FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Podpora syntaxe LaTeXu v rámci služby Grammarly
Student:	Tomáš Hojek
Vedoucí:	Ing. Ondřej Guth, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Proveďte analýzu a návrh softwarového řešení, které usnadní používání služby Grammarly pro dokumenty psané v LaTeXu. Navržený software umožní provést plnohodnotnou kontrolu pravopisu textu latexovského dokumentu pomocí Grammarly (služba v době zadání nepodporuje syntaxi LaTeX, pouze prostý text) a následnou aplikaci změn navržených službou do původního dokumentu; vše za účelem eliminace nebo alespoň minimalizace či usnadnění manuální činnosti (odstranění značek LaTeX, zanesení změn do původního dokumentu) pro autora dokumentu. Vhodnou podmnožinu navrženého řešení implementujte jako prototyp. Své výsledky vhodným způsobem otestujte.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 17. ledna 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Podpora syntaxe LaTeXu v rámci služby Grammarly

Tomáš Hojek

Katedra softwarového inženýrství
Vedoucí práce: Ing. Ondřej Guth, Ph.D.

26. června 2019

Poděkování

Tímto bych rád poděkoval svému vedoucímu práce, Ing. Ondřeji Guthovi, Ph.D., za ochotu, poskytnuté rady, vždy pozitivní přístup a obětovaný čas, který mi věnoval během tvorby prototypu aplikace a psaní této bakalářské práce. Děkuji také mé rodině a přátelům za podporu během celého mého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 26. června 2019 ……

Odkaz na tuto práci

Hojek, Tomáš. *Podpora syntaxe LaTeXu v rámci služby Grammarly.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Tato práce se zaměřuje na tvorbu aplikace umožňující jazykovou kontrolu textu napsaného v jazyku LaTeX. Jedná se o aplikaci, která sama neprovádí kontrolu textu, ale umožňuje ji pomocí externích služeb. V této práci se využívá služba Grammarly, která umožňuje kontrolu anglických textů. Součástí práce je rešerše služby Grammarly, rešerše již jednoho existujícího řešení, které ale spolupracuje s jinou externí službou, a analýza požadavků. Na základě požadavků byl navrhnut a implementován prototyp výše zmíněné aplikace. Prototyp byl implementován v programovacím jazyce Java a JavaFX jako desktopová multiplatformní aplikace.

Klíčová slova LaTeX, služba Grammarly, detekce gramatických chyb, jazyková kontrola, prostý text, kontrolování anglických textů

Abstract

This thesis deals with the creation of an application that allows checking texts written in LaTeX. This application does not check the text itself but allows to use it with external services. In this work the Grammary service is used, which allows you to check English texts. In this thesis is also a review of the Grammary service, review of one existing solution, which communicates with another external service, and an analysis of requirements. Based on requirements, a prototype of the above-mentioned application was designed and implemented. The prototype was implemented in the Java programming language and JavaFX as a desktop multiplatform application.

Keywords LaTeX, Grammarly service, detect grammar mistakes, language check, plain text, checking English texts

Obsah

Seznam obrázků

Seznam tabulek

Úvod

LaTeX je sada maker pro sázení technických a odborných textů ve velmi vysoké typografické kvalitě pomocí programu TeX. Dokument napsaný v LaTeX je obyčejný textový dokument, který obsahuje kromě prostého textu i celou řadu příkazů pro překladač. Tyto příkazy definují podobu výsledného dokumentu i vlastního textu. Text je však promíchán s těmito příkazy a výslednou podobu dokumentu je možné získat až po jeho kompilaci překladačem.

Pro vytváření LaTeX dokumentů existuje mnoho specializovaných editorů (například TeXstudio nebo Texmaker), které většinou automaticky kontrolují pravopisné chyby a překlepy. Pokud však uživatel chce zkontrolovat tento text například i na gramatické chyby, je většina editorů bezradná a je nutné využít některou z externích služeb pro jazykovou kontrolu textů. Mezi takové služby patří například: Grammarly [1] nebo LanguageTool [2]. Jelikož tyto služby umí pracovat pouze s prostými texty, je nutné buď z textu ručně odstranit všechny LaTeX výrazy, nebo dokument přeložit a výsledný text zkopírovat a vložit do editoru služby (tato možnost však bývá problematická). Nejvíc práce je ovšem při procesu samotné kontroly takového textu, neboť provedené opravy je nutné následně ručně aplikovat přímo do zdrojového textu.

Tato práce se zaměřuje na zjednodušení a především minimalizaci manuální činnosti uživatele při jazykové kontrole dokumentů psaných v LaTeX. Pro tuto kontrolu bude využívána služba Grammarly, na které bude též probíhat testování a výchozím jazykem pro tyto kontroly bude angličtina.

Struktura práce

V úvodu této práce byl velmi krátce představen LaTeX a základní problém týkající se jazykové kontroly v dokumentech obsahujících LaTeX výrazy. V následující kapitole bude popsáno zadání bakalářské práce a její cíle. V kapitole 2 bude popsána služba Grammarly, její možnosti a omezení, a projekty podobné tématu této bakalářské práce. V další kapitole bude popsána provedená

analýza. Kapitola 4 je stěžejní částí této práce, neboť zde bude popsán návrh aplikace a implementace prototypu. V poslední části (kapitola 5) bude popsán způsob testování vytvořeného prototypu aplikace.

Kapitola 1

Cíle práce

1.1 Popis zadání

Cílem této práce je navrhnout vhodné softwarové řešení, které usnadní používání služby Grammarly pro dokumenty psané v LaTeX. Grammarly je služba, která umožňuje kontrolovat anglické texty po stylistické i gramatické stránce. Služba však umí pracovat pouze s prostým textem a LaTeX výrazy jsou proto v kontrolovaném textu velmi problematické. Z toho důvodu je potřeba provést rešerši služby Grammarly a na základě této rešerše navrhnout takové řešení, které umožní provést plnohodnotnou kontrolu anglického textu a následně provedené úpravy aplikovat do původního textu. Na základě tohoto návrhu bude vytvořen funkční prototyp, na kterém bude provedeno závěrečné testování.

1.2 Shrnutí cílů

Cíle bakalářské práce lze na základě zadání shrnout následovně:

1. provést rešerši služby Grammarly (pro použití jako nástroje pro jazykovou kontrolu dokumentů psaných v LaTeX),
2. na základě provedené rešerše navrhnout softwarové řešení,
3. implementovat funkční prototyp,
4. provést závěrečné testování vytvořeného prototypu.

Kapitola 2

Služba Grammarly a podobné projekty

Tato kapitola obsahuje popis a rešerši služby Grammarly, která bude výchozí online službou pro jazykovou kontrolu anglických textů, a dále jsou zde popsány podobné projekty nebo projekty s tématikou alespoň částečně podobnou této práci.

2.1 Služba Grammarly

Grammarly je online služba, která provede analýzu textu a označí problematická místa v kontrolovaném textu. Následně pro každé označené místo navrhne možnosti, jak danou chybu opravit. Služba v době psaní této bakalářské práce podporuje pouze texty psané v anglickém jazyce. Grammarly však umí rozlišovat 4 dialekty, ze kterých si uživatel může vybrat. Jedná se o britskou, americkou, kanadskou a australskou (či novozélandskou) angličtinu. V textu jsou kontrolovány stylistické i gramatické chyby a případně navrhovány i záměny určitých slov za vhodnější výrazy. Dále je kontrolováno i správné použití interpunkce[1]. [3, 4, překlad autor]

V rámci rešerše byla tato služba kontaktována za účelem získání informací, zda je poskytováno API (application programming intergace, volným překladem rozhraní pro programování aplikací). Případně pokud by toto API existovalo, ale nebylo veřejně dostupné, zda by služba byla ochotna autorovi bakalářské práce API zpřístupnit. Bohužel odpověď byla záporná (API neexistuje) [5]. Přepis e-mailové komunikace je v příloze C.

Jelikož služba nemá API, bylo rozhodnuto (po konzultaci s vedoucím práce), že pro kontrolu textu bude preferován Grammarly Editor, který je dostupný online na stránkách Grammarly bez potřeby jakékoliv instalace. Pro použití tohoto editoru musí mít uživatel založený účet u Grammarly,

[1] například: Mr Doe vs. Mr. Doe, což je závislé právě na zvoleném dialektu.

ale tato podmínka je uplatňována i v případě ostatních možností, které služba Grammarly nabízí (například rozšíření do prohlížeče, desktopová aplikace). V editoru se zobrazuje pouze prostý text bez jakéhokoliv formátování. Pokud tedy uživatel text zkopíruje a vloží do editoru, bude provedena konverze na prostý text, který bude následně v editoru analyzován. Pokud uživatel editoru chce formátování zachovat a má text uložený v dokumentu, může využít možnosti nahrání dokumentu do editoru a po kontrole opravený dokument stáhnout. Grammarly editor podporuje dokumenty ve formátech .odt, .doc, .docx, .rtf, .txt[2]. [6, překlad autor]

2.1.1 Omezení služby Grammarly

Tato služba má podle dokumentace řadu omezení, z nichž některá platí pouze pro Grammarly Editor. První z těchto omezení je, že služba může být jedním uživatelem využívána maximálně na 5 zařízeních. Dále existují limity na množství zkontrolovaného textu za den a také za měsíc (30denní perioda). V rámci 24 hodin je možné zkontrolovat maximálně 100 dokumentů nebo 50 000 slov a v rámci 30 dní je možné zkontrolovat maximálně 300 dokumentů nebo 150 000 slov. [7, překlad autor]

Další omezení již platí pouze pro Grammarly Editor. Omezení jsou zde celkem dvě. První z nich je maximální počet znaků, 100 000 znaků zahrnující i mezery a odřádkování, pro kontrolu vloženého textu nebo 1 dokumentu. Druhé omezení je na maximální velikost nahrávaného souboru, což je 100 kB. [7, překlad autor]

Poslední dvě omezení, která se vztahují na používání Grammarly Editoru, byla otestována. Při nahrání příliš dlouhého textu byla zobrazena informační hláška, že dokument musí mít méně než 60 A4 v závorce uvedeno 100 000 znaků. Neboli první omezení se skutečně uplatňuje. Druhý limit však Grammarly Editor povolil přesáhnout a podle informační hlášky omezení platí až pro soubory větší než 4 MB.

2.2 Podobné projekty

V době psaní této bakalářské práce neexistují žádná softwarová řešení pro kontrolu angličtiny pomocí služby Grammarly v dokumentech psaných v jazyku LaTeX. Při opomenutí podmínky na kontrolu pomocí služby Grammarly a rozšíření na možnost kontroly pomocí libovolné externí služby, není situace o mnoho lepší. Obecně totiž existuje velice málo programů snažících se zprostředkovat kontrolu angličtiny v textech obsahující LaTeX výrazy.

Mnoho editorů pro LaTeX uvádí, že podporují i kontrolu jazyka (angličtiny), ale tato kontrola zahrnuje pouze překlepy a text po gramatické stránce

[2] V průběhu rešerše služby však několikrát nastal problém s formátováním textu ve staženém dokumentu.

zkontrolován není. Existuje i několik programů, které z těchto dokumentů vygenerují text vhodný pro kontrolu v jiných službách, ale není možné tyto změny aplikovat zpátky do původního textu. V této kapitole bude představeno jedno nalezené řešení, které umožňuje kontrolu textu.

2.2.1 TeXtidote

Prvním podobným projektem je nekomerční program nazvaný TeXtidote [8]. Předností programu je skutečnost, že je zveřejněn a šířen pod licencí GNU. Autorem tohoto programu je profesor Sylvain Hallé z katedry Matematiky a informatiky na Université du Québec à Chicoutimi v Kanadě. Text v této kapitole vychází z popisu programu uveřejněným jejím autorem na veřejném úložišti GitHub v souboru `Readme.md`. [9, překlad autor]

Program je napsán v programovací jazyku Java – verze 1.8 – a je tedy možné ho spustit pod jakýmkoliv běžným operačním systémem – Windows, Linux, Mac OS. Program ovšem funguje pouze přes příkazovou řádku a absence grafického rozhraní může některé uživatele odradit.

