalace podpory nového jazyka stačí tento soubor rozšířit o nový objekt.

1. *id* Označuje identifikátor jazyka. Tento identifikátor je zobrazen na druhém tlačítku grafického rozhraní doplňku
2. *name* Název jazyka. Tato položka nemá žádný vliv na správný běh doplňku.
3. *aff_file* Označuje název souboru přípon, uloženém v adresáři languages.
4. *dic_file* Označuje název slovníku v adresáři languages.

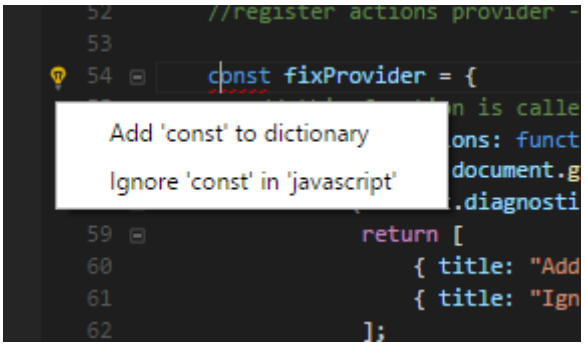
4.11 Možnost nekontrolovat klíčová slova

Jedním z požadavku na tento doplněk je právě funkce nekontrolovat klíčová slova podle typu upravovaného dokumentu. *VSCode* má pro každý podporovaný typ dokumentu jednoznačný identifikátor. Pokud uživatel otevře poprvé typ souboru, pro který neexistuje předem připravený soubor s klíčovými slovy, je tento soubor vytvořen v adresáři settings a pojmenován právě podle tohoto jednoznačného identifikátoru.

Uživatel může manuálně vepsat do těchto souborů klíčová slova, která nechce, aby doplněk bral v potaz. Slovo se zapisuje vždy jedno na jeden řádek. Toto manuální řešení je dost nepraktické, proto je zde možnost přidávat slova rovnou z grafického rozhraní *VSCode*. Manuální řešení se hodí hlavně v případě, že uživatel stáhne k danému typu dokumentů celou řadu klíčových slov. Tyto slova je pak možné jednoduše doplnit do souborů ve složce settings. Pro tento případ by naopak bylo dosti nepraktické přidávat slova po jednom přes rozhraní *VSCode*.

Přidávání slov přes *VSCode* je umožněno díky implementaci *ActionProvideru*. *ActionProvider* je úzce spjatý s *DiagnosticCollection*. *DiagnosticCollection* musí obsahovat *Diagnostic*, což v našem případě znamená nesprávně napsané slovo. Pokud tuto *Diagnostic* obsahuje, je po vytvoření *ActionProvideru* vždy volána metoda *provideCodeActions*, ale pouze v případě, že uživatel na nesprávné slovo klikne. Díky tomu se na levé straně na stejném řádku objeví malá žlutá žárovka 4.9. Po kliknutí na tuto žárovku má uživatel na

výběr toto nesprávně napsané slovo přidat buď do slovníku daného jazyka, nebo do slovníku daného typu dokumentu.



Obrázek 4.9: Žárovka s rožšířenou nabídkou

Poďme si ukázat implementaci tohoto *ActionProvideru*:

```
// If is not in array yet
if (fixProviderLanguageIds.indexOf(vscode.window.activeTextEditor.document.
    languageId) == -1){
    // Register code action provider
    fixer = vscode.languages.registerCodeActionsProvider(vscode.window.
        activeTextEditor.document.languageId, fixProvider);
    // add to array so its not used twice for same document
    fixProviderLanguageIds.push(vscode.window.activeTextEditor.document.
        languageId);
}
```

ActionProvider jde pro jeden typ dokumentu zaregistrovat vícekrát. Programátor si proto musí dát pozor, aby se této praktiky vyvaroval, neboť by všechny položky nabídky po rozkliknuti žárovky byly zobrazeny více než jednou.

Jednoduchou kontrolou ověříme, jestli se v poli nenachází již registrovaný *ActionProvider* pro stejný typ dokumentu. Pokud ne, *ActionProvider* pomocí *registerCodeActionsProvider* vytvoříme a zaznamenáme si informaci, že už pro daný typ dokumentu vytvořen byl.

Implementace metody *provideCodeActions*:

```
provideCodeActions: function (document, range, context, token) {
    let word = document.getText(range);
    if (context.diagnostics[0] != undefined){
       return [
       { title: "Add '" + word + "' to dictionary", command: "
           addToDictionary", arguments: [word] },
       { title: "Ignore '" + word + "' in '" + vscode.window.
           activeTextEditor.document.languageId + "'", command: "
           addToIgnored", arguments: [word] }
        ];
     }
}
```

Tato metoda má předepsanou formu, a proto nemůžeme ovlivnit argumenty, které poskytuje. Naštěstí disponuje vším, co potřebujeme. Nejprve je nutné zjistit, pro které slovo byla zavolána. Návratovou hodnotou této metody jsou jednotlivé položky, které se zobrazí při rozkliknuti žárovky.