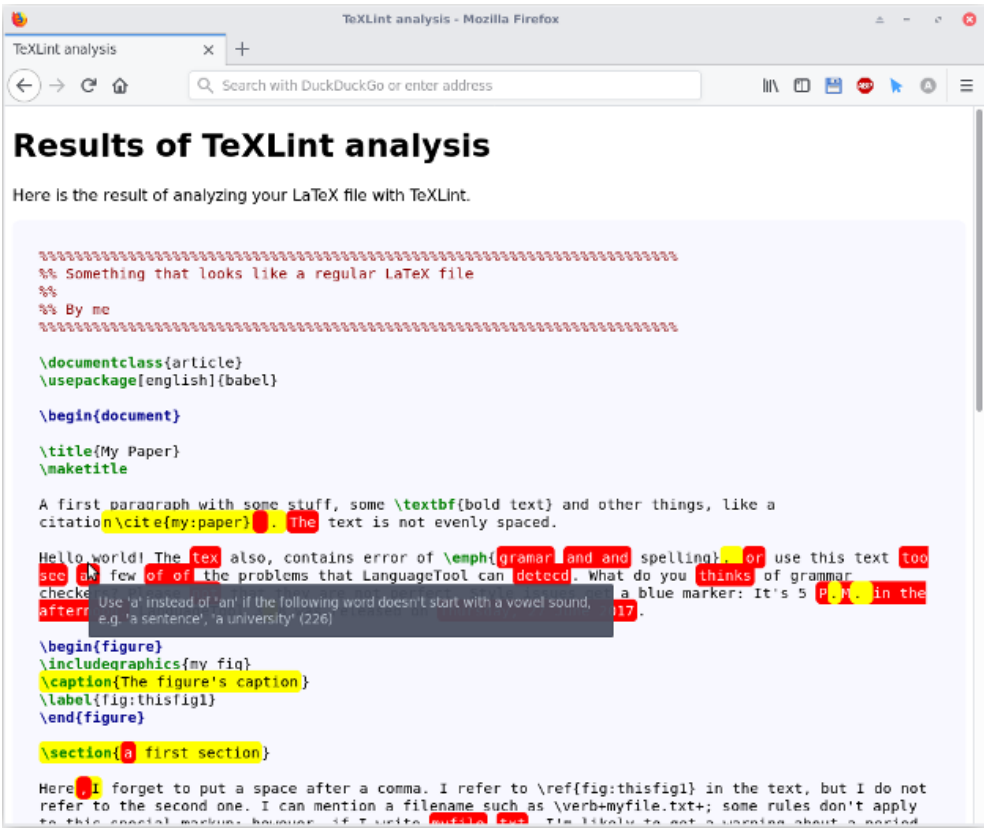
Ke kontrole textu je využívána služba LanguageTool [2]. Využití této služby je jedna z hlavních předností tohoto nástroje, neboť služba umožňuje kontrolu textů v mnoha různých jazycích. Program se službou komunikuje pomocí API rozhraní, které služba poskytuje. To přináší výhodu v podobě jednoduchosti, neboť uživatel nemusí vědět, jak se kontrola provede.

Služba LanguageTool ani TeXtidote text přímo neopravují, ale pouze umožňují najít potencionální chyby, které musí uživatel opravit již sám. Po provedení kontroly jsou uživateli zobrazeny potencionální chyby a doporučení, jak je opravit. Podle vstupních parametrů při spuštění tohoto nástroje se výsledky kontroly mohou zobrazit dvěma různými způsoby.

```
* L25C1-L25C25 A section title should start with a capital
↪  letter. [sh:001]
  \section{a first section}
  ^^^^^^^^^^^^^^^^^^^^^^^^
* L38C1-L38C29 A section title should not end with a punctuation
↪  symbol.
  [sh:002]
  \subsection{ My subsection. }
  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^
* L15C94-L15C99 Add a space before citation or reference.
↪  [sh:c:001]
   things, like a citation\cite{my:paper} .The text
```

Výpis kódu 2.1: Výstup programu TeXtidote v konzoli. Získáno z oficiální dokumentace [9]

Obrázek 2.1: Příklad výstupu programu TeXtidote v okně webového prohlížeče. Získáno z [10]



První způsob, který je výchozí volbou, spočívá v tom, že se do konzole vypíší části textu s komentáři, co je v dané části textu špatně, viz 2.1. Druhý způsob, pokud při spuštění programu přidáme argument `--html`, zobrazí LaTeX dokument ze vstupu v okně internetového prohlížeče a barevně označí místa v textu, kde jsou potencionální chyby. Pokud uživatel přemístí kurzor myši na některou barevně vyznačenou část textu, zobrazí se mu informace, co je špatně a proč. Ukázka výstupu programu z oficiální dokumentace k prohlédnutí zde 2.1. Samotné opravení chyb v LaTeX dokumentu musí uživatel udělat sám.

Software umožňuje kromě kontroly samotného jazyka i zobrazení textu s odstraněnými výrazy jazyka LaTeX. To může být užitečné v případě kontroly jinou službou.

Na závěr kapitoly je potřeba říci, že tento program je aktuálně nejlepším nástrojem pro kontrolu angličtiny, a nejen té, v LaTeX dokumentech. Nástroj provede kontrolu bez interakce s uživatelem a zobrazí následná doporučení, jak a kde opravit potencionální chyby. Výsledná oprava chyb ve zdrojovém textu je však na samotném uživateli.

Kapitola 3

Analýza

3.1 Analýza uživatelských požadavků

Jelikož služba Grammarly nemá API, výsledné softwarové řešení nebude přímo komunikovat s touto službou a nebude tedy přímo zprostředkovávat kontrolu textu. Bude nutné, aby tuto kontrolu provedl uživatel pomocí služby Grammarly sám.

Definování požadavků je důležitý bod analýzy. Má-li být výsledné softwarové řešení použitelné a fungovat dle očekávání, je nutné definovat požadované funkcionality a vlastnosti aplikace. Proto budou nyní uvedeny funkční a nefunkční požadavky, které byly sestaveny na základě konzultací s vedoucím práce.

3.1.1 Funkční požadavky

Funkční požadavky se používají k definování chování aplikace [11]. Uvedeny jsou postupně podle významnosti pro výslednou aplikaci.

F1: Generování textu bez LaTeX výrazů Aplikace bude umožňovat generování prostého textu vhodného pro kontrolu gramatiky pomocí externích služeb (především službou Grammarly).

F2: Aplikování provedených změn Uživatel bude moci nahrát opravený text do aplikace a na základě tohoto textu a zdrojového textu bude provedena aplikace provedených změn do původního textu obsahujícího LaTeX výrazy.

F3: Konfigurace příkazů a prostředí V aplikaci bude možnost upravovat seznam příkazů a prostředí, které se v textu mohou vyskytovat. K nim bude moct uživatel uvést doplňující údaje, tj. počet povinných parametrů, zda se mají ve výsledném textu zobrazit i parametry příkazu obsažené ve složených či hranatých závorkách.

F4: Konfigurace varování Aplikace bude umožňovat zvolit strategii pro přidávání výstražných komentářů v místech, kde došlo ke změně. Tato konfigurace je určena zejména pro jednodušší kontrolu uživatelem po aplikaci provedených změn zpět do původního textu.

F5: Konfigurace obecně Z důvodu uživatelské přívětivosti bude aplikace obsahovat předdefinované základní konfigurace a v případě změny těchto konfigurací budou tyto změněné konfigurace uloženy a použity i při následujících spuštění aplikace.

F6: Textové dokumenty Služba Grammarly umožňuje pracovat jak s prostým textem, tak s textovými dokumenty. Proto by aplikace mohla kromě práce s textem podporovat i práci s některým z podporovaných textových dokumentů.

3.1.2 Nefunkční požadavky

Nefunkční požadavky popisují další důležité vlastnosti aplikace. Jsou to například požadavky na bezpečnost systému, nároky na výkonost, a tak podobně. [12]

N1: Desktopová multiplatformní aplikace Jelikož aplikace nemůže se službou Grammarly komunikovat prostřednictvím API, bude desktopová a zcela lokální, neboť kontrolu gramatiky textu bude zajišťovat uživatel sám (za pomoci externích služeb/aplikací). Dále aplikace bude multiplatformní, aby ji mohlo používat co nejvíce uživatelů.

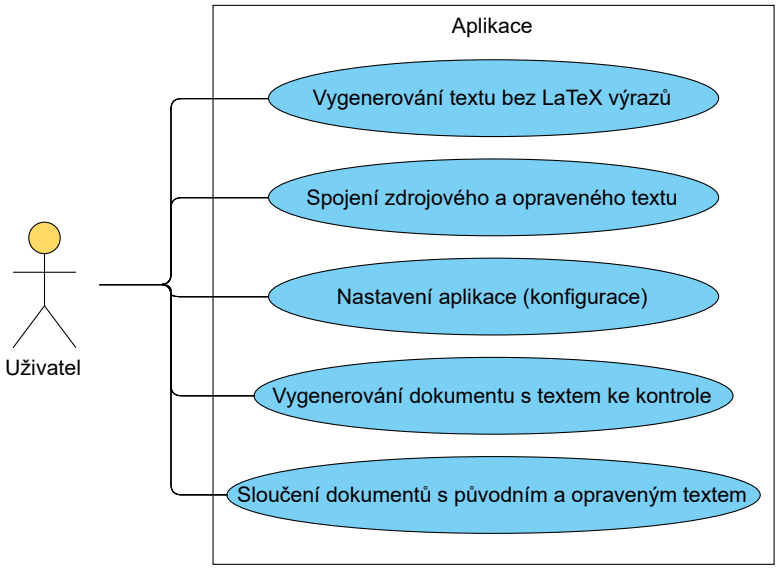
N2: Grafické uživatelské rozhraní Aplikace bude nabízet grafické uživatelské rozhraní, které umožní ovládání klávesnicí a myší.

3.2 Případy užití

Případy užití (nebo též use cases) se používají pro sběr či mapování požadavků na systém. Každý případ užití definuje množinu akcí, které jsou systémem vykonávané a které vedou ke splnění již zmíněných požadavků na systém. Výsledek vzniklý na základě konkrétního případu užití je typicky důležitý pro účastníka případu užití. Účastník je typicky role mimo systém, který případ užití vykonává. V tomto případě bude účastníkem vždy uživatel aplikace. [13]

Na obrázku 3.1 je vidět diagram pro dříve definované funkce aplikace. Aktér je v tomto případě pouze jeden, uživatel aplikace. Pro všechny případy užití bude nyní uvedena jejich krátká specifikace.

Obrázek 3.1: Diagram případů užití



U1: Vygenerování textu bez LaTeX výrazů
Aktér: Uživatel
Popis: Jedná se o úplně základní funkcionalitu aplikace. Uživatel vloží text obsahující LaTeX výrazy do aplikace a ta z tohoto textu vygeneruje ekvivalentní text bez LaTeX výrazů.

U2: Spojení zdrojového a opraveného textu
Aktér: Uživatel
Popis: Jedná se o druhou základní funkcionalitu aplikace. Uživatel vloží zdrojový text obsahující LaTeX výrazy a opravený text, který byl dříve aplikací vygenerován. Na základě těchto dvou textů aplikace vytvoří text sloučený z těchto dvou textů, tzn. původní text s LaTeX výrazy a aplikovanými opravami.

U3: Nastavení aplikace (konfigurace)
Aktér: Uživatel
Popis: V aplikaci se bude nacházet více druhů konfigurací, díky kterým bude uživatel moci upravovat chování aplikace. Jedná se především o konfigurace LaTeX příkazů, LaTeX prostředí. Aplikace bude mít výchozí konfigurace a případné změny bude ukládat.

U4: Vygenerování dokumentu s textem ke kontrole
Aktér: Uživatel
Popis: Uživatel do aplikace nahraje LaTeX dokument a aplikace vygeneruje textový dokument obsahující text určený pro jazykovou kontrolu. Uživatel musí mít možnost zvolit místo uložení textového dokumentu a jeho pojmenování. Pokud se uživatel pokusí nahrát jiný než podporovaný dokument, aplikace na to uživatele musí upozornit.

U5: Sloučení dokumentů s původním a opraveným textem
Aktér: Uživatel
Popis: Uživatel do aplikace nahraje zdrojový LaTeX dokument a opravený textový dokument a vygeneruje LaTeX dokument s aplikovanými opravami. Opět je nutné umožnit uživateli zvolit místo uložení a pojmenování nového dokumentu.

3.2.1 Mapování případů užití na požadavky

V následující tabulce 3.1 je uvedeno mapování případů užití na všechny funkční požadavky.

Tabulka 3.1: Mapování případů užití na funkční požadavky

	U1	U2	U3	U4	U5
F1	X			X	
F2		X			X
F3			X		
F4			X		
F5			X		
F6				X	X

Kapitola 4

Návrh a implementace prototypu

Jak je uvedeno v zadání, výsledkem této práce bude prototyp aplikace. V této kapitole bude popsán návrh a implementace prototypu. Na začátek je však důležité popsat zvolené technologie pro vyvíjený prototyp.

4.1 Zvolené technologie

Systém bude implementován jako desktopová multiplatformní aplikace v jazyce Java. Byla zvolena Java 11, neboť se jedná o nejnovější verzi Javy, která byla vydána s označením LTS, neboli verze s dlouhou dobou podpory. Konkrétně pro Javu 11 to znamená podporu až do roku 2026. [14, překlad autor]

Pro vytvoření GUI byl zvolen framework JavaFX, který se používá pro tvorbu bohatých okenních aplikací. Výhodou tohoto frameworku je jeho multiplatformnost, jednoduchost, efektivnost a moderní vzhled. Další frameworky pro tvorbu GUI jsou Swing a AWT, ale obě tyto možnosti jsou dnes již zastaralé a pro nově vznikající aplikace se nedoporučují. [15, překlad autor]

Při implementaci prototypu byly použity dvě externí pomocné knihovny – JAXB [16] a diff-match-patch [17]. JAXB je knihovna pro práci s dokumenty ve formátu XML. Knihovna diff-match-patch umožňuje porovnat dva textové řetězce a najít rozdíly. Popis použití těchto knihoven bude uveden později v následujících kapitolách.

4.2 Návrh a implementace funkcionalit aplikace

V této kapitole budou popsány návrhy pro generování prostého textu, tedy textu bez LaTeX výrazů, vhodného pro jazykovou kontrolu pomocí externí služby Grammarly a pro sloučení původního a opraveného textu. Nejprve bude uvedeno několik obecných návrhů, které byly vymyšleny, ale z určitých důvodů

se neosvědčily. Nakonec bude představen i finální návrh, který bude popsán detailněji, a bude popsána i jeho samotná implementace v prototypu aplikace.

Nejprve je důležité určit, jak bude propojena funkcionalita generování prostého textu s možnou kontrolou pomocí služby Grammarly. Jak již bylo zmíněno dříve v kapitole o službě Grammarly 2.1, služba neposkytuje v době psaní této práce žádné API rozhraní. Komunikaci, přenos připraveného textu mezi aplikací a službou, musí tedy zajistit sám uživatel. A stejně tak je zřejmé, že i opravený text bude muset uživatel zpětně přenést do aplikace.

4.2.1 Komunikace pomocí dokumentu formátu ODT

První návrhy komunikace se službou jsou postaveny na následující modelové situaci. Uživatel do aplikace vloží LaTeX dokument a spustí generování prostého textu. Program vytvoří nový dokument formátu ODT, který bude uložen a pojmenován dle přání uživatele a bude obsahovat prostý text vhodný pro kontrolu pomocí služby Grammarly. Uživatel tento dokument nahraje do Grammarly Editoru, provede kontrolu a opravu textu a stáhne si opravený dokument ve formátu ODT. Tento opravený dokument následně vloží do aplikace spolu s výchozím LaTeX dokumentem a spustí akci pro sloučení těchto dokumentů.

Navrácení LaTeX výrazů do opraveného dokumentu však není jednoduché. Je potřeba nalézt místa, kam se mají tyto výrazy vložit a k dispozici máme pouze původní dokument a opravený dokument, ve kterém se mohl změnit text i na místech, kde původně začínal či končil některý z odstraněných LaTeX výrazů. Jelikož je pro opravu textu vytvářen nový dokument, přichází v úvahu přidat do tohoto dokumentu něco, co situaci zjednodušší (například nějaký symbol či značku). Tato úprava však nesmí ovlivnit text či možnost jeho kontroly.

Formát ODT umožňuje definovat skrytý text [18], což vypadá jako perfektní řešení, neboť LaTeX výrazy nemusí být vůbec odstraněny – budou pouze skryty. Toto řešení je však funkční pouze pokud editor podporuje tuto funkčnost, což bohužel Grammarly Editor nedělá – zobrazuje se i skrytý text. Do textu není možné umisťovat ani netisknutelné znaky, neboť Grammarly Editor tyto znaky automaticky odstraňuje.

Ve formátu ODT (ale i v ostatních textových formátech) je možné definovat styly formátování. Každý styl má své označení, které se v editorech nezobrazuje, určuje barvu písma, font, velikost fontu atp. Každé slovo by tedy mohlo mít definovaný vlastní styl, styly by se mezi sebou lišily pouze označením/jménem stylu a ostatní specifikace by byly stále stejné. Díky tomu by se toto řešení nijak neprojevilo v editorech dokumentů ODT. Editor služby Grammarly nezobrazuje formátování textu, a tak se neprojeví rozhodně ani v něm. Vrácení LaTeX výrazů na svá původní místa by znamenalo sledovat text a jeho označení styly před a po kontrole. Po opravení textu mohou nastat dále popsané případy.

- Jedno slovo bylo nahrazeno za jiné, ale původní styl zůstal zachován. V takovém případě změna nebude vůbec zaregistrována, což nevadí a okolní LaTeX výrazy budou jednoduše vráceny na svá původní místa.
- Ke slovu byl připojen další text, který získal styl formátování předcházejícího slova. Tato situace je celkem jednoduše identifikovatelná (více slov definovaných stejným stylem a původní slovo zůstalo zachováno). V tomto případě lze jednoduše umístit zpět LaTeX výrazy, které byly původně před slovem. Hůře se ale určuje, kam umístit LaTeX výrazy, které byly původně za slovem – mají být opět za ním nebo se mají vložit až za nově vložený text.
- Jedno slovo bylo nahrazeno za jiné slovo/slova, ale nebyl zachován původní styl formátování a nyní je použit neznámý styl (základní styl formátování používaný Grammarly Editorem). Opět na základě stylu formátování okolního textu je možné určit, které LaTeX výrazy zde původně byly a umístit je před či za tuto změnu. U tohoto případu je možnost potencionální chyby, pokud bude několik slov nahrazeno za několik jiných slov a uvnitř původního uskupení slov byl LaTeX výraz. Avšak tento případ je problematický vždy a nelze mu nijak předejít.
- Došlo ke smazání slova. Opět podle stylu formátování okolního textu je možné určit, jestli zde byl LaTeX výraz a případně ho vrátit na jeho původní místo.
- Vložení textu, který je formátován neznámým stylem. Tato situace je relativně snadno identifikovatelná, ale je složitější rozhodnout, zda případný LaTeX výraz by měl být umístěn před nově vložený text, nebo až za něj. Opět se však jedná o případ, který je problematický vždy, nezávisle na rozhodnutí používat styly formátování.

Bohužel i v tomto případě návrh neuspěl, neboť Grammarly Editor ignoruje označení stylů formátování a při generování opraveného dokumentu zachovává pouze obsah definovaného stylu formátování, tzn. font, velikost fontu atp. V důsledku tedy editor služby zjistí, že se jedná o text, který má všude stejné formátování a výsledný opravený dokument má pouze dva definované styly formátování. Prvním stylem formátování je původní styl formátování textu a druhým je styl formátování, který použil editor služby Grammarly pro vložení nových textových řetězců (oprav).

Postup a myšlenka stylů formátování, které byly popsány v předchozím odstavci, však vedly téměř k vytyčenému cíli, a proto další návrh byl založený na této myšlence s drobnými úpravami. Styly formátování by se mohly lišit i v základních aspektech, tj. velikost fontu, kurzíva, tučné písmo atp. Tyto aspekty by bylo možno mezi sebou kombinovat, čímž by bylo možné vytvořit dostatečně velké spektrum různých stylů. Tento návrh byl jako první implementován i v prototypu aplikace a dlouho rozvíjen a přizpůsobován chování

editoru služby Grammarly. Výsledkem byl prototyp, který relativně fungoval. Hlavním problémem bylo, že při příliš velkých změnách, jako je například vložení delší věty či přepsání velké části odstavce, Grammarly Editor způsoboval nepředvídatelné chyby ve formátování textu. Problémy s tímto návrhem se projevily 15. 3. 2019, kdy služba Grammarly udělala několik menších změn v chování editoru k formátování textu. V důsledku vytvářený prototyp přestal fungovat. Tento návrh a myšlenka využití dokumentů ODT formátu byly proto zamítnuty.

4.2.2 Finální návrh a implementace prototypu

Na základě dříve popsaných návrhů a problémů, které měly, byl vytvořen již finální návrh. Obecný postup použití by mohl být popsán následovně. Uživatel vloží LaTeX text a vygeneruje z něj prostý text, který vloží do editoru služby Grammarly. Provede doporučované opravy a následně vezme opravený text a vloží jej do aplikace. Ta na základě porovnání prostého a opraveného textu nalezne rozdíly. Na základě nalezených rozdílů provede sloučení zdrojového LaTeX textu a opraveného textu. Text může být do služby Grammarly předán jak jednoduchým zkopírováním, tak pomocí dokumentu ve formátu ODT. Tento postup má výhodu především v tom, že je naprosto nezávislý na chování editoru služby a teoreticky by aplikace mohla být použita i v kombinaci s jinou externí službou pro jazykovou kontrolu textu.

4.2.2.1 Generování prostého textu

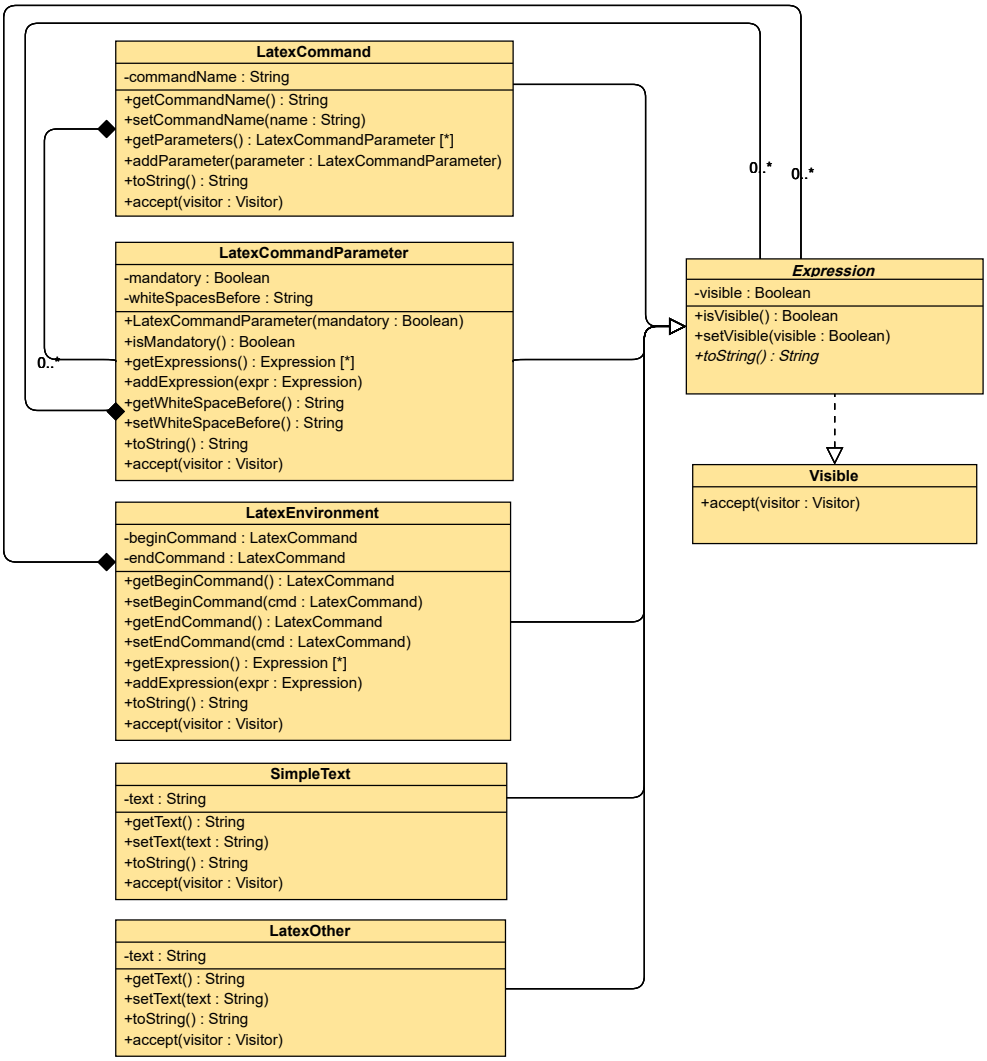
V této kapitole bude podrobněji popsán navrhovaný způsob generování prostého textu. Nejdříve je nutné přijatý LaTeX text analyzovat dle pravidel pro programovací jazyk LaTeX a vytvořit odpovídající datovou strukturu, která bude ekvivalentně reprezentovat přijatý text. Diagram tříd na obrázku 4.1 ukazuje navrhovanou strukturu systému tříd a vztahy mezi nimi. Následuje charakteristika a význam jednotlivých tříd zachycených v diagramu.

Expression je obecný představitel jakéhokoliv výrazu či uskupení znaků vyskytujících se v textu. `Expression` obsahuje proměnnou `visible` definující, zda má být daný výraz zobrazen v generovaném prostém textu.

LatexCommand je představitelem libovolného LaTeX příkazu nebo bloku, který se v textu může vyskytovat. Jeho proměnnými jsou `commandName`, reprezentující jméno příkazu, a seznam parametrů. Příklad LaTeX příkazu: `\maketitle`.