1. *title* Označuje popisek položky v menu po rozkliknuti žárovky.
2. *command* Označuje název identifikátoru, pro který musí být zaregistrovaná funkce. Tato funkce je poté po kliknutí na tento popisek v menu zavolána.
3. *arguments* Označuje, jaké argumenty mají být této funkci předány.

Jak je zmíněno v druhém bodě, identifikátory musí být nejprve přiřazeny k funkci, aby funkce mohly být následně zavolány. Toho se dá docilit pomocí funkce *registerCommand*.

```
1 vscode.commands.registerCommand("addToDictionary", addToDictionary.bind(
      this));
2 vscode.commands.registerCommand("addToIgnored", addToIgnored.bind(this));
```

První argumentem této metody je indentifikátor, druhým poté funkce, která má být zavolána. Těmito prostředky jsme docílili splnění bodu 4 a 5 z našeho návrhu.

4.12 Publikace

Pokud je doplněk plně funkční a řádně otestovaný je načase ho publikovat. Samotná publikace za použití podpůrného programu je velice jednoduchá. Před ní si však musíme vytvořit uživatelský účet na stránkách *VSCode*. Tím zabezpečíme svůj doplněk proti případné krádeži identity. Po zaregistrování účtu si v uživatelském profilu musíme vygenerovat token pro personální přístup 4.10. Token si uložte, neboť nebudete mít už možnost stejný token zobrazit. Pořád tady ale zůstává možnost vygenerovat token nový. Poté je nutné nainstalovat opět pomocí *NPM* z příkazové řádky vsce, což je program pro publikování doplňků. Vsce nainstalujeme následovně:

```
npm install -g vsce
```

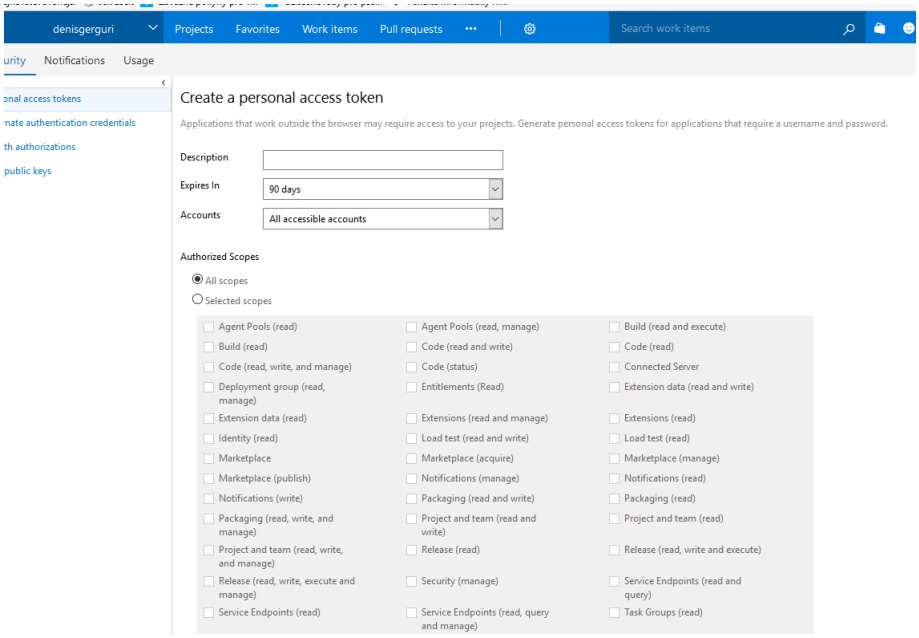
Dále je nutné se zaregistrovat jako vydavatel.

```
vsce create-publisher (vase_jmeno)
```

Budete vyzvání ke vložení dodatečných údajů jako e-mailová adresa a hlavně námi vytvořený bezpečnostní token. Poté již stačí ve složce s doplňkem spustit příkaz:

```
vsce publish 1.0.0
```

První verze s číslem 1.0.0 je na světě. Dokumentace doplňku byla vytvořena pomocí krátkých animací. Odkaz na tyto animace je umístěn v *README.md* souboru. Animace se krásně zobrazují, jak na hlavní straně doplňku v internetovém prohlížeči na stránce *VSCode* marketu, tak i v rozhraní *VSCode*. Doplněk lze nalézt pod názevem *hunspell-spellchecker*.