LatexCommandParameter reprezentuje parametr konkrétního LaTeX příkazu. Tento parametr může být povinný nebo volitelný, rozpoznatelný dle typu použitých závorek ohraničujících parametr. Pro povinné parametry se používají složené závorky, zatímco pro volitelné parametry se

Obrázek 4.1: Struktura tříd reprezentující přijatý LaTeX text



používají závorky hranaté. Tuto skutečnost zde reprezentuje proměnná `mandatory`. Dále se před parametrem mohou vyskytovat bílé znaky, tzn. například mezera či odřádkování, které samotný text nijak neovlivňují a nezobrazují se. Tyto bílé znaky však autor textu může vložit například z důvodu lepší čitelnosti a aplikace by je měla nejdříve pro generovaný text pro jazykovou kontrolu odstranit a následně při slučování textů opět navrátit. Bílé znaky reprezentuje proměnná `whiteSpaceBefore`. Parametr ještě obsahuje seznam dalších `Expression`, které jsou obsaženy uvnitř parametru. Příklad LaTeX příkazu s povinným i volitelným parametrem: `\includegraphics [width=3cm] {logo}`.

LatexEnvironment je představitelem LaTeX prostředí. Prostředí se vyznačuje především tím, že má počáteční příkaz, `beginCommand`, a ukončující příkaz, `endCommand`. Mezi těmito dvěma příkazy se může vyskytovat libovolné množství dalších LaTeX výrazů. Příklad LaTeX prostředí: `\begin{document} Dokument \end{document}`.

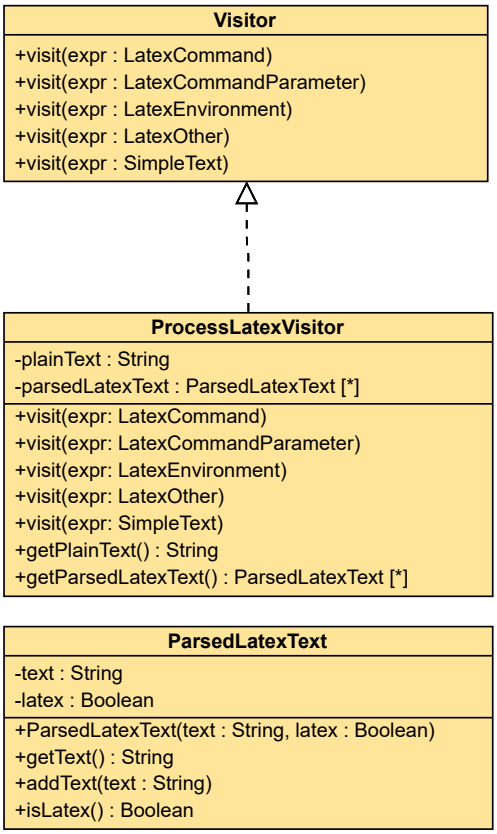
LatexOther reprezentuje zbývající LaTeX výrazy, které se nedají zařadit do žádné z výše uvedených kategorií. Jedná se například o zápis matematických výrazů, komentáře, zápis znaků se speciálním významem v jazyku LaTeX atp. Tento výraz je reprezentován proměnnou `text` a hodnota proměnné `visible` je v tomto případě vždy `false`.

SimpleText reprezentuje prostý text, který by měl být zobrazen uživateli na výstupu pro kontrolu pomocí služby Grammarly.

Díky převodu přijatého textu do strukturované podoby je jednodušší určit, co by mělo být obsaženo ve výsledném generovaném textu a co ne. Pro operaci vygenerování prostého textu je zde vhodné použít návrhový vzor zvaný Visitor. Tento vzor obecně obsahuje dva typy tříd – třídy, které jsou navštěvované, nesoucí v sobě základní funkcionalitu, a třídy navštěvující, které přinášejí dynamicky všem navštěvovaným třídám dodatečné chování [19]. Tento vzor je zde vhodný především proto, že dříve popsaná struktura může být zpracovávána různými způsoby a vygenerování prostého textu bez LaTeX příkazů je zajisté jedním z nich. Pokud by tedy struktura měla být později zpracovávána i jiným způsobem, stačí doplnit novou navštěvující třídu a není třeba nijak zasahovat do dřívějšího návrhu a kódu popsané struktury. Na obrázku 4.1 si lze všimnout, že třída `Expression` implementuje `Visible`, který požaduje implementovat jedinou metodu a to `accept(visitor : Visitor)`. Díky tomu každý návštěvník, který bude dědit od třídy `Visitor` bude moct navštěvovat popsané třídy a definovat pro každou z nich dodatečné chování.

Návštěvník (visitor) navrhovaný pro zpracování struktury na prostý text je znázorněn v diagramu tříd na obrázku 4.2. Na tomto diagramu je zachycen i obecný `Visitor`, který musí být implementován každým návštěvníkem. `ProcessLatexVisitor` je návštěvník, který ze struktury LaTeX výrazů vytvoří dva výstupy. Jedním z nich je prostý text určený pro kontrolu pomocí externí služby Grammarly. Druhým výstupem je seznam obsahující prostý text a LaTeX text, a ten je určen pro pozdější slučování s opraveným textem. Druhý zmíněný výstup je velmi důležitý, neboť popsaná struktura tříd reprezentující původní LaTeX text může být velmi složitá. Oproti tomu je tento výstup vcelku prostý – neobsahuje nadbytečné informace identifikující, co konkrétní textový řetězec znamená, ale pouze informaci, zda se jedná o prostý text, či něco z jazyku LaTeX. Uživateli tedy bude na výstupu pro generování prostého textu zobrazen text obsažený v proměnné `plainText` uvedeného návštěvníka.

Obrázek 4.2: Visitor pro generování prostého textu



Implementace převodu textu do popsané struktury tříd

Nyní bude popsána část implementace generování prostého textu, konkrétně převod LaTeX textu do struktury tříd popsané na diagramu tříd 4.1. Jedná se o nejsložitější část implementace funkcionality generování prostého textu.

První nápad zahrnoval použití vcelku komplikovaných regulárních výrazů. Regulární výraz 4.1, zapsaný v jazyku java, je ukázkou, jak komplikované regulární výrazy byly. V tomto konkrétním případě se jedná o regulární výraz, který hledal matematické zápisy ve zdrojovém textu.

```
String REGEX_MATH="(?<=[^\\\\])((\\${1,2}[^\\$]+[^\\\\]\\${1,2})
                   |(\\\\\\[[^\\[\\]]+[^\\\\]\\\\\\])
                   |(\\\\\\([^\\(\\)]+[^\\\\]\\\\\\)));"
```

Výpis kódu 4.1: Ukázka regulárního výrazu

Regulární výrazy v jazyce Java však mají svá omezení a jedním z nich je rekurze, kterou je občas velmi složité, až nemožné pomocí regulárního výrazu zapsat. Příkladem by mohl být následující jednoduchý výraz:

`{\title{\LaTeX{}}} text`. U tohoto výrazu je problém se zmíněnou rekurzí, neboť je potřeba vytvořit 3 instance třídy `LatexCommand`. Tyto instace LaTeX příkazů musí být do sebe vnořeny, přičemž pro správné zpracování zmíněného výrazu musíme počítat otevírací závorky, a poté uzavírací závorky, aby bylo možné správně určit, kde začíná a končí každý z příkazů. Nakonec tedy bylo od používání regulárních výrazů pro hledání LaTeX výrazů upuštěno.

Finální implementace tedy prochází přijatý zdrojový text písmenko po písmenku a vytváří požadovanou strukturu tříd. Na základě aktuálního písmene lze definovat následující základní kategorie.

Komentář Aktuální písmeno je % a všechno za tímto znakem až do odřádkování je komentář.

Matematický výraz Pro matematický výraz existuje již více možností, neboť může být uzavřen do jedné z následujících kombinací znaků: `$ $`, `$$ $$`, `\( \)`, `\[ \]`. Pokud se tedy aktuální znak a případně následující znak shodují s jednou ze zmíněných variant, tak se jedná o začátek matematického výrazu.

LaTeX prostředí Aktuální písmeno je \ a za ním následuje klíčové slovo `begin` pro začátek prostředí nebo slovo `end` pro konec prostředí. Dále pak prvním povinným parametrem LaTeX prostředí je název prostředí, který je součástí příkazu na začátku i na konci.

Znaky se speciálním významem Jedná se o znaky, které mají v jazyku LaTeX speciální význam, například $ či {, a pro jejich zápis v textu je nutné před ně umístit symbol \.

Speciální znaky a akcenty Jedná se například o diakritiku, kterou lze v jazyku LaTeX zapsat i pomocí speciální skupiny znaků. Zápis speciálních znaků a akcentů začíná symbolem \, za kterým následuje jedna z předdefinovaných kombinací znaků. V rámci implementace prototypu jsou speciální znaky a akcenty považovány za LaTeX výrazy a z výsledného prostého textu jsou odstraněny. Eventualitou by byla jejich náhrada za symbol, který představují, ale služba Grammarly by si s těmito znaky nemusela poradit (nebylo testováno). Pokud by byl navíc speciální znak odstraněn, bylo by složité rozhodnout o navrácení původní kombinace znaků představující speciální symbol či akcent.

LaTeX příkaz Aktuální písmeno je \, což ale k identifikaci, že se jedná o LaTeX příkaz nestačí, jak vyplývá z předešlých kategorií. Definice LaTeX v příkazu by tedy v tomto případě mohla vypadat následovně. Označení či jméno LaTeX příkazu začíná symbolem \ a je následováno malými či velkými písmeny abecedy a posledním symbolem za písmeny

abecedy může být jeden ze znaků *, @. Následovat může téměř libovolné množství bílých znaků, za kterými můžou následovat parametry příkazu.

Speciální symbol vlnovky Vlnovka, nebo též tilda (~), je v jazyku LaTeX symbol pro nezlomitelnou mezeru. V rámci implementovaného prototypu není podporována možnost náhrady symbolu či textového řetězce za jiný. Aby bylo možné text zkontrolovat, je v tomto případě implementována výjimka a tilda je nahrazena za mezeru. Důsledkem však je, že tilda se na dané místo v textu již nikdy nevrátí (po jazykové kontrole dokumentu s pomocí prototypu jsou odstraněny všechny výskyty tildy).

4.2.2.2 Slučování zdrojového a opraveného textu

Sloučení původního a opraveného prostého textu bude probíhat na základě porovnání prostého textu vygenerovaného z původního zdrojového textu a opraveného textu. Na základě tohoto porovnání jsou získány rozdíly v těchto dvou textech, které je třeba při slučování textů vyřešit. V částech textu, kde nedošlo k žádné změně je situace velice jednoduchá a stačí vrátit odstraněné LaTeX výrazy na svá původní místa. Naopak u nalezených rozdílů mohou nastat následující situace:

1. opravený text obsahuje slovo/slova navíc v místě, kde původně nezačínal ani nekončil žádný LaTeX výraz,
2. opravený text obsahuje slovo/slova navíc v místě, kde původně začínal nebo končil LaTeX výraz,
3. v opraveném textu bylo smazáno jedno či více slov

V prvním případě je situace velice jednoduchá, neboť stačí do výsledného textu vložit tento nový text. V druhém případě je však situace poněkud komplikovanější, neboť nelze obecně rozhodnout, zda má být nový text vložen před či za původní LaTeX výraz. Proto je vhodné zvolit taktiku, která bývá obvykle správná a upozornit uživatele na toto problematické místo. Obecně platí, jestliže byl vložen nový textový řetězec do původního textu a mezi původním a novým textem není žádná mezera, tak se jedná o upravení již existujícího slova a nový text by měl být vložen před LaTeX výraz. V případě varianty, že mezi původním a novým textem mezera je, je nutné zvolit jednu ze dvou možností a tu poté všude dodržovat, aby se jednalo o očekávatelné chování. Navíc je v tomto případě vždy vhodné uživatele na tento případ upozornit.

Poslední uvedená situace týkající se smazané části textu je řešitelná následovně. Do výsledného textu nebude tento text zcela přirozeně vložen a případné LaTeX výrazy, které byly původně kolem (nebo uvnitř) smazané části textu, by měly být vloženy do výsledného textu ve stejném pořadí jako byly v původním textu. Zde je však nutné opět upozornit uživatele na možnou

chybu – pokud byly LaTeX výrazy uvnitř smazané části textu, tak v novém textu již nemusí dávat smysl.

Prototyp aplikace při slučování změn v textech nejprve získá instanci dříve popsaného visitoru `ProcessLatexVisitor`, který obsahuje vygenerovaný prostý text a seznam obsahující prostý text i LaTeX výrazy. Následně je porovnán vygenerovaný a opravený text pomocí externí knihovny diff-match-patch. Tuto knihovnu je možné použít v mnoha programovacích jazycích, nejen v jazyku Java, a její popis je možné nalézt na [17]. Tato knihovna porovná zadané dva texty a vrátí spojový seznam reprezentující provedené porovnání textů. Každý objekt v tomto seznamu obsahuje část textu z jednoho z porovnávaných textů a enumeration hodnotu definující typ změny:

- úryvek textu je v obou porovnávaných textech stejný (EQUALS),
- úryvek byl z opraveného textu smazán (DELETE),
- úryvek byl do opraveného textu přidán (INSERT).

Na základě získaného porovnání textů, typů změn a dříve popsaných situací, které je nutné při slučování textů řešit, je sestaven výsledný text s aplikovanými opravami. V případě vloženého textu v místě, kde původně končil nebo začínal LaTeX výraz, prototyp aplikace vkládá nový text za původní pozici LaTeX výrazu. Dále pak prototyp vkládá upozornění pro uživatele, které vypadá následovně: `%\n% GRAMMARLY TODO!`.

Implementovaný prototyp kontroluje ještě jeden aspekt, který prozatím nebyl zmíněn. Zdrojový LaTeX text může obsahovat různé kódování pro odřádkování, a proto prototyp před generováním prostého textu zjistí, jaké kódování je použito a změní ho. Prototyp díky tomu všude pracuje s LF[3] odřádkováním. Po sloučení změn je do výsledného textu vráceno původní kódování odřádkování.

4.2.2.3 Konfigurace

Konfigurace, nebo též nastavení, je důležitou součástí vytvářeného návrhu a prototypu aplikace. Jak již bylo zmíněno v diagramu tříd 4.1, obecný výraz, tedy `Expression`, obsahuje proměnnou `visible`. Tato proměnná určuje, zda se má výraz v generovaném textu pro kontrolu pomocí služby Grammarly zobrazit, či ne. Konfigurace proto musí umožňovat definovat, které LaTeX příkazy a prostředí mají být v tomto textu zobrazeny. Dále pak konfigurace můžou obsahovat i další možná nastavení, která uživateli usnadní používání aplikace. V případě vytvořeného prototypu byla přidána možnost nastavit vhodnou strategii přidávání varujících komentářů do textu sloučeného ze zdrojového a opraveného textu.

[3] zkratka z anglického označení Line Feed, zapisuje se jako '\n'

Konfigurace příkazů musí obsahovat jméno příkazu, počet povinných parametrů příkazu a zda má být do generovaného prostého textu zobrazen i obsah povinných či nepovinných parametrů. Počet povinných parametrů příkazu by měla být nepovinná položka. Tato možnost je zahrnuta v konfiguracích z toho důvodu, že při strojovém procházení textu není možné vždy přesně určit, kde končí parametry příkazu a kde začíná blok. Například u LaTeX textu `\section{Příklad} {blok}` není možné strojově zjistit, že text `{blok}` již není parametrem příkazu `section`. Uvedený příklad by byl problematický především v případě, že obsah příkazu `section` by se neměl zobrazovat, ale obsah bloků by se obecně zobrazovat měl. Pro tyto případy je proto vhodné mít možnost zadat i počet povinných parametrů příkazu, čímž se problém z uvedeného příkladu vyřeší.

Konfigurace prostředí je do značné míry podobná konfiguraci příkazů. Ideálně by měla obsahovat jméno prostředí, zda se má obsah prostředí v generovaném prostém textu zobrazit, a počet povinných parametrů prostředí.

Z důvodu uživatelské přívětivosti by se konfigurace měli ukládat a při opětovném spuštění aplikace automaticky načíst. Prototyp aplikace pro ukládání konfigurací používá Java Preferences API [20], které umožňuje ukládat uživatelská nastavení na základě dvojice klíč a hodnota. Tato data se ukládají pouze pro aktuálního uživatele a kořenovou cestou je cesta java package, ve kterém jsou uložené konfigurace, tedy `cz.cvut.fit.hojektom.configuration`. Seznam jmen LaTeX příkazů, které byly do konfigurací přidány, se ukládá pod klíč `latexCommands` a podobně seznam jmen LaTeX prostředí se ukládá pod klíč `latexEnvironments`. V hodnotě je poté uložen seznam LaTeX příkazů a prostředí oddělených symboly `";;"`. Další hodnoty příkazů a prostředí je možné nalézt složením klíče z přecházejícího klíče, názvu příkazu či prostředí a identifikátoru. Tyto hodnoty jsou v klíči odděleny tečkou. Identifikátorem je myšleno klíčové slovo odpovídající hledané informaci/proměnné:

- zobrazovat obsah složených závorek (`curlyBracket`),
- zobrazovat obsah hranatých závorek (`squareBracket`),
- počet povinných parametrů (`numberOfParameters`),
- zda se má zobrazit obsah prostředí (`display`).

Posledním typem konfigurace je volba strategie generování varujících komentářů. V rámci prototypu aplikace bylo připraveno 5 základních možností, ze kterých si uživatel může zvolit tu, která nejlépe odpovídá jeho představám:

- komentáře se nebudou vkládat nikdy,
- komentáře se vloží vždy, a to před místo možné chyby,
- komentáře se vloží vždy ,a to za místo možné chyby,

- komentáře se vloží vždy, a to před i za místo možné chyby,
- komentáře se vloží dle potřeby, tedy pokud je vysoká šance, že mohlo dojít k chybě.

4.3 Grafické rozhraní

V této kapitole bude popsán návrh a implementace grafického rozhraní aplikace. Jelikož pro většinu dnešních aplikací je angličtina hlavním jazykem, kterým s uživatelem komunikují, tak i v tomto případě bude aplikace komunikovat v angličtině. V této části textu se proto budou často vyskytovat anglické termíny a věty, ke kterým bude uveden i autorův volný překlad (v případě delšího překladu formou poznámky pod čarou, jinak v závorce za uvedeným výrazem).

4.3.1 Návrh grafického rozhraní

Důkladné promyšlení návrhu grafického rozhraní je podstatnou částí návrhu, a proto mu je věnována samostatná kapitola, stejně jako následné implementaci tohoto návrhu. Nová aplikace musí být na první pohled jednoduchá a intuitivní, aby se uživatel rychle zorientoval a bylo pro něj používání aplikace příjemné a automatické.

Jak je znázorněno na obrázku 4.3, aplikace bude obsahovat v horní části menu se čtyřmi záložkami: Plain text editor (Editor prostého textu), Generate ODT document (Generování ODT dokumentu), Generate LaTeX document (Generování LaTeX dokumentu), Configuration (Konfigurace). Návrhy obsahů těchto záložek budou popsány v následujících podkapitolách.

4.3.1.1 Editor prostého textu

Tato záložka bude výchozí a zobrazí se hned po spuštění aplikace. Její rozložení je znázorněno na obrázku 4.4. Její význam a použití je velice jednoduché. Uživatel zkopíruje text, který obsahuje LaTeX výrazy a který chce zkontrolovat, a vloží ho do prvního textového pole s instrukcí „Here insert your LaTeX text:“[4]. Následně uživatel stiskne tlačítko Generate (generovat) a uživateli se zobrazí připravený text pro jazykovou kontrolu pomocí externí služby Grammarly. Vygenerovaný text se zobrazí v novém okně, jehož návrh je znázorněn na obrázku 4.5. Uživatel text zkopíruje a vloží do editoru služby Grammarly a provede opravu textu. Tím je z pohledu grafického rozhraní realizován první případ užití, tedy U1 (Vygenerování textu bez LaTeX výrazů).

Opravený text uživatel vloží do druhého textového pole s instrukcí „Here insert checked plain text:“[5]. Poté, co uživatel stiskne tlačítko Merge (spojit),

[4] v překladu „Zde vložte váš LaTeX text:“
[5] v překladu „Zde vložte opravený text:“

je opět v novém okně 4.5 zobrazen výsledný text, který odpovídá původnímu LaTeX textu s provedenými opravami. Tato druhá část navrhované záložky odpovídá případu užití U2 (Spojení zdrojového a opraveného textu).

4.3.1.2 Generování ODT dokumentu

Rozložení této obrazovky je znázorněno na obrázku 4.6. Tato obrazovka je určena pro vložení LaTeX dokumentu a následné vygenerování dokumentu ve formátu ODT obsahujícím text bez LaTeX výrazů, tedy vhodného pro jazykovou kontrolu pomocí externí služby Grammarly. Pro jednoduché a intuitivní používání bude aplikace umožňovat nahrát LaTeX dokument pouhým přetažením dokumentu do vyznačené oblasti, nebo zvolení cesty k souboru na disku. Tato obrazovka je návrhem pro grafické zpodobnění případu užití U4 (Vygenerování dokumentu s textem ke kontrole).

4.3.1.3 Generování LaTeX dokumentu

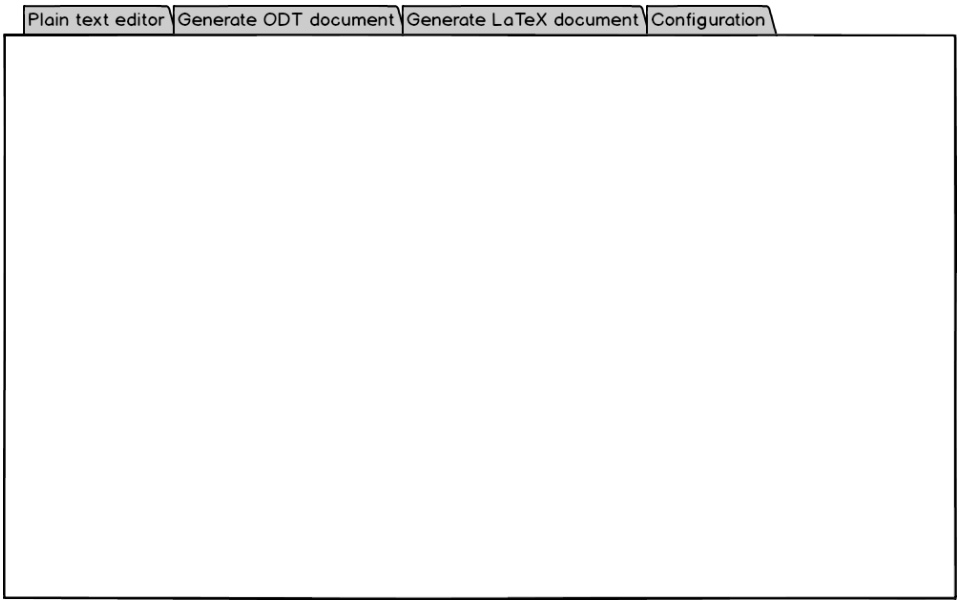
Návrh rozložení této obrazovky je na obrázku 4.7 a odpovídá případu užití U5 (Sloučení dokumentů s původním a opraveným textem) . Tato obrazovka svojí funkčností navazuje na předchozí obrazovku Generování ODT dokumentu. Uživatel vloží původní LaTeX dokument a opravený text v dokumentu formátu ODT. Následně vygeneruje nový LaTeX dokument, jehož obsah bude odpovídat původnímu LaTeX dokumentu se změnami v textu po opravě. Z důvodu jednotnosti bude tato obrazovka vypadat relativně podobně jako obrazovka předešlá.

4.3.1.4 Konfigurace

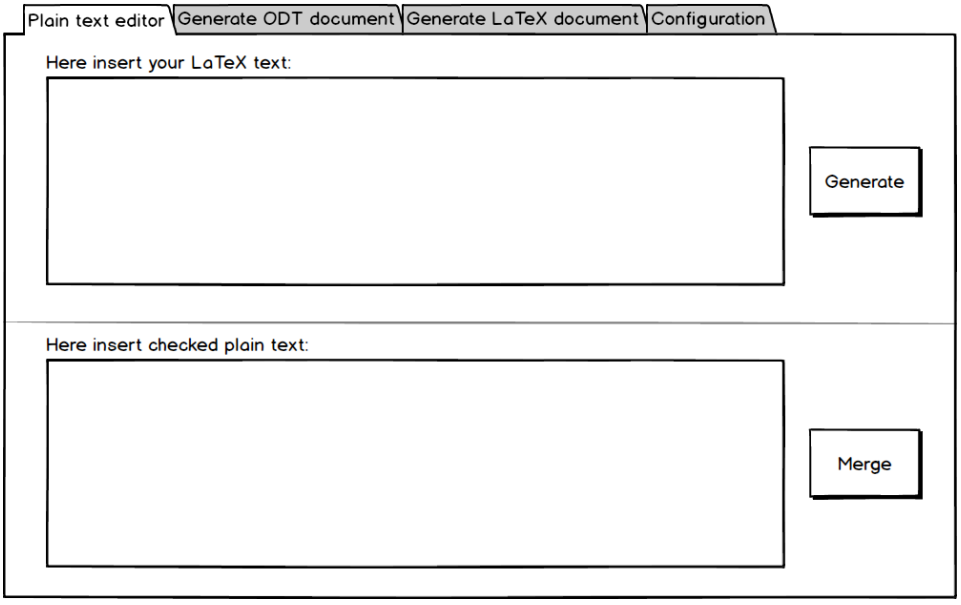
Návrh rozložení obrazovky konfigurací je znázorněn na obrázku 4.8 a je grafickou realizací případu užití U3 (Nastavení aplikace (konfigurace)). Tato obrazovka slouží k úpravě konfigurací, které umožňují upravit chování aplikace při generování textu pro jazykovou kontrolu a při následné aplikaci provedených změn do původního textu. Z této obrazovky lze otevřít konfiguraci seznamu příkazů, která umožňuje definovat podrobnější chování aplikace pro jednotlivé příkazy. Návrh obrazovky konfigurací příkazů je znázorněn na obrázku 4.9.