Obrázek 4.10: Generování personálního tokenu

Kapitola 5

Navrhovaná vylepšení

Každý program je neustále ve vývoji a vždy se najde prostor pro jeho zlepšení. Tato kapitola by se měla věnovat myšlenkám, které by mohly přispět ke zkvalitnění dosavadního řešení.

5.1 Vylepšená možnost nekontrolovat klíčové slova

Mnou implementované řešení sice splňuje požadavky na toto kritérium, dalo by se ovšem vylepšit. Současné řešení spoléhá na uživateli, který si slovník klíčových slov vytvoří buďto sám, anebo stáhne jejich seznam. Mnohem kvalitnější řešení by spočívalo v propojení se zvýrazňováním syntaxe na základě typu dokumentů. Zvýrazňovaní syntaxe je ve *VSCode* popisováno pomocí souboru zvaných *tmLanguage*. *tmLanguage* proto, protože pochází z textového editoru *TextMate* [6], který je dostupný pouze na operační systém *macOS*. Jedná se o sadu jazykových pravidel používaných k přiřazení jmen prvkům dokumentu, jako jsou například právě klíčová slova, komentáře, textové řetězce apod. Obrázek 5.2 ukazuje takto vytvořenou jednoduchou jazykovou gramatiku.

```
{  scopeName = 'source.untitled';
   fileTypes = ( );
   foldingStartMarker = '\{\s*$';
   foldingStopMarker = '^\s*\}';
   patterns = (
      {  name = 'keyword.control.untitled';
         match = '\b(if|while|for|return)\b';
      },
      {  name = 'string.quoted.double.untitled';
         begin = '"';
         end = '"';
         patterns = (
            {  name = 'constant.character.escape.untitled';
               match = '\\.';
            }
         );
      },
   );
}
```

Obrázek 5.1: Gramatika v jayzce tmLanguage

Celý proces spočívá ve využití regulárních výrazů, pomocí kterých jsou určitě části kódu označeny do skupin, mezi kterými můžeme dále popisovat vztahy a přiřazovat jim finální podobu, tedy barvu.

Ač se to zdá být na první pohled složité, nic z toho bychom vlastně ani nepotřebovali. Místo klíčových slov uložených ve složce settings, by stačilo ukládat pouze barvu v hexadecimálním nebo jiném tvaru, kterou bychom chtěli ignorovat. Poté by stačilo před samotnou kontrolou pravopisu pouze zjistit, jestli dané slovo není zbarveno touto ignorovanou barvou. *VSCode* bohužel zatím nepodporuje možnost danou barvu z textu zjistit.

5.2 Časová složitost

Dosavadní řešení při jakékoliv i sebemenší změně upravuje a porovnává opět všechny slova. Bylo by dobré navrhnout algoritmus, který by nekontroloval oblasti, ve kterých nedošlo ke změně a jejich umístění v textu se od poslední kontroly nezměnilo.

Tato optimalizace by se projevila hlavně u dlouhých textových dokumentů. U řádku následovaných za upravovaným textem by nedocházelo ke změně, pouze pokud by se změny týkaly jednoho jediného řádku. V opačném případě by nedocházelo ke změně v řádcích před upravovaným textem, a to i v případě že by uživatel udělal několika řádkovou úpravu. Algoritmus by nejspíš spočíval v ukládání *metadat*, který by vedli o daných úpravách informace.

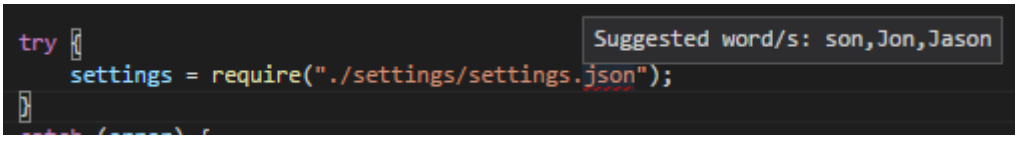
Další z možností by mohl být asynchronní přístup, kdy by byl výsledek spojen dohromady.

5.3 Složitější regulární výrazy

Mějme například název souboru *package.json*. Dosavadní implementace nahradí v daném výrazu tečku mezerou. Vzniknou tedy dvě slova.*Package*, který nebude v rámci anglického slovníku označen za chybný, a *json*, který za chybný označen bude4.3. Bylo by vhodné v rámci příštích verzí implementovat způsob, kdy by tyto koncovky souborů byly ignorovány. Nejspíše pomocí regulárních výrazů.