Dále je možné otevřít konfiguraci seznamu prostředí, která se chová velice podobně jako konfigurace seznamu příkazů a návrh obrazovky pro tuto konfiguraci je na obrázku 4.10. Poslední položkou konfigurací je výběr strategie pro přidávání varujících komentářů do textu, například v případě, kdy nebylo možné rozhodnout správnou aplikaci provedené změny a mohlo tedy dojít k chybě.

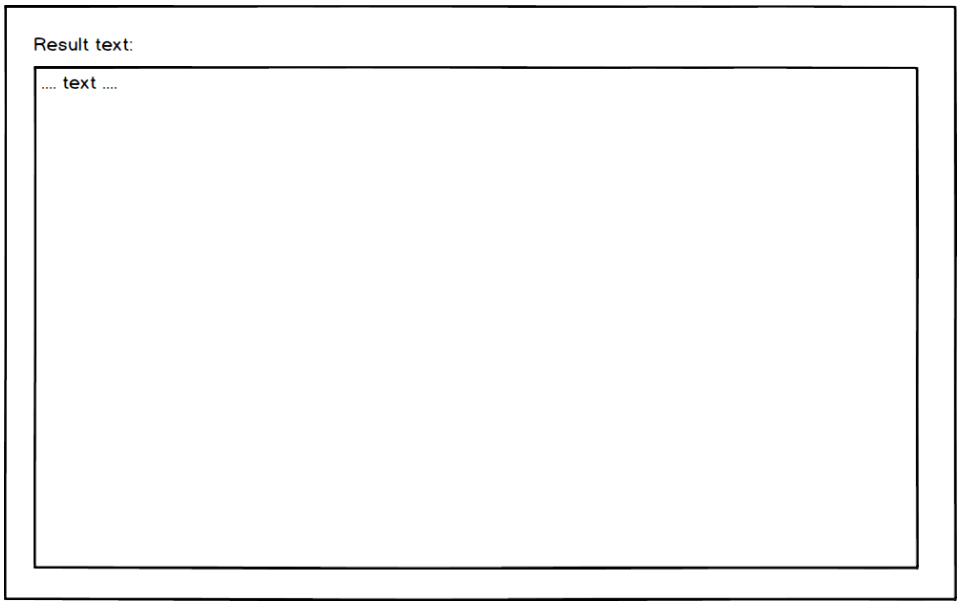
Obrázek 4.3: Návrh menu aplikace



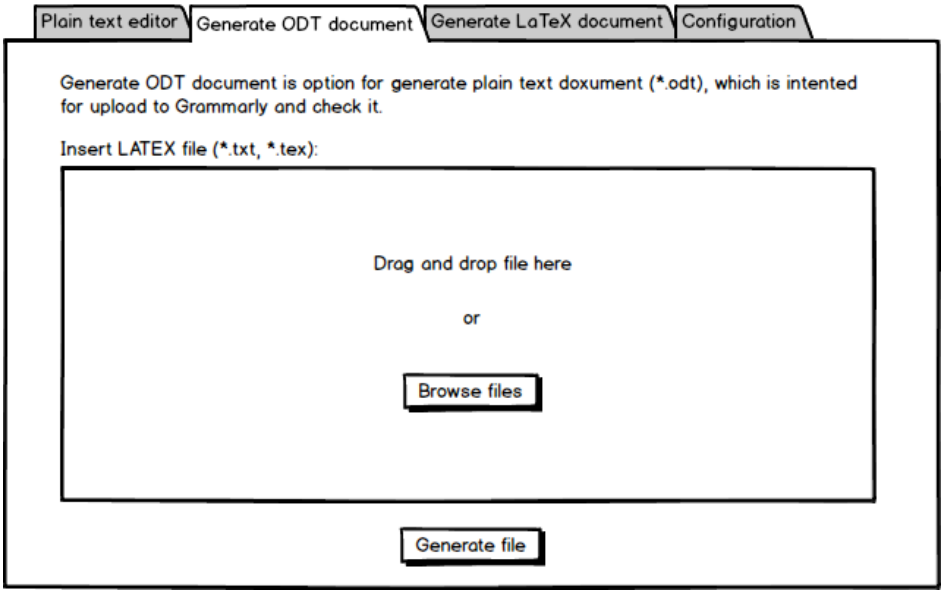
Obrázek 4.4: Návrh výchozí obrazovky



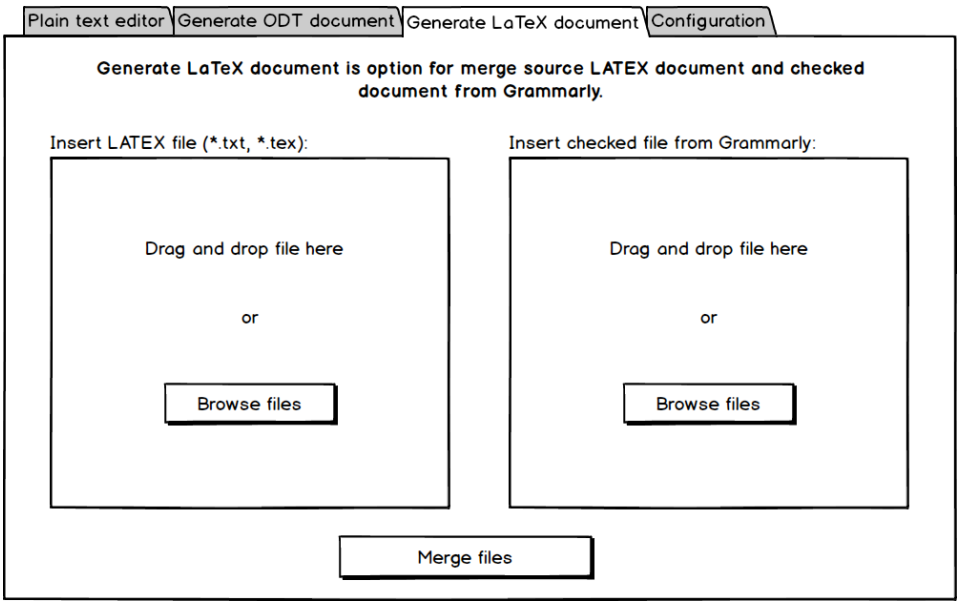
Obrázek 4.5: Návrh obrazovky pro zobrazení vygenerovaného textu



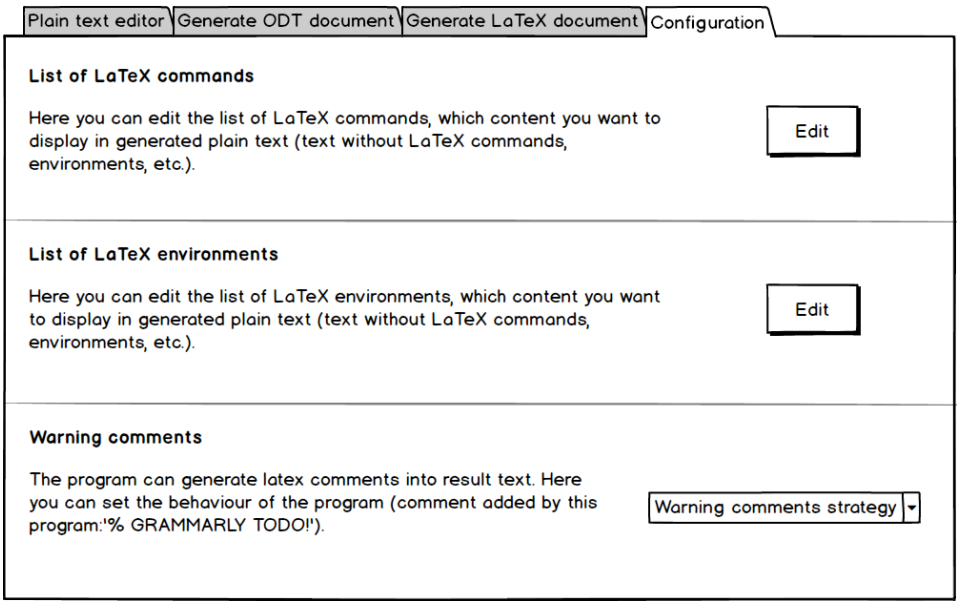
Obrázek 4.6: Návrh obrazovky pro generování ODT dokumentu



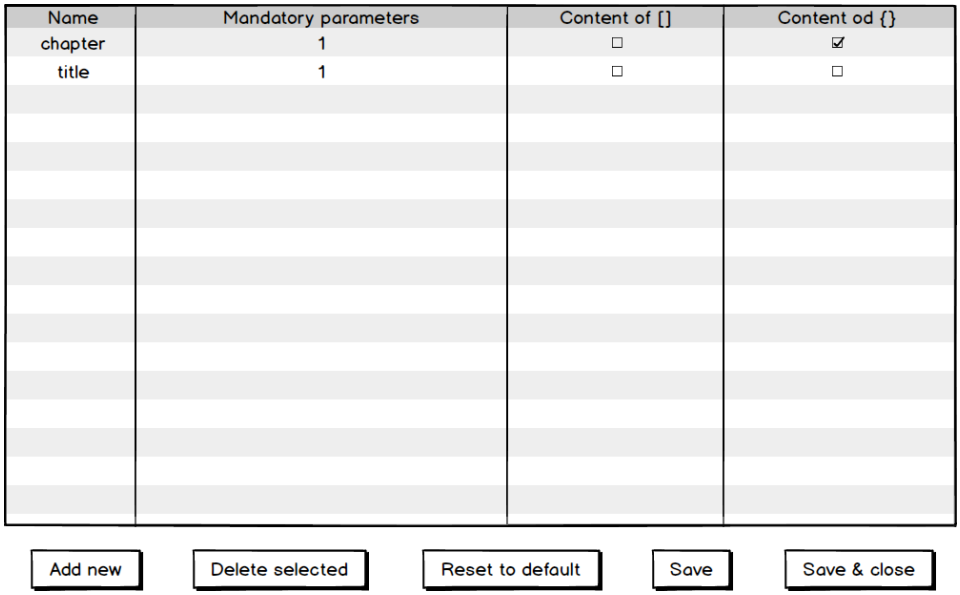
Obrázek 4.7: Návrh obrazovky pro sloučení zdrojového a opraveného textu uložených v příslušných dokumentech



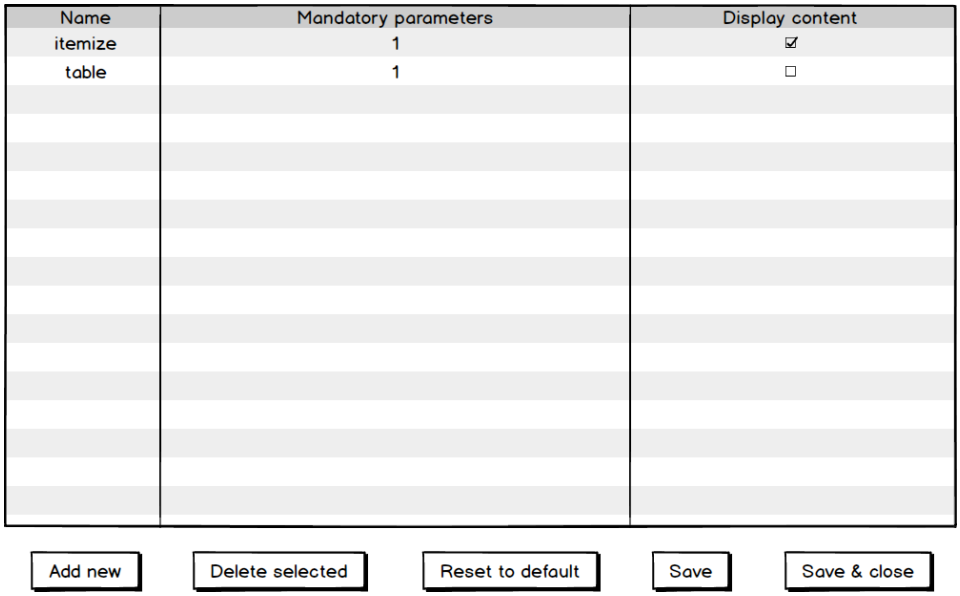
Obrázek 4.8: Návrh obrazovky konfigurací



Obrázek 4.9: Návrh obrazovky konfigurací příkazů



Obrázek 4.10: Návrh obrazovky konfigurací prostředí



4.3.2 Implementace grafického rozhraní

Tato kapitola se bude zabývat popisem implementace navrženého grafického rozhraní, tedy z jakých komponent a jak je realizován dříve uvedený návrh obrazovky. Proto se v této kapitole budou často vyskytovat názvy komponent JavaFX v anglickém jazyce.

Pro implementaci základu grafického rozhraní byla jako rodičovská (nebo též kořenová) komponenta použit `AnchorPane`. Anchor znamená v překladu kotva, což je výstižné označení výhody právě této komponenty. Kotvy totiž definují, jak se komponenta přichytává k okrajům rodičovské komponenty. To znamená, že při vložení nové komponenty do `AnchorPane` je možné definovat kotvy mezi `AnchorPane` a vloženou komponentou. Při změně velikosti okna rodičovská komponenta (zde `AnchorPane`) přenese tuto změnu i na komponenty pod ní podle nastavených kotev. Rodičovskou komponentou je tedy `AnchorPane` a komponentou pod ní je `TabPane`, který umožňuje vkládat nové záložky (Tab) a který má ukotvené všechny čtyři strany k rodičovské komponentě.

V rámci implementace jsou výjimečně používány i css styly, které umožňují hromadně upravovat vzhled některých komponent. Css styly jsou použity například pro definici barvy pozadí grafického rozhraní aplikace. Díky tomu je barva pozadí definována pouze na jednom místě a případná změna barvy na tomto místě změní barvu pozadí v grafickém rozhraní celé aplikace.

4.3.2.1 Editor prostého textu

Obsah této záložky je implementován následovně. Hlavní komponentou je `VBox`, což je kontejner, do kterého se vkládají další komponenty, přičemž tyto komponenty jsou řazeny vertikálně (pod sebe). Uvnitř kontejneru `VBox` jsou další dva kontejnery `HBox`. `HBox` je kontejner, do kterého se vkládají další komponenty, a tyto komponenty jsou řazeny horizontálně (vedle sebe). Tyto dva `HBox` kontejnery jsou pro vizuální přehlednost odděleny horizontální komponentou `Separator` (oddělovač) a jelikož mají velmi podobnou strukturu, bude zde popsán pouze první z nich. Uvnitř `HBox` kontejneru je umístěn `Label` (textový popisek), `ScrollPane` a `Button` (tlačítko) s již zmíněným nápisek Generate. Zmíněný `Label` a `ScrollPane` jsou zabaleny uvnitř ještě jednoho `VBox` kontejneru. To je z důvodu, aby byly seřazeny pod sebou. Komponenta `Label` obsahuje jednu z instrukcí uvedených v návrhu této záložky, aby uživatel věděl, který text kam vložit. `ScrollPane` je kontejner, který zobrazí posuvník v případě, že velikost tohoto kontejneru není dostatečná a obsah by tedy jinak nebyl vidět. `ScrollPane` v tomto případě obsahuje komponentu `TextArea`, do které je možné vkládat textové řetězce.

4.3.2.2 Generování ODT dokumentu

Při implementaci byl opět jako hlavní kontejner pro tuto záložku zvolen `VBox`. Ten obsahuje `Label`, který krátce popisuje (v angličtině), k čemu tato záložka slouží (text je shodný s textem v návrhu této obrazovky). `VBox` kontejner dále obsahuje komponentu `AnchorPane`, v následujícím textu bude tato konkrétní komponenta označována jako `Drap&Drop` komponenta, a `HBox`. `HBox` kontejner obsahuje pouze `Button` s nápisem Generate file (Vygeneruj soubor) a je zde použit, aby bylo tlačítko zarovnané na střed okna aplikace.

Zmíněná komponenta `Drag&Drop`, je použita v implementaci GUI vícekrát, a proto zde bude nyní podrobněji popsána. Tato komponenta je určena pro již zmíněnou funkcionalitu nahrávání zdrojového dokumentu – přetažení dokumentu do vyznačené oblasti, nebo zvolení cesty k souboru na disku. Komponenta obsahuje dva `VBox` kontejnery. Oba tyto kontejnery mají ukotvené všechny čtyři kotvy (neboť komponenta `Drag&Drop` je vlastně `AnchorPane`) a přesně se překrývají (jsou umístěny na sobě). Z těchto dvou kontejnerů je však vždy zobrazen jen jeden (druhý je neviditelný a tedy neaktivní). První `VBox` kontejner je znázorněn i na obrázku 4.6 a je aktivní, dokud uživatel nenahraje LaTeX dokument. Jakmile uživatel nahraje zdrojový dokument, je první `VBox` skryt. Místo něj se zobrazí druhý `VBox` kontejner, který uživateli oznamuje, že soubor byl úspěšně nahrán. První `VBox` kontejner obsahuje:

- `Label` s instrukcí „Insert LATEX file (*.txt, *.tex):“[6],
- `Region`, což je velmi obecný prvek definující pouze místo / určitou oblast,
- dvě komponenty `Label` s instrukcemi „Drag and drop file here“ a „or“[7],
- `Button` s nápisem Browse files (Procházet soubory).

Zmíněná komponenta `Region` je zde použita v kombinaci s css styly a je v ní zobrazen symbolický obrázek pro přehlednější a názornější komunikaci s uživatelem (jedná se o obrázek B.1). Druhý `VBox` kontejner obsahuje:

- `Region`, který je opět použit v kombinaci s css styly a je v něm zobrazen symbolický obrázek pro přehlednější a názornější komunikaci s uživatelem (jedná se o obrázek B.2),
- `Label` obsahující výpis cesty k souboru na disku,
- `Button` s nápisem Cancel (Zrušit).

Komponenta `Drag&Drop` používá i css styly. Pokud nebyl nahrán zdrojový dokument, tak je obvod komponenty zvýrazněn přerušovanou čárou, barva

[6] v překladu „Vložte LaTeX soubor (*.txt, *.tex):“
[7] v překladu „Přetáhněte a pusťte soubor sem“ a „nebo“

pozadí komponenty je šedá a je použit již zmíněný obrázek B.1. Pokud uživatel drží nad komponentou zdrojový dokument, tak je změněna barva pozadí komponenty a komponenta navíc lehce zprůsvitní. Pokud byl nahrán zdrojový dokument, tak je obvod komponenty zvýrazněn plnou čarou, barva pozadí je lehce zelená a je změněn i obrázek symbolizující nahraný dokument B.2.

4.3.2.3 Generování LaTeX dokumentu

Pro implementaci byl opět jako hlavní kontejner použit `VBox`. Ten obsahuje `Label` s krátkým popisem (v angličtině), k čemu tato záložka slouží (text je shodný s textem v návrhu obrazovky). `VBox` kontejner dále obsahuje dva `HBox` kontejnery. V prvním `HBox` kontejneru jsou dvě instance dříve popsané `Drag&Drop` komponenty, které mají jen drobné úpravy v textovým instrukcích pro uživatele (viz návrh obrazovky 4.7). V druhém `HBox` kontejneru je `Button` s nápisem Merge files (Spoj soubory), podobně jako tomu bylo v případě předešlé obrazovky.

4.3.2.4 Konfigurace

Implementace základní obrazovky konfigurací je velice prostá a základním kontejnerem je `VBox`. Následně pro každou ze tří konfigurací byla použita vzájemně podobná struktura v podobě `HBox` kontejneru, který obsahuje dvě `Label` komponenty a `Button` komponentu s nápisem Edit (upravit). Zmíněné dvě `Label` komponenty, představující název konfigurace a popis konfigurace (jak je vidět i z návrhu obrazovky na obrázku 4.8), jsou ještě navíc zabaleny do `VBox` kontejneru, aby byly zarovnány pod sebe. Pro konfiguraci varujících komentářů není použita jako třetí komponenta `Button`, ale `ComboBox`, která funguje na principu rozbalovací nabídky a umožňuje výběr z několika předdefinovaných hodnot.

Implementace okna konfigurací příkazů a prostředí je stejná až na trochu jiné sloupce v tabulce, proto zde bude popsána pouze implementace konfigurací příkazů. Jelikož se jedná o nové okno, je opět důležité zmínit, že kořenovou komponentou i tohoto okna je `AnchorPane`, který má pod sebou opět `VBox`. `VBox` obsahuje dva prvky a to jsou `ScrollPane` a `HBox`. `ScrollPane` má v sobě komponentu `TableView` (tabulka) a je zde opět použit pro případ, kdy by bylo v tabulce umístěno příliš mnoho dat a nebylo by možné všechny data jinak zobrazit. Tabulka má definované sloupce dle již uvedeného návrhu. Kontejner `HBox` obsahuje horizontálně seřazené komponenty `Button` s nápisy Add new (přidat nový), Delete selected (smazat vybraný), Reset to default (resetovat konfigurace), Save (uložit), Save & close (uložit a zavřít).

Kapitola 5

Testování

5.1 Průběžné testování

V rámci implementace prototypu byly manuálně prováděny pravidelné smoke testy. Tyto testy slouží pro rychlou kontrolu, zda všechny dříve implementované části aplikace stále fungují správně. Tyto testy byly prováděny po každé větší úpravě prototypu.

Zmíněné testy obsahují například text, ve kterém se testuje správná práce se symboly se speciálním významem. Tuto část textu můžete vidět ve výpisu 5.1. Tento testovaný text je velice citlivý a snadno se na něm projeví případná chyba, neboť je potřeba odstranit klíčová slova jazyka LaTeX a odstranit zpětná lomítka před znaky se speciálním významem. Podobně velice citlivý je i text ve výpisu 5.2, kde je použito několik do sebe vnořených bloků. Navíc hned první slovo v tomto testovacím textu je špatně a bude v rámci kontroly opraveno, což může způsobit další problémy. Kompletní text používaný pro tyto testy se nachází v příloze D.

```
\section{Special characters}

Test escaping special symbols:
\begin{itemize}
    \item and: \&,
    \item percent: \%,
    \item dollar: \$,
    \item hashtag: \#,
    \item underscore character: \_,
    \item 1. bracket: \{,
    \item 2. bracket: \}.
\end{itemize}
```

Výpis kódu 5.1: Testování znaků se speciálním výrazem

```
\subsection{Prague}

{Much {{visitor} to Prague} never} \\
set foot outside the Old Town, Castle \\
and Mala Strana.
```

Výpis kódu 5.2: Testování rekurze

5.2 Finální testování

V rámci finálního testování byly otestovány všechny funkce prototypy aplikace. Především bylo otestováno použití konfigurací, jejich správné ukládání a načítání při příštím spuštění aplikace. Dále bylo testováno použití aplikace pro již zmíněný testovací LaTeX text, další texty nalezené na internetu a na text poskytnutý vedoucím bakalářské práce.

Na základě tohoto testování byly odhaleny a dále popsány následující chyby.

- Konfigurace příkazů a prostředí neukládaly počet povinných parametrů a při příštím startu prototypu aplikace došlo k automatickému nastavení této hodnoty na -1, nespecifikováno. Tento problém byl opraven.
- Neodstraňování symbolu vlnovky, tyldy. Tilda má v LaTeX textu význam nezlomitelné mezery, a proto by ideálně měla být nahrazena obyčejnou mezerou a při slučování zdrojového a opraveného textu navrácena zpět do textu. Prototyp však neumožňuje nahrazování znaků. Proto byla doplněna automatická náhrada tyldy za mezeru bez funkcionality navrácení tohoto znaku zpět na jeho původní místo. Jelikož se jedná o vcelku běžný a významný symbol jazyka LaTeX, byla doplněna informace o tomto problému do návrhu aplikace.
- Při testování se objevily LaTeX příkazy, mající velmi specifický zápis, na který se v době tvorby návrhu zapomnělo. Jedná se například o příkaz `\verb`, za kterým může následovat libovolný znak a konec tohoto příkazu je v místě zopakování zvoleného znaku. Návrh aplikace fungující i pro tyto netypické příkazy by proto mohl být jedním z možných pokračováních této práce.

5.3 Zhodnocení prototypu

V prototypu aplikace jsou implementovány všechny funkce navrženého systému. Tyto funkce byly v rámci této práce otestovány, jak již bylo řečeno. Výsledný prototyp aplikace umí z přijatého LaTeX textu vygenerovat text bez LaTeX výrazů a po opravě aplikovat tyto změny do původního textu.

Prorotyp má však problémy s netradičními příkazy, jako je například již zmíněný `\verb` nebo tilda. Prototyp také ne vždy správně pracuje s přijatým dokumentem ve formutu ODT a v důsledku se z dokumentu nepovede načíst žádný text. Zmíněné chyby jsou však spíše minoritní a dá se jim vyhnout. Za běžných podmínek prototyp aplikace funguje velmi dobře a určitě umožní a ulehčí jazykovou kontrolu textu.

Závěr

Cílem bakalářské práce bylo provést rešerši služby Grammarly a zjistit možnosti, jak využít tuto službu v aplikaci, která by umožnila jazykovou kontrolu v LaTeX dokumentech. Na základě této rešerše navrhnout a implementovat prototyp aplikace, a poté jej na závěr otestovat. Jedná se o první pokus vytvořit takovou aplikaci ve spojitosti se službou Grammarly. Obdobných programů ve spojitosti s jinými službami také moc není.