5.4 Automatické stáhnutí slovníku

Instalaci nového slovníku musí uživatel momentálně provést manuálně. V *README.md* souboru má poskytnutých několik odkazů, kde tyto slovníky stáhnout. Uživatel poté musí otevřít adresář s doplňkem, slovníky zde přesunout a také upravit soubor *settings*, aby doplněk o novém jazyku věděl. Ideální by bylo řešení, kdy by uživateli stačilo v grafickém rozhraní VSCode napsat, o jaký jazyk má zájem a ten by byl následně automaticky stažen a nainstalován. Takto nainstalovaný jazyk by stejným způsobem mělo být možné i odinstalovat.



Obrázek 5.2: Přípona označena za chybé slovo (nežádoucí chování)

Kapitola 6

Závěr

Cílem této práce bylo vytvořit doplněk pro kontrolu pravopisu pro nový textový editor zdrojového kódu *Visual Studio Code*. Za tímto účelem bylo nevyhnutelné pochopit, jak taková kontrola pravopisu probíhá a jaké techniky jsou při ní uplatněny. Doplněk byl úspěšně vytvořen a také publikován, čímž považuju cíl práce za splněný.

Práce srovnává a popisuje vývoj kontroly pravopisu od prvních *Linuxových* programů minulého století, až po dnešní nejpoužívanější programy a knihovny. Dále se věnuje tvorbě, rozboru a samotné implementace jednoduché kontroly pravopisu od začátku až po publikaci hotového doplňku pro výše zmíněný textový editor. Doplněk disponuje rozšířenou funkcí nekontrolovat víceslovné fráze známé jako *CamelCase* a možností nekontrolovat klíčové slova různých programovacích jazyků.

Dále popisuje, s jakými problémy se musel autor vypořádat a navrhuje další možností vylepšení hotového doplňku, zejména pokročilou možnost nekontrolovat klíčové slova úzce spjatou s implicitní podporou zvýrazňování syntaxe *VSCode*.

Literatura

[1] Atkinson, K.: GNU Aspell. [Online; navštíveno 12.02.2017].
URL http://aspell.net/

[2] autorů, K.: BSD licenses. [Online; navštíveno 12.02.2017].
URL https://en.wikipedia.org/wiki/BSD_licenses

[3] autorů, K.: BSD licenses. [Online; navštíveno 12.02.2017].
URL https://en.wikipedia.org/wiki/BSD_licenses

[4] autorů, K.: Damerau–Levenshtein distances. [Online; navštíveno 12.02.2017].
URL https://en.wikipedia.org/wiki/Damerau-Levenshtein_distance

[5] autorů, K.: Electron. [Online; navštíveno 12.02.2017].
URL https://github.com/electron/electron

[6] autorů, K.: Language Grammarsí. [Online; navštíveno 12.02.2017].
URL https://manual.macromates.com/en/language_grammars

[7] autorů, K.: Licence MIT. [Online; navštíveno 12.02.2017].
URL https://cs.wikipedia.org/wiki/Licence_MIT

[8] autorů, K.: Metaphone. [Online; navštíveno 12.02.2017].
URL https://en.wikipedia.org/wiki/Metaphone

[9] autorů, K.: Microsoft Visual Studio Code. [Online; navštíveno 12.02.2017].
URL https://code.visualstudio.com/

[10] autorů, K.: n-graml. [Online; navštíveno 12.02.2017].
URL https://en.wikipedia.org/wiki/N-gram

[11] autorů, K.: VS Code API. [Online; navštíveno 12.02.2017].
URL https://code.visualstudio.com/docs/extensionAPI/vscode-api

[12] autorů, K.: Yeoman. [Online; navštíveno 12.02.2017].
URL http://yeoman.io/

[13] Hendricks, K.: OOo-MySpell. [Online; navštíveno 12.02.2017].
URL https://code.google.com/archive/a/apache-extras.org/p/ooo-myspell

[14] Kuenning, G.: International Ispell. [Online; navštíveno 12.02.2017].
URL https://www.cs.hmc.edu/~geoff/ispell.html

[15] Lachowicz, D.: Enchant. [Online; navštíveno 12.02.2017].
URL https://github.com/AbiWord/enchant

[16] Levithan, S.: XRegExp. [Online; navštíveno 12.02.2017].
URL https://github.com/slevithan/xregexp

[17] Németh, L.: Hunspell. [Online; navštíveno 12.02.2017].
URL http://hunspell.github.io/

[18] doc. RNDr. Aleš Horák, P.: Syntaktická a sémantická analýza a reprezentace znalostí. [Online; navštíveno 12.02.2017].
URL https://www.fi.muni.cz/research/nlp/analysis.xhtml.cs

[19] Wormer, T.: Nspell. [Online; navštíveno 12.02.2017].
URL https://github.com/wooorm/nspell

Přílohy

Příloha A

Obsah CD

Přiložené datové médium obsahuje:

- Zdrojové kódy doplňku
- Technickou správu ve formátě pdf
- Zdrojové soubory technické správy pro LaTeX