Na základě provedené rešerše služby Grammarly byla vytvořena analýza požadavků, a následně tyto funkce byly úspěšně implementovány ve vytvořeném prototypu. Aplikace má uživatelsky velmi přívětivý vzhled, což jistě pomůže k rychlému pochopení fungování aplikace. Aplikace má i několik drobných chyb, které již byly popsány v kapitole 5.

Navazující práce by se mohly zaměřit například na:

- možnost konfigurací příkazů se speciálním či netradičním zápisem,
- možnost nahrazení znaku (či skupiny znaků) za zástupný symbol, což by bylo velmi přínosné především ve větách, jejichž součástí je matematický výraz, a následné vrácení tohoto znaku/znaků při slučování textů,
- možnost konfigurací pro funkci nahrazování znaků.

Bibliografie

1. GRAMMARLY INC. *Grammarly* [online software]. 2019 [cit. 2019-04-23]. Dostupné z: https://www.grammarly.com/.
2. LANGUAGETOOLER GMBH. *LanguageTool: proofreading software* [online software]. 2019 [cit. 2019-04-23]. Dostupné z: https://languagetool.org/.
3. Blog: How to Select Your English Dialect | Grammarly Spotlight. In: *Grammarly* [online] [cit. 2019-04-18]. Dostupné z: https://www.grammarly.com/blog/how-to-switch-dialects/.
4. Blog: How does Grammarly work? In: *Grammarly* [online] [cit. 2019-04-18]. Dostupné z: https://support.grammarly.com/hc/en-us/articles/115000090871-How-does-Grammarly-work-.
5. GRAMMARLY SUPPORT. *Re: I have another question* [elektronická pošta]. Zasláno z: support@grammarly.zendesk.com. 2018-11-26 [cit. 2019-03-30] Osobní komunikace.
6. Blog: How do I preserve text formatting? In: *Grammarly* [online] [cit. 2019-04-18]. Dostupné z: https://support.grammarly.com/hc/en-us/articles/115000090832-How-do-I-preserve-text-formatting-.
7. Blog: What are the limitations when using Grammarly? In: *Grammarly* [online] [cit. 2019-04-20]. Dostupné z: https://support.grammarly.com/hc/en-us/articles/115000090911-What-are-the-limitations-when-using-Grammarly-.
8. *TeXtidote: A correction tool for LaTeX documents* [online]. Sylvain Hallé [cit. 2019-04-28]. Dostupné z: https://sylvainhalle.github.io/textidote/.

9. HALLÉ, Sylvain. Readme.md. In: *TeXtidote: a correction tool for LaTeX documents and other formats, GitHub* [online] [cit. 2019-04-11]. Dostupné z: `https://github.com/sylvainhalle/textidote/blob/master/Readme.md`.
10. HALLÉ, Sylvain. *Screenshot.png* [online obrázek] [cit. 2019-04-12]. Dostupné z: `https://raw.githubusercontent.com/sylvainhalle/textidote/master/docs/assets/images/Screenshot.png` Obrázek z dokumentace programu TeXtidote.
11. *PM Consulting: Funkční požadavky* [online] [cit. 2019-04-19]. Dostupné z: `https://www.pmconsulting.cz/slovnikovy-pojem/funkcni-pozadavky/`.
12. *PM Consulting: Funkční požadavky* [online] [cit. 2019-04-19]. Dostupné z: `https://www.pmconsulting.cz/slovnikovy-pojem/nefunkcni-pozadavky/`.
13. Případy užití (Use Cases). In: *OCUP.CZ* [online] [cit. 2019-04-21]. Dostupné z: `http://ocup.ocup.cz/2010/07/pripady-uziti-use-cases.html`.
14. *Oracle: Oracle Java SE Support Roadmap* [online] [cit. 2019-04-20]. Dostupné z: `https://www.oracle.com/technetwork/java/java-se-support-roadmap.html`.
15. LEARN JAVAFX: JavaFX Tutorial. In: *tutorialspoint* [online] [cit. 2019-04-20]. Dostupné z: `https://www.tutorialspoint.com/javafx/`.
16. *JAXB: Java Architecture for XML Binding* [online]. Oracle [cit. 2019-06-01]. Dostupné z: `https://www.oracle.com/technetwork/articles/javase/index-140168.html`.
17. GOOGLE INC. *Diff-match-patch* [online software]. 2019 [cit. 2019-06-01]. Dostupné z: `https://github.com/google/diff-match-patch`.
18. Wordprocessing Text: Formatting. *Open Office XML* [online] [cit. 2019-06-05]. Dostupné z: `http://officeopenxml.com/WPtextFormatting.php`.
19. Visitor. *Algoritmy.net* [online] [cit. 2019-06-10]. Dostupné z: `https://www.algoritmy.net/article/1643/Visitor`.
20. *Preferences API Overview* [online]. Oracle [cit. 2019-06-20]. Dostupné z: `https://docs.oracle.com/javase/8/docs/technotes/guides/preferences/overview.html`.
21. HADRIEN. *Download icon* [online obrázek] [cit. 2019-03-25]. Dostupné z: `https://www.flaticon.com/free-icon/download_127942#term=download&page=3&position=61`.

22. PHAM, Linh. *Interface* [online obrázek] [cit. 2019-03-25]. Dostupné z: https://www.flaticon.com/free-icon/file-interface-symbol-of-text-paper-sheet_54551#term=text%20file%20icon&page=1&position=17.

Příloha **A**

Seznam použitých zkratek

GUI Graphical user interface

XML Extensible markup language

API Application Programming Interface

ODT OASIS Open Document Format for Office Applications

LTS Long-term support

AWT Abstract Window Toolkit

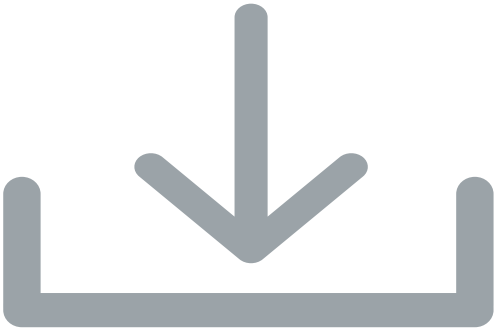
GNU GNU's Not Unix!

LF Line Feed

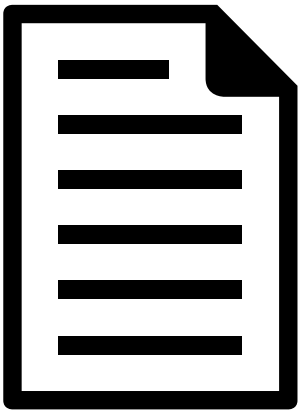
Příloha B

Seznam dalších obrázků

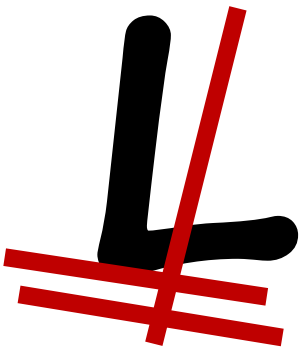
Obrázek B.1: Ikona pro vložení souboru [21]



Obrázek B.2: Ikona pro nahraný textový soubor [22]



Obrázek B.3: Autorem této práce vytvořené logo pro prototyp aplikace



Příloha C

E-mailová komunikace se službou Grammarly

Přepis e-mailové komunikace mezi autorem práce a službou Grammarly [5] (v anglickém jazyce):

Tomáš Hojek (Nov 24, 12:09 PM PST)

Dear Sir/Madam,

I am a student of CTU (The Czech Technical University in Prague), Faculty of Information Technology, and I am starting to think about a topic of my bachelor thesis. Lots of students of technical universities write their bachelor thesis using TeX (or LaTeX) and, in my opinion, it is annoying to copy the whole text and then remove all keywords and macros to check the correctness of the actual text.
I have decided to make a plugin (for now I am thinking about a plugin for Chrome). This plugin, on click, opens a new browser window and displays a text field. This plugin would enable the user to write a text into the text field and would check the correctness of English text inside while ignoring LaTeX keywords.
So, my request is: Could you, please, provide me with your web API to check the correctness of English texts? I do not intend to use the plugin for any financial gain, only to complete my thesis. After I will finish my bachelor thesis, I would like to give this tool/plugin to you as a gift and as my thanks to you (for providing the web API).

Best regards,

Tomáš Hojek
hojektom@fit.cvut.cz

Ann (Grammarly Support) (Nov 26, 4:07 AM PST)

Hello Tomáš,

Thanks for your message!

We currently do not have the Grammarly API on our active development roadmap, but we constantly evaluate these priorities, so I will pass the request along to our product team.

As an alternative, you could purchase Grammarly Business until we finish the API. If you would like to learn more about Grammarly for your workplace, please visit https://www.grammarly.com/business.

Best wishes,
Ann

Autorův volný překlad e-mailové komunikace s podporou služby Grammarly:

Tomáš Hojek (Listopad 24, 12:09)

Vážený pane/paní,

jsem studentem ČVUT v Praze, Fakulta informačních technologií, a přemýšlím nad tématem mé bakalářské práce. Mnoho studentů technických vysokých škol píše své závěrečné práce v jazyku TeX (nebo LaTeX) a je podle mě nepříjemné kopírování textu a následné mazání všech klíčových slov a maker LaTeXu kvůli jazykové kontrole tohoto textu. Rozhodl jsem se vytvořit plugin (aktuálně přemýšlím nad pluginem do prohlížeče Chrome). Tento plugin, na kliknutí, otevře nové okno prohlížeče a zobrazí textové pole. Tento plugin by umožnil uživateli psát text do textového pole a kontrolovat správnost anglického textu ignorujíce LaTeX výrazy.
Moje otázka zní: Můžete mi prosím poskytnout API rozhraní pro kontrolu správnosti anglických textů? Nechci používat plugin pro jakýkoliv finanční zisk, pouze pro mou závěrečnou práci. Po dokončení práce bych vám výsledný plugin daroval jako mé poděkování (za poskytnuté API).

S pozdravem,

Tomáš Hojek
hojektom@fit.cvut.cz

Ann (Grammarly Support) (Listopad 26, 4:07)

Ahoj Tomáši,

děkujeme za vaši zprávu!

V současné době neplánujeme vývoj Grammarly API, ale neustále vyhodnocujeme priority vývoje, takže předám Váš požadavek našemu produkčnímu týmu.

Jako alternativu si můžete koupit Grammarly Business, dokud nedokončíme API. Pokud se chcete dozvědět více o Grammarly, prosím navštivte https://www.grammarly.com/business.

S pozdravem,
Ann

Příloha D

Testovací text

Následující text v jazyce LaTeX byl používán na smoke testy během implementace prototypu.

```
\documentclass{article}
\usepackage{graphicx}

 % Comment

\begin{document}

\title{Introduction to test \LaTeX{}} %comment

\author{An Author's Name}

\maketitle

\begin{abstract}
This is abstract text: This simple document contains basic
↪  features of \LaTeX{}.
\end{abstract}

\section{Basic math $c^2 = a^2 + b^2$}

In this section are tested using of mathematic.
It is called "Pythagoras' Theorem" and can be written in one
↪  short equation: $$a2 + b2 = c2$$.
It is called "Pythagoras' Theorem" and can be written in one
↪  short equation: \[a2 + b2 = c2\].
It is called "Pythagoras' Theorem" and can be written in one
↪  short equation: \(a2 + b2 = c2\).
```

```
24 It is called "Pythagoras' Theorem" and can be written in one
   ↪ short equation: $a2 + b2 = c2$.
25 \begin{equation}
26     \label{simple_equation}
27     \alpha = \sqrt{ \beta }
28 \end{equation}
29
30 \section{English mistakes}
31
32 This section is simple test for checking English in latex
   ↪ documents.
33
34 \subsection{Prague}
35
36 {Much {{visitor} to Prague} never} \\
37 set foot outside the Old Town, Castle \\
38 and Mala Strana.
39
40 \subsection{Castles}
41
42 There is many reasons to visit Czech Republic. Karlstejn is one
   ↪ of oldest and most important castles in the czech republic.
43
44 ...
45
46 \section{Graphics}
47
48 \begin{figure}
49     \centering
50     \includegraphics[width=3.0in]{myfigure}
51     \caption{Simulation Results}
52     \label{simulationfigure}
53 \end{figure}
54
55 \section{Special characters}
56
57 Test escaping special symbols:
58 \begin{itemize}
59     \item and: \&,
60     \item percent: \%,
61     \item dollar: \$,
62     \item hashtag: \#,
63     \item underscore character: \_,
64     \item 1. bracket: \{,
```

```
65      \item 2. bracket: \}.
66  \end{itemize}
67
68  \end{document}
69
70  test of right parsing and working with end of the file
```

Příloha E

Obsah přiložené SD karty

- `readme.txt` ..... stručný popis obsahu SD karty
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF