FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Editor zdrojových kódů WooWoo dokumentů
Student:	David Straka
Vedoucí:	Ing. Tomáš Kalvoda, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Seznamte se s programem a formátem WooWoo, aktuálně využívaným pro vytváření více-formátových (PDF, HTML, ePub) studijních textů pro předměty BI-PKM a BI-ZMA.

Prozkoumejte možnosti moderních editorů, či integrovaných vývojových prostředí, (jako např. Atom, VS Code, atp.) pro vytváření jejich rozšíření a modifikovatelnost.

Navrhněte a implementujte rozšíření vámi vybraného editoru o nástroje usnadňující autorům studijních textů práci s WooWoo dokumenty. Zejména se soustřeďte na přehlednou prezentaci logické struktury dokumentu, zobrazování matematických výrazů a různých grafických objektů, navigaci mezi různými částmi dokumentu a vyhledávání pomocí labelů objektů. Využijte styl používaný v existujících HTML a PDF šablonách.

Svou implementaci v rámci možností otestujte (jednotkové testy, uživatelské testování).

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 4. listopadu 2020 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Editor zdrojových kódů WooWoo dokumentů

David Straka

Katedra softwarového inženýrství
Vedoucí práce: Ing. Tomáš Kalvoda, Ph.D.

13. května 2021

Poděkování

Děkuji vedoucímu mé práce Ing. Tomáši Kalvodovi, Ph.D. za četné konzultace, zpětnou vazbu, cenné rady a za čas, který vedení této práce věnoval. Také mu děkuji za zajímavé téma, za kterým stojí především snaha poskytnout studentům kvalitní studijní materiály. V neposlední řadě bych rád poděkoval mé rodině za podporu během studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při 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. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, 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.

V Praze dne 13. května 2021 . . . . . .

Odkaz na tuto práci

Straka, David. *Editor zdrojových kódů WooWoo dokumentů*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021. Dostupný také z WWW: ⟨`https://github.com/davidstraka2/bachelors-thesis`⟩.

Abstrakt

Tato práce se zabývá návrhem, implementací a otestováním rozšíření editoru Atom pro usnadnění tvorby WooWoo dokumentů. Důraz je kladen zejména na přehlednou prezentaci logické struktury dokumentů využívajících existující šablony. Součástí práce je také seznámení se s formátem WooWoo, rešerše možností rozšíření stávajících editorů nebo možností zobrazování matematických výrazů a různých grafických objektů.

Klíčová slova WooWoo editor, rozšíření editoru Atom, Atom package, WooWoo formát

Abstract

This thesis deals with the design, implementation, and testing of an extension of the Atom editor for streamlining the creation of WooWoo documents. Emphasis is put especially on clear presentation of the logical structure of documents using the existing templates. Familiarization with the WooWoo format, research of the possibilities of the extension of current editors and possibilities of displaying mathematical expressions and different graphic objects are all also part of the thesis.

Keywords WooWoo editor, Atom editor extension, Atom package, WooWoo format

Obsah

Seznam obrázků

Seznam výpisů kódu

Úvod

Pro tvorbu materiálů pro podporu studia, jako jsou skripta, handouty ke cvičením a různé další studijní materiály, lze využít mnoha různých formátů, které mají každý své výhody a nevýhody. Zejména techničtější předměty v materiálech typicky zahrnují množství různých matematických výrazů, diagramů, grafů nebo výpisů kódu. Dlouhodobě využívaným standardem pro tvorbu těchto typů dokumentů je LaTeX, díky jeho rozsáhlým typografickým možnostem pro velice přesnou sazbu. Tento formát však má strmější učící křivku oproti například Word dokumentům. Dalším z problémů tohoto formátu je překlad do HTML (nebo také EPUB), jenž existuje pouze v nepříliš kvalitní podobě díky různým externím nástrojům.

Na Fakultě informačních technologií Českého vysokého učení technického v Praze proto vznikl z iniciativy Ing. Tomáše Kalvody, Ph.D. z Katedry aplikované matematiky zcela nový formát nazývaný WooWoo, již v současnosti využívaný pro tvorbu studijních textů pro více předmětů.

Tento formát se snaží vyřešit problém oddělení samotného obsahu dokumentů od informací o způsobu jejich zobrazení a spolu s tím problém kvalitního víceformátového (HTML a PDF; v budoucnu plánován také EPUB) výstupu z jednoho zdroje. Zatím však pro něj neexistuje podpora v žádném editoru a autor dokumentů je tak odkázán na psaní strohého zdroje, kde jedinou zpětnou vazbou je výsledný výstup, jehož vygenerování navíc trvá nezanedbatelnou dobu (v případě materiálů pro výše zmíněné předměty se jedná i o více než 10 minut).

Cílem této práce je seznámit se s tímto nově vzniklým formátem, prozkoumat možnosti moderních editorů či vývojových prostředí, zejména co se týče jejich rozšiřitelnosti a navrhnout, implementovat a otestovat rozšíření vybraného editoru, jehož účelem má být právě poskytnutí chybějící okamžité zpětné vazby při tvorbě WooWoo dokumentů a tím zjednodušení práce s nimi. Zejména jde pak o přehlednou prezentaci logické struktury dokumentů, zobrazování matematických výrazů a různých grafických objektů (TikZ diagramy),

navigaci mezi různými částmi dokumentu, vyhledávání pomocí značek objektů nebo zvýrazňování WooWoo syntaxe. Při plnění těchto cílů bude využito stylu založeného na existujících šablonách. Výsledek by měl být multiplatformní a nezávislý na internetovém připojení.

Kapitola 1

Rešerše a analýza

1.1 Analýza požadavků

Na editor je kladena řada funkčních i nefunkčních požadavků, které jsou blíže popsány níže.

Mezi funkční požadavky patří následující:

F1 zvýrazňování syntaxe WooWoo formátu,

F2 zobrazení náhledu dokumentu nebo jeho části,

F3 živá aktualizace náhledu dle aktuálně se měnícího obsahu dokumentu,

F4 zobrazení matematických výrazů v náhledu dokumentu,

F5 zobrazení grafických objektů v náhledu dokumentu,

F6 zobrazení obsahu dokumentu,

F7 zobrazení vnitřních prostředí v názvech částí v obsahu dokumentu,

F8 zobrazení přehledu značek dokumentu,

F9 vyhledávání v přehledu částí dokumentu,

F10 vyhledávání v přehledu značek dokumentu.

Mezi nefunkční požadavky pak patří zejména:

N1 multiplatformnost (editor by měl fungovat na aktuálních verzích nejpoužívanějších desktopových operačních systémů – Windows 10, macOS 10.15 a Linuxové systémy),

N2 nezávislost na internetovém připojení,

N3 možnost jednoduchého rozšíření pro podporu dalších šablon.

```
.MojeCast Nadpis
  ...
  volitelný YAML meta-blok
  ...
```

Výpis kódu 1.1: Obecný formát definice části WooWoo dokumentu

1.2 Formát WooWoo

WooWoo je nový formát pro tvorbu (zejména) studijních textů, vzniklý z iniciativy Ing. Tomáše Kalvody, Ph.D. z Katedry aplikované matematiky na Fakultě informačních technologií Českého vysokého učení technického v Praze. V současnosti je tento formát využívaný pro tvorbu studijních textů pro dva předměty, s nimiž se studenti setkávají hned v prvním semestru bakalářského studia (aktuálně dobíhající akreditace) – Základy matematické analýzy (BI-ZMA) a Přípravný kurz matematiky (BI-PKM).

Primárním cílem tohoto formátu je dle [1] kompletní oddělení zdroje dokumentů s možnými meta-informacemi od informací o způsobu jejich zobrazení. Zdroj dokumentu a případné meta-informace chce podle [1] formát prezentovat v člověkem čitelné podobě. Dále se tento formát v kombinaci s šablonami snaží vyřešit problém kvalitního víceformátového (HTML a PDF; v budoucnu plánován také EPUB) výstupu z jednoho zdroje.

WooWoo dokument se dle [1] může dělit do více částí a může obsahovat bloky a objekty (všechny tyto konstrukty jsou podrobněji popsány v nadcházejících podsekcích). Části dokumentu, bloky i objekty mohou být dle [1] dále doplněny o meta-bloky (odsazenými YAML bloky).

Konkrétní dostupné typy částí dokumentu, objektů a dalších konstruktů jsou dle [1] závislé na použité šabloně. Stejně tak dle [1] šablona určuje způsob nakládání s informacemi z meta-bloků.

1.2.1 Části dokumentu

Každý WooWoo dokument může dle [1] být rozdělen do více částí. Podle [1] každou definici části dokumentu tvoří na jednom řádku její typ, nadpis a následně může na dalších přímo navazujících řádcích volitelně obsahovat meta-blok (odsazený YAML blok). Typ části dokumentu je dle [1] uvozen tečkou a začíná velkým písmenem; od nadpisu je oddělen mezerou. Definice části vždy začíná na začátku řádku, nemůže mít před sebou ani bílé znaky (mezery).

Pro ilustraci viz zobecněný formát definice části dokumentu 1.1 nebo definice kapitoly studijního textu BI-PKM i s meta-blokem, která je následována blokem textu (tento studijní text využívá existující FIT Template šablony) 1.2.

```
.Chapter Úvod
  label: chap-intro

Než se pustíme do hlavního výkladu, je vhodné čtenáře nejprve
seznámit s cíli a smyslem tohoto kurzu, resp. studijního textu.
```

Výpis kódu 1.2: Část dokumentu ve zdroji studijního textu k BI-PKM [2]

```
.MujObjekt:
  ...
  volitelný YAML meta-blok
  ...

  odsazený blok


  odsazený blok


  ...
```

Výpis kódu 1.3: Obecný formát definice objektu WooWoo dokumentu

1.2.2 Objekty

Každý WooWoo dokument může dle [1] obsahovat libovolné množství objektů. Každou definici objektu tvoří dle [1] na jednom řádku jeho typ a následně může na dalších přímo navazujících řádcích volitelně obsahovat meta-blok (odsazený YAML blok). Typ objektu je dle [1] uvozen tečkou, začíná velkým písmenem a je následován dvojtečkou. Tělo objektu může obsahovat odsazené bloky, oddělené mezi sebou alespoň dvěma prázdnými řádky.

Pro ilustraci viz zobecněný formát definice objektu 1.3 nebo definice konkrétního typu objektu ze studijního textu BI-PKM i s jeho meta-blokem, která je následována bloky 1.4.

1.2.3 Bloky

Blok je dle [1] série po sobě jdoucích, jednotně odsazených řádků textu (textový blok) a vnějších prostředí, oddělených jedním prázdným řádkem. Textový blok i vnější prostředí mohou být podle [1] na dalších přímo navazujících řádcích doplněny meta-blokem (odsazeným YAML blokem). Dle [1] v sobě textový blok dále může obsahovat vnitřní prostředí. Vnější a vnitřní prostředí jsou podrobněji popsány v nadcházejících podsekcích.

```
.Question:
  difficulty: 2

  Nechť $a>b>0$. O číslech $a$ a $b$ říkáme, že jsou ve zlatém
  poměru "Hodnota tohoto poměru se také někdy nazývá zlatým
  ↪  řezem.".footnote, pokud poměr $a+b$ ku $a$ je stejný
  jako $a$ ku $b$. Jaký je tento poměr, tedy
  $\varphi = \frac{a}{b}$?

  .solution:

    $\displaystyle\varphi = \frac{1+\sqrt{5}}{2}$.
```

Výpis kódu 1.4: Objekt ve zdroji studijního textu k BI-PKM [2]

```
Čemu se rovná součet prvních $n$ lichých přirozených čísel? Tj.
čemu se rovná součet

  1 + 3 + 5 + \cdots + (2n-1) = \sum_{j=1}^n (2j-1), \quad n \in
  \mathbb{N}.

Své tvrzení dokažte.

.solution:

  $\sum_{j=1}^n (2j - 1) = n^2$. Tento vztah laskavý čtenář
  snadno dokáže pomocí matematické indukce.
```

Výpis kódu 1.5: Blok ve zdroji studijního textu k BI-PKM [2]

Pro ilustraci viz blok ze studijního textu BI-PKM 1.5. Uvedený blok obsahuje textový blok (s vnitřním prostředím), následovaný zkráceným vnějším prostředím bez hlavičky (které se ve FIT Template šabloně chová jako vnější prostředí reprezentující blokovou matematiku), následované dalším textovým blokem a vnějším prostředím `.solution`.

1.2.4 Vnější prostředí

Každý blok může dle [1] obsahovat libovolné množství vnějších prostředí. Syntaxe vnějšího prostředí je dle [1] podobná syntaxi objektů, kde jediným rozdílem je, že typ vnějšího prostředí začíná malým písmenem a tělo obsahuje pouze jeden blok textu.

```
.mojeVnejsiProstredi:
  ...
  volitelný YAML meta-blok
  ...

  odsazený blok
```

Výpis kódu 1.6: Obecný formát definice vnějšího prostředí WooWoo dokumentu

```
.quote:
  author: 'Nápis na vstupu do Platónovy Akademie'

  Bez znalosti geometrie sem nikdo nevstupuj.
```

Výpis kódu 1.7: Vnější prostředí ve zdroji studijního textu k BI-PKM [2]

Hlavičku vnějšího prostředí je dle [1] možno vynechat. Jedná se pak v podstatě o speciální blok, který je odsazen více než blok předchozí. Chování tohoho zkráceného vnějšího prostředí či speciálně odsazeného bloku pak dle [1] určuje konkrétní použitá šablona jako zkratku nějakého z vnějších prostředí.

Definice vnějšího prostředí může být dle [1] uvozena namísto tečky vykřičníkem. V takovém případě se podle [1] jedná o křehké prostředí a neprovádí se žádné další parsování nebo transformace jeho obsahu.

Pro ilustraci viz zobecněný formát definice vnějšího prostředí 1.6, definice konkrétního typu vnějšího prostředí ze studijního textu BI-PKM i s jeho meta-blokem, která je následována blokem textu (tento studijní text využívá existující FIT Template šablony) 1.7, případně definice konkrétního typu křehkého vnějšího prostředí ze stejného zdroje 1.8.

1.2.5 Vnitřní prostředí

Každý textový blok v sobě dle [1] může obsahovat libovolné množství vnitřních prostředí. Vnitřní prostředí musí být definováno na jednom řádku. Vnitřní prostředí lze dle [1] definovat dvěma způsoby: krátkou formou a verbózní formou. Typ vnitřního prostředí začíná dle [1] v obou případech malým písmenem.

Krátká forma vnitřního prostředí má dle [1] prefixovou podobu. Definici takového prostředí tvoří dle [1] jeho typ, který je uvozen tečkou a je následován dvojtečkou. Na typ prostředí dle [1] přímo navazuje tělo (text) prostředí bez jakýchkoli bílých znaků (mezer; a to před tělem i v něm).

Pro ilustraci viz zobecněný formát definice krátké formy vnitřního prostředí 1.9 nebo definice konkrétního typu krátké formy vnitřního prostředí ze studijního textu BI-PKM 1.10.

```
!tikz:
  options: 'scale=1.3'
  filename: fig-komplexni-rovina

  \def\a{1.2}
  \def\b{.8}
  \draw[thick,->] (-2,0) -- (2,0) node[above] {$\Re$};
  \draw[thick,->] (0,-2) -- (0,2) node[left] {$\Im$};
  \draw[gray,dashed] (\a,-2) -- (\a,2) (-2,\b) -- (2,\b);
  \fill[black] (\a,\b) circle (0.05) node[anchor=south west]
    {$a+\ii b$};
  \draw[thick] (.05,\b) -- (-.05,\b) node[anchor=south east]
    {$b$};
  \draw[thick] (\a,.05) -- (\a,-.05) node[anchor=north west]
    {$a$};
```

Výpis kódu 1.8: Křehké vnější prostředí ve zdroji studijního textu k BI-PKM [2]

```
... text text .vnitrniProstredi:tělo text text ...
```

Výpis kódu 1.9: Obecný formát definice krátké formy vnitřního prostředí Woo-Woo dokumentu

```
.cite:ieee754
```

Výpis kódu 1.10: Krátká forma vnitřního prostředí ve zdroji studijního textu k BI-PKM [2]

Verbózní forma vnitřního prostředí má dle [1] postfixovou podobu. Definici takového prostředí tvoří dle [1] jeho tělo (text) obklopené dvojitými horními uvozovkami. Na tělo prostředí dle [1] přímo navazuje jeho typ, který je uvozen tečkou (mezi ukončující uvozovkou těla a tečkou není mezera). Tělo prostředí může v sobě dále obsahovat další vnitřní prostředí. Na typ prostředí může dle [1] dále volitelně navazovat číselná reference do meta-bloku textového bloku, v němž je vnitřní prostředí obsaženo. Tato reference je od typu prostředí dle [1] opět oddělena tečkou.

Pro ilustraci viz zobecněný formát definice verbózní formy vnitřního prostředí 1.11 nebo definice konkrétního typu verbózní formy vnitřního prostředí (s číselnou referencí) ze studijního textu BI-PKM 1.12.

Dále dle [1] existují dvě zkrácené formy verbózních vnitřních prostředí, které používají před jeho typem namísto tečky zavináč `@`, respektive křížek `#`. Za zavináčem je pak namísto standardního formátu typu vnitřního pro-

```
... text text ...
... text "tělo tělo".vnitrniProstredi.1 text ...
... text text ...
  1:
    klic: hodnota
```

Výpis kódu 1.11: Obecný formát definice verbózní formy vnitřního prostředí WooWoo dokumentu

```
"čísla s plovoucí desetinnou čárkou".notion.1
```

Výpis kódu 1.12: Verbózní forma vnitřního prostředí ve zdroji studijního textu k BI-PKM [2]

středí číslo. Tyto zkrácené formy už nemohou být následovány číselnou referencí. Chování těchto zkrácených forem je dále určeno šablonou. Ve starší verzi WooWoo se navíc namísto zavináče používala tečka.

1.2.6 Další konstrukty

WooWoo dokument může obsahovat jednořádkové komentáře, které jsou uvozeny procentem `%`. Komentáře nemohou být v těle křehkých vnějších prostředí. V současnosti ve [1] není definován způsob, jak znak procenta vložit mimo křehká vnější prostředí tak, aby nezačal komentář.

Dokument také může být rozdělen do více souborů. Obsah z jiného souboru se do dokumentu vloží s pomocí direktivy `.include cesta/k/souboru.woo`. Tato direktiva v současnosti není ve specifikaci [1] popsána a není tak například známo, jak přesně by se vložil soubor s mezerami ve své cestě.

1.2.7 FIT Template

FIT Template je WooWoo šablona, která je aktuálně v praxi používána pro všechny existující studijní texty vytvořené s pomocí WooWoo.

Specifikace [1] uvádí, že tato šablona dělí dokument na následující části: kapitoly (`.Chapter`), sekce (`.Section`) a podsekce (`.Subsection`). Části dokumentu mají dle [1] jednu vyžadovanou položku metadat, značku `label`. Tyto značky jsou dle [1] využívány pro reference částí dokumentu.

Bloky tato šablona podle [1] považuje za odstavce textu. FIT Template dle [1] poskytuje následující objekty: `.Definition`, `.Corollary`, `.Lemma`, `.Theorem`, `.Proof`, `.Remark`, `.Example`, `.Figure`, `.Question` a `.Table`.

Dále dle [1] FIT Template poskytuje následující vnější prostředí: `.align`, `.caption`, `!codeblock`, `.enumerate`, `.equation`, `.image`, `.itemize`, `.quote`, `!sage`, `.solution`, `!tabular` a `!tikz`. Vnější prostředí bez hlavičky šablona považuje za zkratku pro prostředí `.equation`.

```
Matematika je velmi úzce spjata s tzv.
"vědeckou metodou poznání".1, o které lze bez velkého přehánění
prohlásit, že tvoří základ naší civilizace. Častým lidským cílem
je hledání hlubšího porozumnění světa a řešení různých problémů.
Matematika v této činnosti nehraje roli pouhého početní stroje.
Nejprve je nutné problém analyzovat, rozebrat ho na části a
zkoumat jejich vztahy a chování. Typicky dojde k vytvoření
matematického modelu, který (více či méně) dobře vystihuje náš
problém. Následně se v rámci tohoto abstraktního modelu snažíme
dobrat k závěrům a poznání původního problému.
  1:
    link: 'https://en.wikipedia.org/wiki/Scientific_method'
```

Výpis kódu 1.13: Starší syntaxe zkrácené formy vnitřního prostředí ve zdroji studijního textu k BI-PKM [2]

Tato šablona poskytuje dle [1] následující vnitřní prostředí: `.cite`, `.code`, `.emphasize`, `.eqref`, `.footnote`, `.item`, `.math`, `.notion`, `.quoted`, `.reference` a `.todo`. Zkrácenou formu vnitřního prostředí s mřížkou FIT Template dle [1] považuje za zkratku `.reference`, zkrácenou formu se zaviná-čem pak za číselý odkaz do meta-bloku. Dále dle [1] šablona určuje umístění textu mezi dva dolary (`$text text$`) jako zkratku pro prostředí `.math`.

1.2.8 Problematická místa

Vzhledem k tomu, že je WooWoo nový formát, některé věci v něm nejsou ještě zcela dořešené. Například možnost vložení procenta mimo křehká prostředí bez toho, aby začalo nový řádkový komentář. Stejně tak ve FIT Template šabloně ještě nejsou zcela dořešena například vnější prostředí `.enumerate`, `.itemize` či `!tabular`.

S tím jak se formát vyvíjí, obsahují existující dokumenty syntaktické prvky ze starších verzí WooWoo, jako například zkrácenou formu vnitřního prostředí s tečkou namísto zavináče. Viz příklad 1.13 přímo ze zdroje studijního textu k BI-PKM.

Existující dokumenty také obsahují prvky, které čistě z pohledu specifikace WooWoo formátu nejsou zcela validní. Například prostředí `!tabular` v sobě obsahují vniřní prostředí s matematickými výrazy [2], viz příklad ze zdroje studijního textu k BI-PKM 1.14. Pro jejich korektní zpracování by tak muselo dojít k další transformaci, přestože jde o obsah křehkého prostředí. Obdobně například položka `title` v meta-blocích některých objektů obsahuje matematické výrazy (navíc pro ně ani není použita stejná syntaxe jako pro zkratku vnitřního matematického prostředí `.math`, viz příklad ze zdroje studijního textu k BI-PKM 1.15).

```
!tabular:

    Řecké písmeno  Kapitálka  Česká výslovnost LaTeX
    -------------  ---------  ---------------- -------
    $\alpha$                    alfa            alpha
    $\beta$                     beta            beta
    $\gamma$       $\Gamma$     gama            gamma
    $\delta$       $\Delta$     delta           delta
    $\epsilon$                  epsilon         epsilon
    $\zeta$                     zeta            zeta
    $\eta$                      éta             eta
    $\theta$       $\Theta$     théta           theta
    $\kappa$                    kapa            kappa
    $\lambda$      $\Lambda$    lambda          lambda
```

Výpis kódu 1.14: Křehké prostředí `!tabular` obsahující matematické výrazy ve zdroji studijního textu k BI-PKM [2]

```
.Proof:
  title: 'Důkaz iracionality \(\sqrt{2}\)'
  show: true
```

Výpis kódu 1.15: Meta-blok obsahující matematické výrazy ve zdroji studijního textu k BI-PKM [2]

Dále existující dokumenty obsahují prvky, které jsou zcela validní, ale jejich zobrazení v náhledu nebude jednoduché. Jedná se například o číselné reference do meta-bloku ve vnitřních i vnějších prostředích obsahujících matematické výrazy.

Formát také obsahuje prvky, které nejsou regulární (například správné párování začátků a konců vnitřních prostředí) a také prvky, které nejsou ani bezkontextové (například změna významu na základě úrovně odsazení je kontextuálně závislá).

Tato problematická místa bude potřeba dále zvážit ať už při tvorbě parseru, zobrazení náhledu, implementaci navigačních prvků, realizaci zvýrazňování syntaxe nebo výběru knihoven a dalších nástrojů.

1.3 Porovnání vybraných editorů

Vzhledem k rozsáhlejším požadavkům na rozšíření uživatelského prostředí editoru k přehledné prezentaci logické struktury dokumentů se přímo vybízí využít webové technologie, jako jsou strukturovací jazyk HTML a stylovací jazyk CSS (nebo nějaká z jeho nadstaveb), jež si dle [3] kladou za cíl právě přehlednou prezentaci dokumentů.

Využití těchto technologií v desktopových aplikacích umožňuje framework Electron [4], který v sobě spojuje webový prohlížeč Chromium a JavaScript *runtime* Node.js. Na tomto frameworku je postaveno hned několik populárních open-source editorů zdrojových kódů. Tyto editory dále porovnáme z hlediska jejich vlastností a možností pro jejich rozšíření.

1.3.1 Atom

Atom je editor aktivně vyvíjený společností GitHub. Dle [5] je Atom vytvořen převážně v jazyce JavaScript a jeho starší části také v jazyce CoffeeScript. Tyto jazyky je pak dle [5] také možné použít pro vytváření rozšíření zvaných *packages*. Tato rozšíření mají dle [5] kromě přístupu k rozsáhlému API Atomu také plný přístup k jeho DOM a mohou tak libovolně modifikovat jeho uživatelské rozhraní. Dále dle [5] mohou rozšíření upravovat styly Atomu za pomoci CSS nebo Less. Rozšíření si také podle [5] mohou vzájemně poskytovat verzovaná APIs. Rozšíření je dle [5] možno publikovat v oficiálním repozitáři, odkud si je koncoví uživatelé mohou nainstalovat přímo z rozhraní Atomu. Atom také dle [5] poskytuje integrovaný systém pro testování rozšíření při jejich tvorbě. Architektura Atomu je dle [5] vysoce modulární (mnoho funkcionalit, které jsou v Atomu poskytovány, je odděleno do samostatných *core packages*, které jsou v Atomu předinstalovány).

Rozšiřitelnost a přívětivé uživatelské rozhraní jsou dle [5] hlavními cíli Atomu. Atom dále podle [5] poskytuje rozšiřitelný systém pro zvýrazňování syntaxe, změnu témat syntaxe a témat uživatelského rozhraní, integrovanou podporu práce s nástrojem Git, *core package* pro kontrolu pravopisu, oficiální *package* pro živou kolaborativní editaci textu a mnoho dalších užitečných funkcionalit. Atom má kolem sebe rozsáhlou aktivní komunitu (dle [6] 55 tisíc *stars*, 16 tisíc forků a stovky issues u jeho veřejného GitHub repozitáře).

1.3.2 Visual Studio Code

Visual Studio Code (zkráceně VS Code) je editor aktivně vyvýjený společností Microsoft. Dle [7] je VS Code vytvořen převážně v jazyce TypeScript. TypeScript a JavaScript je pak dle [7] možno přímo použít pro vývoj rozšíření. Rozšíření mají dle [7] přístup k rozsáhlému VS Code API, které také umožňuje různé úpravy uživatelského rozhraní editoru. Narozdíl od Atomu však VS Code dle [7] kvůli dosažení většího výkonu či odezvy neumožňuje rozšířením

plný přístup k jeho DOM. V současnosti tak API [7] nenabízí například možnost přidání horní lišty s nástroji. Rozšíření je také podle [7] možno publikovat v oficiálním „obchodě“, odkud si je opět mohou koncoví uživatelé nainstalovat přímo z rozhraní VS Code. Stejně jako Atom poskytuje také VS Code dle [7] integrovaný systém pro testování rozšíření.

VS Code dále dle [7] poskytuje všechny z výše uvedených funkcionalit Atomu (rozšiřitelné zvýrazňování syntaxe, podpora různých témat, integrovaná podpora Git a další). VS Code má ještě rozsáhlejší komunitu než Atom (dle [8] 116 tisíc *stars*, téměř 19 tisíc forků a tisíce issues u jeho veřejného GitHub repozitáře).

1.3.3 Light Table

Light Table je editor vyvýjený menší společností Kodowa. Jeho vývoj či údržba již není příliš aktivní – i přes občasný [9] *commit* ve veřejném repozitáři vyšla nejaktuálnější verze v roce 2016. Dle [9] je Light Table vytvořen v jazyce ClojureScript a rozšíření (*plugins*) by měla být tvořena také v tomto jazyce. Dokumentace [10] tohoto editoru je však bohužel zejména v oblasti tvorby rozšíření spíše nedostačující (sekce k tvorbě a publikaci rozšíření mají jednotky odstavců). Rozšíření je dle [10] možné publikovat ve formě *pull request* s meta-soubory do veřejného GitHub repozitáře (*pull request* pak musí být autory Light Table schválen a spojen do hlavní větve). Uživatel si pak může rozšíření nainstalovat přímo z rozhraní Light Table. Z dokumentace [10] není zřejmé, nakolik může rozšíření modifikovat uživatelské rozhraní a jestli má přímý přístup k DOM.

Dle [10] Light Table poskytuje okamžitou zpětnou vazbu při psaní kódu a umožňuje do editoru vložit například grafy, hry, vizualizace a další objekty. Light Table má z porovnávaných editorů nejmenší komunitu (dle [9] přes 11 tisíc *stars*, téměř tisíc forků a nižší stovky issues u jeho veřejného GitHub repozitáře).

1.3.4 Závěr porovnání

Všechny tři porovnávané editory jsou dle [6, 8, 9] dostupné na aktuálních verzích všech běžně používaných desktopových operačních systémů (Windows, macOS, Linuxové distribuce) pod permisivními open-source licencemi.

Na základě provedeného porovnání se jeví Atom jako nejvhodnější volba pro další rešerši, analýzu a následnou realizaci, a to díky jeho kvalitní dokumentaci, rozsáhlé komunitě pro řešení případných problémů a zejména díky z porovnávaných editorů největší míře modifikovatelnosti.

1.4 Architektura

Atom *packages* je dle [5] možno tvořit v jazycích JavaScript nebo CoffeeScript. Další možností je samozřejmě napsat *package* v jiném jazyce, který je možno do jednoho z Atomem podporovaných jazyků přeložit.

Dle [5] je potřeba, aby *package* obsahoval v kořenovém adresáři meta-soubor `package.json`. V tomto souboru bude dle [5] uveden například unikátní název *package* a jeho verze, dále zde může být uveden jeho popis, autor a kontakt na něj, odkaz na *issue tracker*, odkaz na repozitář se zdrojem či jiné webové stránky a mnoho dalších informací ve stylu npm *packages*. Dále zde dle [5] mohou být uvedeny různé závislosti z npm, které *package* potřebuje ke svému běhu (nebo pouze k vývoji). Tyto závislosti jsou pak dle [5] automaticky nainstalovány při instalaci samotného *package*. Také zde podle [5] může být uvedena cesta k *main* souboru *package* (pokud není uvedena, použije se výchozí možnost `index.coffee` nebo `index.js`) nebo seznam příkazů, které *package* v Atomu aktivují.

Hlavní modul (*main* soubor) *package* může dle [5] implementovat následující funkce, jejichž volání spravuje sám Atom:

- `activate(state)` – tato funkce je volána, když je *package* aktivován (například po zavolání příkazu, který je v `package.json` definován jako aktivační) a jejím prvním argumentem je stav z posledního volání `serialize`;
- `initialize(state)` – tato funkce je podobná funkci `activate(state)`, je ale volána dříve, ještě před deserializací (při volání `activate(state)` je zajištěno, že je pracovní prostředí Atomu připraveno);
- `serialize` – tato fukce je volána, když je zavíráno okno Atomu a vrací JSON popisující stav `package`, který je později podán funkci `activate(state)` při opětovné aktivaci;
- `deactivate` – tato funkce je volána, když je zavírano okno Atomu nebo když je `package` deaktivován.

Dále dle [5] může hlavní modul exportovat `config` objekt definující položky nastavení *package*, které si může uživatel přizpůsobit (jejich aktuální hodnotu pak lze získat opět pomocí API). Kód *package* má dle [5] vždy k dispozici globální proměnnou `atom`, která poskytuje přístup k API Atomu. Soubory s JavaScript/CoffeeScript kódem patří dle [5] do složky `lib`, soubory s testy pak do složky `spec`.

Dále může *package* dle [5] obsahovat různé složky s meta-soubory, které definují určitou funkcionalitu *package* v rámci Atomu (jedná se o jednodušší alternativu k použití API, které má ale samozřejmě mnohem širší možnosti). Takovými složkami jsou dle [5] například složky `grammars` s *grammar* soubory přidávajícími do Atomu podporu zvýrazňování syntaxe pro nové jazyky,

`keymaps` se soubory definujícími klávesové zkratky příkazů *package* (ty si pak může koncový uživatel upravit v nastaveních), `menus` se soubory přidávajícími položky do hlavního menu a kontextového menu Atomu, `snippets` se soubory definujícími úryvky kódu pro rychlé použití nebo `styles` s CSS či Less soubory upravujícími styly Atomu.

Atom také dle [5] umožňuje poskytnout či použít *services*, verzovaná API, umožňující komunikaci napříč více *packages*. Tohoto by v budoucnu mohlo být využito pro rozdělení *package* do více částí, kde hlavní *package* by například poskytnul *services* pro přidání nových funkcí či tříd pro překlad prvků WooWoo AST do HTML. Takto by mohl vzniknout celý ekosystém *packages* pro podporu WooWoo, kde by například jeden *package* přidal podporu pro zobrazení matematických výrazů, další *package* by přidal podporu pro zobrazení TikZ obrázků a jiný *package*, který by definoval šablonu, by řekl, že závisí na hlavním *package* a dalších konkrétních *packages* přidávajících podporu zobrazení dalších prvků. Tento *package* by také řekl, jak konkrétně šablona, kterou definuje, nazývá a zobrazuje různé typy prostředí, objektů a částí dokumentu. Takto by v budoucnu mohlo být docíleno modulární podpory více WooWoo šablon.

Vydaný *package* si podle [5] uživatelé mohou nainstalovat ze zabudovaného správce *packages* (nebo z terminálu s pomocí nástroje apm, který je také součástí Atomu). Aby bylo možno *package* publikovat, musí dle [5] jeho výstupní JavaScript či CoffeeScript soubory (spolu s dalšími podpůrnými soubory) být zveřejněny v GitHub repozitáři. Následně je dle [5] za pomoci nástroje apm možno publikovat buď aktuální obsah hlavní větve, kdy apm s pomocí GitHub API sám vytvoří nový release/tag (verze je brána z meta-souboru `package.json`), nebo je možno vytvořit release/tag vlastními silami a následně nástroji apm říct, který konkrétní tag má publikovat.

1.5 Zvýrazňování syntaxe

Podle [5] v sobě Atom nabízí zabudované dva způsoby, jak přidat podporu zvýrazňování syntaxe pro nové jazyky.

První, dle [5] novější způsob je založený na nástroji Tree-sitter. Tree-sitter je dle [11] parser generátor, který na základě bezkontextové gramatiky zapsané v jeho vlastním formátu vygeneruje parser, jehož výstupem je derivační strom (CST). Aby pak fungoval efektivně, mělo by dle [11] navíc jít o LR(1) gramatiku. Vygenerovaný parser je dle [11] v jazyku C a lze ho pak použít i v jiných aplikacích. Aby mohl být použit v Atomu, musí být dle [5] zkompilován pro požadované platformy, spolu s tím vygenerovány *bindings* pro Node.js a tento celý výstup publikován jako knihovna na npm. Tuto knihovnu následně dle [5] využije vytvářený *package* přidávající do Atomu podporu nového jazyka s pomocí *grammar* souboru v CSON nebo JSON formátu, ve kterém dále specifikuje, které vrcholy CST připadají kterému *grammar scope* (*grammar*

scopes v podstatě určují CSS třídu, která bude kódu, který odpovídá danému vrcholu, přiřazena). Těmto *grammar scopes* pak dle [5] s pomocí odpovídajících CSS tříd přiřazují styly různé další *packages* přidávající do Atomu syntaktická témata.

Výhodou tohoto přístupu je, že podle [5] funguje rychleji a přesněji (parser je založený na bezkontextové gramatice namísto rozšířených regulárních výrazů), než dále zmíněná alternativa. Nevýhodou je, že i dle Tree-sitter dokumentace [11] je náročný na realizaci. Navíc vzhledem ke kontextově závislým prvkům WooWoo formátu by takto vzniklý parser stejně nebyl schopen WooWoo parsovat zcela korektně a nemohl by tak být sám o sobě využit například pro překlad do HTML pro náhled dokumentu.

Druhým způsobem je dle [5] vytvoření gramatiky ve formátu založeném na TextMate gramatikách. Jedná se o poměrně jednoduchý systém, který dle [5] využívá (rozšířené) regulární výrazy. Postup vytvoření *grammar* souboru je zde dle [5] podobný, jako v předchozím případě, pouze namísto vrcholů CST přiřazujeme *grammar scopes* vzorům popsaným (rozšířenými) regulárními výrazy (a samozřejmě zde odpadá definice použité parser knihovny). Vnitřně pak dle [5] Atom využívá engine (rozšířených) regulárních výrazů Oniguruma, konkrétní podporované konstrukty v (rozšířených) regulárních výrazech jsou tedy dané právě tímto enginem.

Výhodou tohoto přístupu je jednoduchost implementace. Další výhodou je, že podobný způsob zvýrazňování syntaxe využívají další populární editory (např. TextMate, po kterém je tento způsob zvýrazňování syntaxe pojmenovaný, dále třeba VS Code [7] nebo Sublime Text [12]) a vytvořená gramatika by se tak dala s drobnými úpravami použít i v nich. Nevýhodou tohoto přístupu je již zmiňovaná nižší rychlost a přesnost. Dále je nutno zmínit, že Tree-sitter gramatiky jsou aktuálně dle [5] preferovány a Atom na ně chce postupně přejít. Nicméně vzhledem k velkému množství existujících TextMate gramatik není pravděpodobné, že by k nějakému úplnému ukončení jejich podpory došlo v dohledné době.

Atom dále dle [5] umožňuje namísto JSON/CSON souboru popsat gramatiku přímo v rámci API a dynamicky ji měnit za běhu. Toho využivá například *package* tasks [13]. V našem případě by toto mohlo být užitečné například pro přesnější zvýrazňování syntaxe v závislosti na použité šabloně. Dále by tento přístup mohl zmenšit opakování kódu (za předpokladu že by obdobné regulární výrazy byly využity také pro parsování) a lepší testovatelnost.

1.6 Parsování

Pro další práci s WooWoo dokumentem (ať už jde o překlad do HTML, vytvoření přehledu určitých prvků a další) je vhodné z dokumentu vygenerovat abstraktní syntaktický strom (AST) – stromovou strukturu, která popisuje skutečný sémantický význam zdroje (na rozdíl od derivačního stromu, CST,

který zachycuje spíše přesnou syntaktickou strukturu). Pro vygenerování AST bude nutné vytvořit parser. Toho je možné docílit mnoha způsoby.

Jedním způsobem je využít parser generátorů – programů, kterým je dodán popis gramatiky v nějakém požadovaném formátu, na jehož základě samy vygenerují výsledný parser. Většina parser generátorů však přijímá nejvýše bezkontextové gramatiky. Pro případnou podporu kontextově závislých prvků jazyka je možno vygenerovaný parser vlastními silami upravit, což však vyžaduje pokročilé znalosti v oblasti tvorby parserů.

Pro generování AST proto vytvoříme vlastní parser, který bude pro zjednodušení implementace vnitřně využívat (rozšířených) regulárních výrazů pro parsování těch částí WooWoo formátu, kde to bude možné. Pro parsování neregulárních bezkontextových prvků WooWoo formátu (například správné určení začínajících a ukončovacích tokenů vnitřních prostředí) nebo kontextově závislých prvků (například změna významu na základě úrovně odsazení) bude potřeba použití (rozšířených) regulárních výrazů doplnit o vlastní funkcionalitu, jako různá počítadla, zásobníky apod.

AST obecně nemají nějaký standardně používaný formát (viz [14]), liší se parser od parseru, ale v mnoha existujících AST (pro jiné jazyky) si můžeme všimnout častých prvků, které bude vhodné v našem AST také zahrnout. Jedná se například o pozici (řádek, sloupec a odsazení od začátku souboru) začátku a konce vrcholu stromu. Tato pozice umožní například přesnou navigaci mezi částmi dokumentu, synchronní posuv náhledu se zdrojem a mnoho dalších funkcionalit.

1.7 Zobrazení náhledu

V analýze požadavků 1.1 se nachází řada funkčních požadavků na živý náhled dokumentu. Jedním požadavkem je **F2**, dalším požadavkem pak **F3**.

Pro zobrazení náhledu bude AST dokumentu získaný parsováním přeložen do HTML, které bude následně zobrazeno v novém podoknu Atomu. Toto HTML je možno libovolně stylovat díky CSS (případně jeho nadstavby Less, jejíž podporu má v sobě Atom podle [5] integrovanou). Je však dobré mít na paměti, že uživatel může chtít použít různá témata uživatelského prostředí a je tak lepší nechat co nejvíce stylování právě na nich a v případě nutnosti úpravy stylů dělat takové volby, které budou čitelné napříč populárními tématy prostředí (například tedy není dobré nastavit barvu textu napevno na bílou či naopak černou). Dále je s tímto HTML možno dál libovolně pracovat za pomoci JavaScriptu a například tak náhled doplnit o interaktivní prvky.

Živá aktualizace náhledu bude zajištěna díky API Atomu, které dle [5] umožňuje reagovat na různé změny v editoru. Při detekci změny pak bude nový obsah opět přeložen na HTML a zobrazen uživateli. Implementace nějakého chytřejšího systému, kde by se aktualizoval pouze pozměněný obsah, by zřejmě byla složitější a pravděpodobně ani ne zcela nutná. Je však namístě

nějakým způsobem cachovat objekty náročnější na překlad či vykreslení, aby bylo dosaženo přiměřené rychlosti odezvy.

1.7.1 Zobrazování matematických výrazů

Dále je kladen funkční požadavek **F4**. Pro zapisování matematických výrazů aktuálně v praxi používaná WooWoo šablona FIT Template používá LaTeX matematická prostředí. Snažit se přesně zobrazit tyto výrazy není vůbec jednoduché a je tak namístě použít k jejich zobrazování buď nativní instalaci prostředí LaTeX, nebo jednu z mnohých knihoven pro zobrazování matematických výrazů na webu.

Výhodou použití nativní instalace je kvalita výstupu a samozřejmě maximální možná podpora LaTeX maker. Nevýhodou použití nativní instalace je nedostatečná rychlost. Generování obrázku z jediného jednoduchého výrazu trvá řádově vteřiny (vyzkoušeno na vlastním stroji s procesorem Intel Core i5 7300HQ, 8 GB RAM, operačním systémem Windows 10). Vzhledem k velmi vysokému množství matematických výrazů ve WooWoo zdrojích existujících studijních textů (řádově vyšší desítky až stovky výrazů na kapitolu, viz [2]) by tak i při spuštění více podprocesů nemohlo být docíleno akceptovatelné odezvy.

Druhou možností je již zmíněné využití knihoven, jejichž výhodou je oproti nativní instalaci právě rychlost. Další výhodou těchto knihoven je, že nevznikají externí závislosti (jejichž instalaci si musí koncový uživatel pohlídat). Nevýhodou knihoven je nižší pokrytí podporovaných LaTeX maker, případně nižší kvalita sazby (která se však mezi knihovnami výrazně liší také v závislosti na jejich nastavení). Dvěma populárními knihovnami jsou MathJax a KaTeX, které mezi sebou dále blíže porovnáme. U knihovny MathJax navíc je vhodné oddělit od sebe pro účely porovnání MathJax 2.x a MathJax 3.x, protože novější velká verze je dle [15] od základu přepsána a funguje v mnohých ohledech jinak.

MathJax dle [16] poskytuje kvalitní sazbu matematických výrazů, vstup ve formátech MathML, (La)TeX a ASCIIMath a výstup ve formátech HTML + CSS, SVG, nebo MathML ve verzi 3.x (a ve verzi 2.x dle [17] navíc výstupy PreviewHTML a CommonHTML). MathJax v sobě dále dle [16] má zabudované nástroje pro asistenci nevidomým. Dle [15] také podporuje možnost definice vlastních maker. Verze 3.x je dle [15] rychlejší než verze 2.x o zhruba 60 až 80 procent, ale neposkytuje zatím tak širokou podporu maker jako starší verze. MathJax je dle [15] dostupný (mimo jiné) z npm pod permisivní open-source licencí. HTML výstupy [2] z WooWoo zdroje existujících studijních materiálů využívají pro zobrazování matematických výrazů právě MathJax 2.7.

KaTeX je dle [18] nejrychlejší knihovna pro zobrazování matematických výrazů na webu, která poskytuje kvalitní sazbu jejíž rozložení je založeno na rozložení sazby původního TeX. Dle [19] také podporuje možnost definování

vlastních maker. Za vývojem KaTeX stojí podle [18] Khan Academy a KaTeX je dostupný (mimo jiné) z npm pod permisivní open-source licencí. Podle [20] KaTeX nenabízí tak širokou podporu matematických maker jako MathJax.

Vzhledem k nejširší podpoře matematických maker mezi porovnávanými knihovnami bude pro realizaci nejvhodnější použít MathJax 2.x i na úkor menší rychlosti za předpokladu, že bude postačující pro zobrazení živého náhledu. Navíc tím bude docíleno parity s existujícími HTML výstupy WooWoo co do podpory zobrazování matematických výrazů. Z dostupných výstupů MathJax 2.x bude nejlepší použít SVG, protože je dle [17] druhý nejrychlejší a oproti nejrychlejšímu výstupu PreviewHTML poskytuje dostatečně kvalitní úroveň sazby.

1.7.2 Zobrazování grafických objektů

Posledním funkčním požadavkem z oblasti zobrazení náhledu je **F5**. FIT Template šablona a studijní texty, které ji využívají, dle [1, 2] obsahují obrázky popsané LaTeX kódem využívajícím *package* TikZ.

Pro podporu zobrazení těchto objektů se opět nabízí dvě hlavní možnosti – použití nativní instalace TeX distribuce, nebo použití knihovny. Nativní instalace bude pravděpodobně mít opět rychlostní nevýhodu (a nevýhodu externí závislosti).

Jediná knihovna pro zobrazování TikZ obrázků na webu, kterou se během rešerše podařilo objevit, je knihovna TikZJax. Tato knihovna dle [21] využívá originální zdroj TeX zkompilovaný do WebAssembly a její výstup by tak teoreticky měl být ekvivalentní výstupu nativní instalace. Bohužel je ale aktuálně dle [21] TikZJax distribuován pouze přes CDN, nikoliv například přes npm a při tvorbě produkčního balíčku v sobě zahrnuje soubory písem z LaTeX editoru BaKoMa. Tato písma však dle [22] neposkytují licenci umožňující volné šíření a TikZJax tyto soubory nezahrnuje [21] ve svém repozitáři.

Vzhledem k nefunkčnímu požadavku **N2** a vzhledem k tomu, že výstupy některých obrázků obsažených v existujících studijních textech [2] za použití lokálního sestavení TikZJax nedosahují bez potřebných souborů písem dostačující kvality, bude při realizaci vhodnější použít nativní instalaci TeX distribuce.

1.8 Navigace

V analýze požadavků 1.1 se nachází řada funkčních požadavků na navigaci v dokumentu. Tyto funkční požadavky lze rozdělit na dvě skupiny – požadavky na navigaci v částech dokumentu a požadavky na navigaci ve značkách v dokumentu.

Navigaci v částech dokumentu by bylo možné realizovat jako navigační okno ve stylu těch, která nabízí například prohlížeče PDF dokumentů nebo textové procesory (např. Microsoft Word [23]). Uživateli by se tak zobrazil

víceúrovňový seznam, kde každá úroveň obsahuje všechny části dokumentu dané úrovně seřazeny za sebou tak, jak se v dokumentu vyskytují. Nad seznamem by bylo umístěno vyhledávací pole, s jehož pomocí by uživatel mohl vyfiltrovat pouze relevantní části dokumentu. Vytvořit takový seznam půjde díky AST vzniklému parsováním. Pro zobrazení každé položky seznamu, tedy názvu části dokumentu, by mohla být zavolána stejná funkce, která jej zobrazí v náhledu dokumentu. Tím bude postaráno i o zobrazení případných vnitřních prostředí v názvu dokumentu. Při kliknutí na položku seznamu by pak došlo ke skoku na odpovídající řádek v editoru (řádek, kde začíná část dokumentu, také umožní určit AST).

Pro zobrazení seznamu značek obsažených v dokumentu s možností fuzzy vyhledávání, skokem na vybranou značku ze seznamu do dokumentu nebo z reference na značku se nabízí využít *core package* Atomu Symbols View. Tento *package* dle [24] umožňuje takovou navigaci mezi symboly dokumentu, které je nutné popsat v *tags* souboru ve formátu kompatibilním s nástrojem Ctags. Ctags z pochopitelných důvodů neumí tento *tags* soubor vygenerovat pro WooWoo dokumenty. Toto by tedy bylo nutné naimplementovat v našem rozšíření, nebo případně využít jednoho ze způsobů jak rozšířit Ctags o vlastní parser (viz [25]).

Implementace generování *tags* souboru se zdá jako relativně jednoduché řešení, protože Ctags formáty mají podle [25] jednoduchou syntaxi. Navíc všechny informace, které je v *tags* souboru nutno uvést, získáme z AST vytvořeného při parsování, které budeme implementovat už jen kvůli překladu WooWoo do HTML pro zobrazení náhledu. Bohužel Symbols View dle [26] při zobrazení symbolů v aktuálně otevřeném souboru pro jejich získání sám volá nástroj Ctags, namísto toho, aby se je pokusil získat z již existujícího *tags* souboru (toto se liší od chování při zobrazení symbolů v celém projektu, kdy Symbols View nejprve zkouší číst existující *tags* soubor) a chybí tak způsob, jak mu náš vlastní soubor předat. Vzhledem k velkému množství značek ve WooWoo zdrojích existujících studijních textů je zrovna možnost zobrazení symbolů pouze z aktuálního souboru důležitá.

Existuje také *package* atom-ctags, fork Symbols View *package*, který Symbols View v mnoha směrech rozšířil a pozměnil [27]. Nabízí se tedy otázka, zdali by v tomto směru nemohl fungovat lépe. Tento fork ale obsahuje dle jeho *issue tracker* [27] nemalé množství problematických chyb a jeho vývoj či údržba je dle [27] neaktivní od roku 2017.

Dalšími možnostmi je tedy vytvořit nový fork Symbols View *package* a upravit jej dle vlastních potřeb, nebo rozšířit Ctags o vlastní parser (což by ale možná také nakonec vyžadovalo drobné úpravy Symbols View *package*). Zde už se však zdá jednodušší vytvořit si vlastní přehled značek v otevřeném souboru, kde implementace bude převážně stejná či hodně podobná jako implementace zobrazení přehledu částí dokumentu.

Alternativně by ještě bylo možné vytvořit *language server* pro komunikaci s Atomem přes Language Server Protocol (viz [28]), kterou na straně Atomu

umožňuje knihovna Atom Language Server Protocol Client [29]. Tento přístup by dle [29] umožnil mimo jiné právě navigaci z reference značky v dokumentu na její definici. Navíc by takto implementovanou funkcionalitu podle [28] bylo možno využít i v jiných editorech, které podporují Language Server Protocol (v současnosti dle [28] například VS Code, Visual Studio nebo Brackets, ve formě rozšíření také například Eclipse, Sublime Text, emacs nebo vim). Tento přístup by ale dle [29] sám o sobě neposkytl seznam všech značek a možnost vyhledávání v něm. Dále by pravděpodobně vyžadoval komplexnější implementaci.

1.9 Testování

Dle [5] v sobě Atom poskytuje integrovaný systém pro spouštění testů, takzvaný *test runner*, postavený na testovacím frameworku Jasmine. Dále [5] uvádí, že všechny testy musí být ve složce `spec` v kořenové složce *package* a musí mít ve jméně příponu `-spec` (tedy například `main-spec.js`). Následně je podle [5] možné všechny testy spustit z terminálu příkazem `atom --test spec` (případně lze namísto celé složky s testy použít seznam konkrétních souborů s testy), nebo přímo z Atomu s pomocí příkazu `window:run-package--specs`, za předpokladu, že je jako aktuální projekt otevřen právě testovaný *package*.

Tento Atomem poskytovaný *test runner* nicméně dle [5] vnitřně využívá Jasmine ve verzi 1.3. V [30] najdeme, že se jedná o poněkud starší verzi z roku 2013. Chceme-li použít novější verzi Jasmine, nebo dokonce úplně jiný testovací framework, můžeme podle [5] využít možnosti vytvoření vlastního *test runner*. Takto vytvořené *test runners* pak navíc dle [5] jde jednoduše poskytnout ostatním vývojářům pro využití v jejich *packages*, které si konkrétní *test runner* pouze stáhnou jako závislost z npm a v `package.json` řeknou Atomu, že chtějí využít onen konkrétní *test runner*.

1.9.1 Výběr komunitního *test runner*

Protože vytvoření vlastního *test runner* nemusí být zrovna jednoduché, není to cílem této práce a zároveň už existuje více *test runners*, mezi kterými si můžeme vybrat, porovnáme několik těchto komunitních *test runners* a pro realizaci vybereme jeden z nich.

Použití testovacího frameworku Jest umožňuje *test runner* atom-test-runner-jest [31]. Tento *test runner* se ale nezdá být aktivně udržován (poslední commit je dle [31] z června 2018), *issue tracker* obsahuje dle [31] dva otevřené issues z celkového počtu tří issues, všechny také z roku 2018. Komunita kolem tohoto *test runner* se zdá malá (jednotky až nízké desítky týdenních stažení během posledního roku z npm [32]).

Použití testovacího frameworku Mocha umožňuje *test runner* atom-mocha-test-runner [33]. Tento *test runner* je na tom z hlediska aktivity údržby

lépe než atom-test-runner-jest. Poslední commit je dle [33] z prosince 2020 (je však pouze o poloautomatický update závislostí; poslední commit bez těchto prostých aktualizací je z června 2019), *issue tracker* dle [33] neobsahuje žádný otevřený issue z celkem dvou issues (z nichž poslední je z prosince 2020). Komunita kolem tohoto *test runner* se zdá být větší než u atom-test-runner-jest (vyšší stovky až přes tisíc týdenních stažení během posledního roku z npm [32]). Tento *test runner* navíc podle [33] spravují přímo vývojáři Atomu.

Použití aktuální verze frameworku Jasmine (což je v době psaní verze 3.7) umožňuje *test runner* atom-jasmine3-test-runner [34]. Tento *test runner* se na tom zdá být nejlépe jak z hlediska aktivní údržby, tak z hlediska velikosti komunity kolem. Nové commity dle [34] přibývají každý týden, *issue tracker* dle [34] neobsahuje žádný otevřený issue z celkových 12 issues (z nichž poslední je z ledna tohoto roku), počty týdenních stažení z npm se dle [32] pohybují kolem vyšších stovek až nižších jednotek tisíc během posledního roku. Autor tohoto *test runner* je navíc velmi aktivní i v dalších komunitních projektech kolem Atomu (například také spravuje [35] GitHub *action* pro použití Atomu v nástroji pro kontinuální integraci GitHub Actions) a Jasmine. Tento *test runner* je podle [34] také využíván pro testování například v *package* autocomplete-plus [36], což je *core package* Atomu, nebo také v *package* Hydrogen [37], oblíbeném prostředí pro práci s Jupyter notebooky.

Podobných *test runners* existuje mnohem více, ale převažuje mezi nimi problém neaktivní údržby, malé komunity, případně i otevřených issues s problematickými chybami. Z výše popsaných *test runners* se jeví atom-jasmine3--test-runner jako nejlepší volba, díky aktivní údržbě a větší komunitě, která tento *test runner* využívá.

Kapitola 2

Realizace

Pro realizaci rozšíření Atomu byl vybrán jazyk TypeScript. TypeScript je dle [38] open-source nadstavba jazyka JavaScript, která jej rozšiřuje o statickou typovou kontrolu. Díky kompilaci do jazyka JavaScript je pak výstup možné použít v Atomu.

Kód rozšíření je doplněn o dokumentační komentáře. Dále byl využit nástroj Prettier pro formátování kódu a nástroj ESLint pro statickou analýzu.

2.1 Zvýrazňování syntaxe

Funkční požadavek **F1** byl na základě provedené rešerše 1.5 realizován vytvořením TextMate gramatiky. Tato gramatika je tvořena *grammar* souborem ve složce `grammars`. Dynamické tvorby gramatiky za pomoci API prozatím nebylo využito, její použití zůstává možným vylepšením do budoucna. Zvýrazňování syntaxe WooWoo zdroje studijního textu k BI-PKM lze vidět na obrázku 2.1.

Podporována je také aktuální zkrácená forma vnitřních prostředí se zavináčem i starší zkrácená forma s tečkou. Jediným prvkem WooWoo formátu, který není podporován, je direktiva `.include`.

Pro zajištění podpory napříč různými populárními tématy syntaxe byla použita pouze obvykle užívaná jména *scopes*, jejichž seznam uvádí [39].

Vzhledem k omezeným vyjadřovacím schopnostem TextMate gramatik (které jsou založeny na rozšířených regulárních výrazech) jsou některé konstrukty zvýrazňovány i tehdy, když nejsou zcela syntakticky správné. Například zvýrazňování syntaxe verbózní formy vnitřních prostředí je řešeno zvýrazňováním ukončující části prostředí (horní dvojitá uvozovka následovaná tečkou a typem prostředí). Na obrázku 2.2 je ilustrován případ, kdy je zvýrazněna syntaxe nevalidního vnitřního prostředí. Další chyby při zvýrazňování syntaxe jsou způsobeny nemožností určit kontext. Jedna z takových chyb je ilustro-

```
.Question:

  Má prázdná množina maximum a minimum?

  .solution:

    Nemá. Aby tato otázka měla naději na úspěch, musela by tato množina
    mít nějaký prvek.


.Section Výroky a logické spojky
  label: sec-vyroky-logicke-spojky

Matematická tvrzení je výhodné zapisovat ve zkrácené formě pomocí symboliky
predikátové logiky. Podrobně bude tato oblast matematiky probrána v předmětu
Matematická logika ("BI-MLO".1). Díky využití tohoto přístupu vynikne logická
struktura tvrzení, která by jinak mohla čtenáři zůstat skryta za větami
přirozeného (v našem případě českého) jazyka. Na tomto místě pouze stručně
shrneme základy, které jsou čtenáři již jistě známy.
  1:
    link: 'https://courses.fit.cvut.cz/BI-MLO'
```

Obrázek 2.1: Zvýrazňování syntaxe WooWoo zdroje studijního textu k BI-PKM [2]

```
1  text text
2  text text".quoted
3  text text
```

Obrázek 2.2: Chyba zvýrazňování syntaxe WooWoo vnitřních prostředí

vána na obrázku 2.3, kde je chybně zvýrazněno „vnitřní prostředí“ v křehkém vnějším prostředí.

2.2 Parsování

Pro parsování dokumentu do AST byl vytvořen vlastní parser. Tvorba parseru se ukázala jako nejnáročnější část realizace. Pro zjednodušení implementace v sobě parser využívá (rozšířené) regulární výrazy kde je to možno. Jejich použití pak doplňuje například použitím počítadla pro správné ukončování vnitřních prostředí. Dále je používán zásobník, který umožnuje sledovat aktuální *scope*.

```
!codeblock:
  language: html

  text text
  text "text".quoted
  text text
```

Obrázek 2.3: Chybějící kontext při zvýrazňování WooWoo syntaxe

Pro parsování meta-bloků (odsazených YAML bloků) byla využita knihovna yaml. Jedná se o aktivně udržovanou, populární knihovnu (4 až 12 milionů týdenních stažení z npm [32] během posledního roku) s permisivní open-source licencí, která také poskytuje TypeScript *types*.

Parser korektně parsuje každý jednostránkový validní WooWoo dokument. Nechává ale oproti WooWoo specifikaci větší volnost v prázdných řádcích mezi bloky, objekty a částmi dokumentu. Dále parser podporuje vnořená vnější prostředí, což opět WooWoo specifikace [1] nedovoluje (narozdíl od objektů). Parser tedy parsuje i některé dokumenty, které nejsou zcela validní dle WooWoo specifikace. Direktivy `.include` aktuálně podporovány nejsou a parser je považuje za normální text bez speciálního významu.

Parser podporuje jak novou zkrácenou formu vnitřního prostředí se zavináčem, tak starší zkrácenou formu s tečkou. Je tomu z důvodu použití starší zkrácené formy ve zdrojích existujících studijních textů.

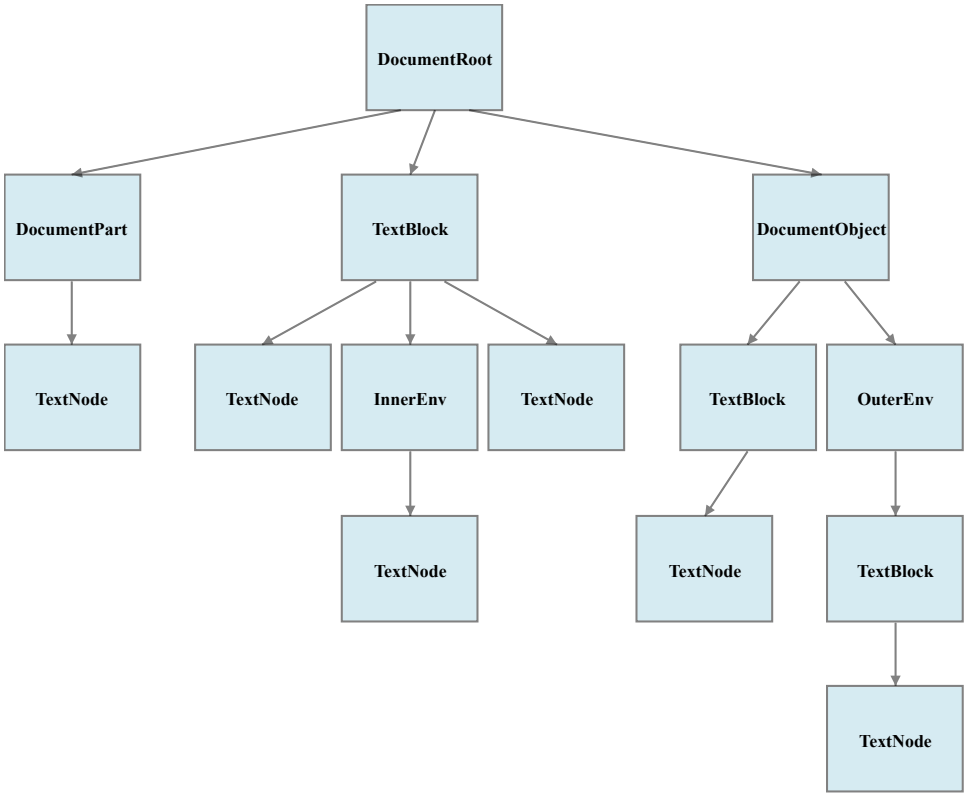
Každý vrchol výsledného AST dědí z abstraktní třídy `ASTNode`. Tato abstraktní třída deklaruje druh vrcholu, jeho rodiče (může být prázdný, což znamená vrchol stromu), seznam jeho dětí (může být prázdný, což znamená list stromu), jestli je křehký, počáteční a koncovou pozici ve zdroji (řádek, sloupec a odsazení od počátku souboru) a jeho meta-data. `ASTNode` také definuje funkce pro porovnání dvou vrcholů nebo výpis vrcholu do JSON.

Dále mohou vrcholy implementovat interface `ValueASTNode` (který deklaruje hodnotu vrcholu) a rozšiřovat abstraktní třídu `VariableASTNode` (která rozšiřuje abstraktní třídu `ASTNode` a deklaruje variantu vrcholu). Variantou vrcholu je myšlen typ částí dokumentu, objektů a vnějších či vnitřních prostředí (slovo `variant` bylo použito namísto slova `type`, protože `type` je dle [38] v jazyce TypeScript klíčové slovo).

Na obrázku 2.4 je ilustrován (bez detailů jako pozice vrcholu nebo typ/varianta) AST vzniklý parsováním zdroje 2.1.

2.3 Zobrazení náhledu

Náhled dokumentu je realizován v samostatném podoknu Atomu, které si uživatel může v pracovním prostředí Atomu zobrazit s pomocí příkazu `wootom:togglePreview`, klávesové zkratky (ve výchozím nastavení Alt + J),



Obrázek 2.4: AST vzniklý parsováním WooWoo zdroje 2.1

```
.Chapter Jméno kapitoly
  label: chap-1

text text "text text".emphasize text


.Question:

  text text "text text".emphasize text?

  .solution:

    Řešení
```

Výpis kódu 2.1: WooWoo zdroj, jehož AST je ilustrován na obrázku 2.4

```
interface Renderer {
    kind?: WooElementKind;
    abstractVariant?: string;

    render<T extends ASTNode>(
        renderingManager: RenderingManager,
        astNode: T,
    ): Node;
}
```

Výpis kódu 2.2: Interface `Renderer`

položky „Wootom: Toggle Preview“ v kontextovém menu nebo podpoložky „Toggle Preview“ položky „Wootom“ v hlavním menu Atomu.

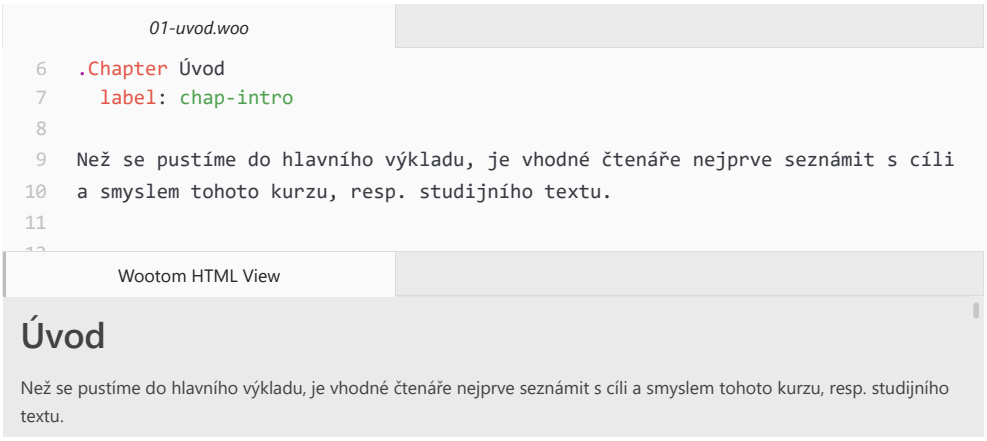
Po zobrazení podokna náhledu (nejprve se otevře napravo podokna editoru aktuálně otevřeného souboru, uživatel se poté může podokno přemístit) je aktuálně otevřený soubor rozparsován do AST.

Poté co je dokument rozparsován, je vzniklý AST reprezentující obsah WooWoo dokumentu podán metodě `RenderingManager#render`. Tato metoda přijme kořen AST, získá pro něj příslušnou implementaci `Renderer` (viz výpis kódu 2.2) z registru `RendererRegistry`, zavolá metodu `Renderer#render` na tento kořen a její výsledek vrátí. Metoda `Renderer#render` pak může teoreticky s přijmutým vrcholem dělat cokoliv, ale v typickém případě vrchol stromu nějakým způsobem tranformuje na HTML, na všechny jeho děti opět zavolá `RenderingManager#render` a výsledky (HTML vrcholy) těchto volání přidá jako obsah do svého HTML. Takto se renderování postupně zanořuje hlouběji a hlouběji až projde celý AST. Pokud je navíc některý z vrcholů, které `RenderingManager#render` zpracovává, jedním z druhů prvků WooWoo dokumentu, které mají variantu, před hledáním příslušného `Renderer` v `RendererRegistry` je navíc získána abstraktní varianta z `VariantRegistry`. Pokud abstraktní varianta nebyla nalezena, je použit výchozí `Renderer`.

Takto je z AST získáno HTML reprezentující logickou strukturu WooWoo dokumentu. Toto HTML je následně podáno metodě `HTMLViewModel#render`, která jej zobrazí uživateli v podoknu pro náhled.

Na obrázku 2.5 je pro ilustraci zobrazen náhled dokumentu s definicí části (zde konkrétně kapitoly), jejím meta-blokem a následujícím blokem textu (zdroj dokumentu vzatý ze studijního textu BI-PKM).

Náhled je živě aktualizován při modifikaci zdrojového souboru. Bylo implementováno jednoduché zobrazování většiny prvků FIT Template šablony dle stylů existujících výstupů. Zobrazování některých prvků ale zatím podporováno není. Z časových důvodů nebylo implementováno zobrazení vnějších prostředí `.enumerate` a `.itemize` a vnitřního prostředí `.item`. Dále nebylo



Obrázek 2.5: Náhled části dokumentu ve zdroji studijního textu k BI-PKM [2]

implementováno zobrazování vnějšího prostředí `.image` a vnitřního prostředí `.todo`, a to z důvodu jejich absence ve zdrojích existujících studijních textů. Všechny ostatní prvky jsou (alespoň částečně) zobrazovány.

Tímto je realizován funkční požadavek **F2** a **F3**. V budoucnu by mohl být náhled vylepšen například o zobrazení celého dokumentu (nikoliv pouze aktuálně otevřeného souboru) nebo o synchronní posuv náhledu a zdrojového souboru (což by mělo být možné díky přesným pozicím vrcholů AST).

2.3.1 Zobrazování matematických výrazů

Zobrazování matematických výrazů je na základě provedené rešerše 1.7.1 řešeno za pomoci knihovny MathJax 2.x s konfigurací SVG výstupu. Uživateli je dále umožněno nadefinovat v nastaveních rozšíření vlastní makra, která jsou předána konfiguraci knihovny MathJax.

Na obrázku 2.6 je pro ukázku zobrazen náhled vnitřních matematických prostředí v textu, jež jsou doplněna o vnější matematické prostředí (blokovou matematiku).

Protože je sazba matematických výrazů výpočetně náročná, jsou výstupní SVG jsou cachována v paměti (do zavření editoru) a je-li stejný výraz použit znovu (při obnovení náhledu nebo třeba v jiném souboru), MathJax už volán není a výsledné SVG je zobrazeno mnohem rychleji (zdánlivě okamžitě).

Byl tak realizován funkční požadavek **F4**.

2.3.2 Zobrazování grafických objektů

Zobrazování TikZ obrázků je na základě provedené rešerše 1.7.2 řešeno závislostí na nativní instalaci TEX distribuce. Obdobně jako uživatel může v nastaveních nadefinovat vlastní makra pro MathJax, tak i zde je umožněno do

Vezměme nejprve velmi jednoduchý, konkrétní a čtenáři jistě známý příklad přímky. Když nám někdo řekne, že máme prímku zadanou rovnicí $y = 2x + 1$, pak tím vlastně říká, že v naší rovině ho zajímá *množina bodů*

$$\{(x, y) \in \mathbb{R}^2 \mid y = 2x + 1\}.$$

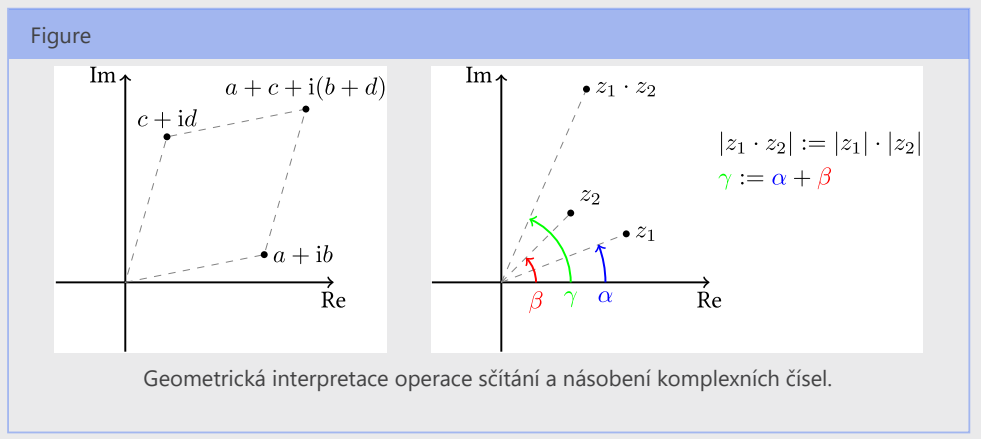
Slovně, množina všech bodů se souřadnicemi $(x, y) \in \mathbb{R}^2$, které splňují rovnost $y = 2x + 1$. Bod o souřadnicích $(1, 1)$ do této množiny nepatří (neleží na této přímce), protože po dosazení do naší podmínky dostaneme rovnost $1 = 3$, která neplatí. Naopak bod $(1, 3)$ do naší množiny patří (na této přímce leží), protože po dosazení do dostaneme rovnost $3 = 3$, která je pravdivá.

Obrázek 2.6: Matematické výrazy v náhledu studijního textu k BI-PKM [2]

preambule LaTeX dokumentu, který je použit pro vygenerování obrázku, vložit vlastní obsah (nejen vlastní makra, ale například i použití dalších LaTeX *packages*).

Při překladu vrcholu AST, který reprezentuje vnější prostředí `!tikz`, je nejprve zobrazeno HTML se (částečně skrytým) zdrojem TikZ. Při tomto překladu je zároveň zavolána asynchronní funkce, která nejprve zkontroluje, zdali se zdroj obrázku nenachází v cache. Pokud ano, je vrácen obsah cache (kterým je výsledné SVG) a ten je zobrazen uživateli. Pokud ne, je TikZ zdroj spolu s preambulí (výchozí, nebo uživatelem definovanou) zapsán do dočasného souboru, na který je zavolán příkaz `latex`, jehož výsledkem je DVI soubor (jehož generování je rychlejší, než PDF). Pro vygenerování SVG z DVI je pak využit příkaz `dvisvgm`. Nakonec je SVG výstup přečten, přidán do cache, jsou uklizeny dočasné soubory a obrázek je zobrazen uživateli. Pokud při procesu nastala chyba, je uživateli zobrazen namísto obrázku obsah LaTeX logu (log soubor je pak také odstraněn). Cache funguje stejným způsobem jako cache pro MathJax výstup (je pouze v paměti, po zavření editoru je ztracena).

Protože vygenerované SVG má fixní barvy výplně, je pro podporu různých témat uživatelského rozhraní uživateli umožněno vybrat v nastaveních jeden ze tří režimů zobrazování těchto SVG výstupů. Výchozím režimem je zobrazení obrázků s bílým pozadím, dalším režimem pak inverze barev SVG a posledním režimem je zobrazení SVG bez jakýchkoliv změn (není přidáno pozadí ani pozměněny barvy).



Obrázek 2.7: TikZ obrázky v náhledu studijního textu k BI-PKM [2]

Tímto byl realizován funkční požadavek **F5**. Ukázka náhledu TikZ obrázku se nachází na obrázku 2.7.

2.4 Navigace

Navigace v dokumentu je podobně jako náhled realizována v podobě navigačního podokna, které si uživatel může v pracovním prostředí Atomu zobrazit s pomocí příkazu `wootom:toggleNavigation`, klávesové zkratky (ve výchozím nastavení Alt + N), položky „Wootom: Toggle Navigation“ v kontextovém menu nebo podpoložky „Toggle Navigation“ položky „Wootom“ v hlavním menu Atomu.

Po zobrazení navigačního podokna (které se nejprve zobrazí napravo podokna editoru aktuálně otevřeného souboru; uživatel si jej případně může přemístit) je aktuálně otevřený soubor rozparsován do AST. Z AST je pak získán seznam všech vrcholů, které reprezentují části dokumentu a seznam všech vrcholů, které mají v meta-datech neprázdnou položku `label`.

Ze seznamu vrcholů částí dokumentu je pak vytvořena jednoduchá stromová struktura reflektující úrovně konkrétních částí (toto je potřeba provést, protože vytvořený AST části nevnořuje podle úrovně) z té je vytvořen číslovaný víceúrovňový HTML seznam obsahující nadpisy daných částí. Nadpisy jsou přeloženy do HTML s pomocí volání `RenderingManager#render`, čímž je zajištěno i zobrazení případných vnitřních prostředí. Seznam vrcholů AST se značkou je abecedně seřazen a poté je z něj vytvořen jednoduchý nečíslovaný HTML seznam, obsahující název značky, druh vrcholu (případě také jeho variantu) a jeho počáteční řádek. Všem položkám obou HTML seznamů je přidán

event listener pro událost *click*, který při kliknutí na položku seznamu skočí v editoru aktuálně otevřeného souboru na pozici začátku daného prvku.

Tyto HTML seznamy jsou vloženy do navigačního podokna a pro přehlednost uvedeny nadpisy. Mezi nadpis a seznam je pak vždy vloženo textové pole, s jehož pomocí může uživatel v seznamu vyhledávat relevantní položky. Vyhledávání je spuštěno při libovolném uživatelském vstupu; pokud uživatel svůj dotaz vymaže, je vyhledávání zrušeno. Při vyhledávání v seznamu částí dokumentu je vyhledáváno v textech nadpisů částí a výsledný seznam je zploštěn (už není víceúrovňový). Při vyhledávání v seznamu značek je vyhledáváno pouze v názvech značek. Seznam výsledků vyhledávání je seřazen od největší shody po nejnižší shodu.

Pro vyhledávání byla využita knihovna Fuse.js, která umožňuje fuzzy vyhledávání, kdy dotaz uživatele nemusí přesně odpovídat vyhledávanému textu (dle dokumentace [40] knihovna využívá modifikovanou verzi algoritmu Bitap). Tato knihovna byla dále vybrána díky její přehledné dokumentaci, dostupnosti TypeScript *types*, permisivní open-source licenci a velké popularitě (mezi 1,5 miliony a 3 miliony týdenních stažení z npm [32] za poslední rok). Knihovna dále dle [40] umožňuje upravit mnoho různých parametrů vyhledávání, takže je v případném budoucím vývoji možno například upravit mezní hodnotu, které musí shoda dotazu a vyhledávaného textu dosáhnout pro jeho zahrnutí ve výsledcích.

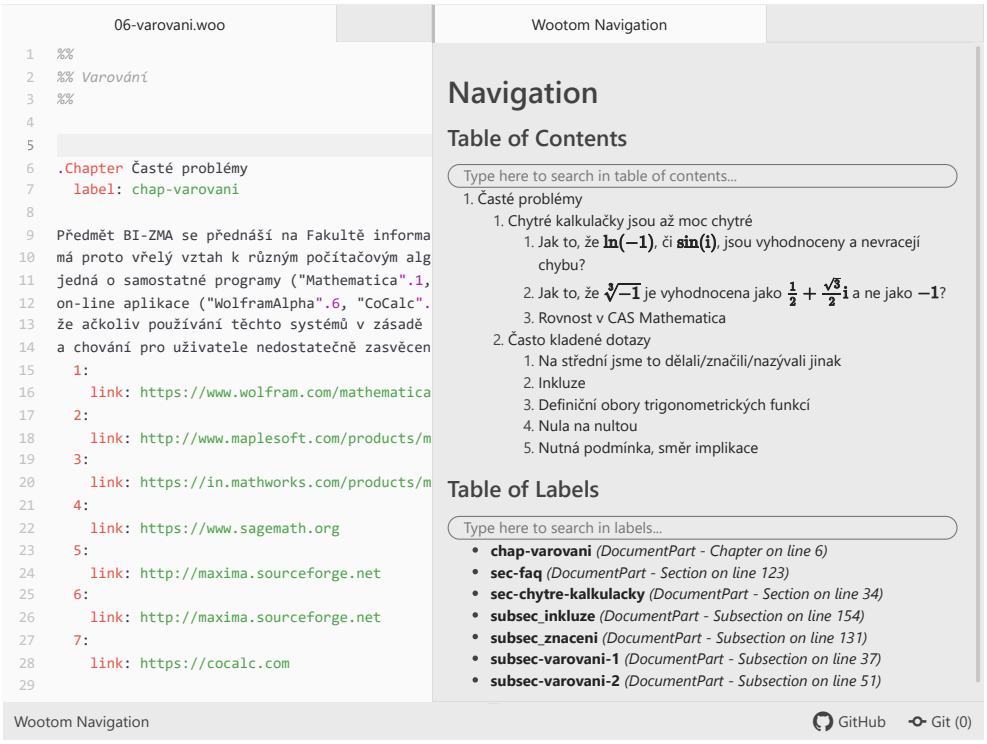
Na obrázku 2.8 je pro ilustraci zobrazeno okno Atomu s otevřeným souborem zdroje studijního textu k BI-PKM na levé straně a jeho navigačním podoknem na pravé straně.

Tímto byly realizovány funkční požadavky **F6** až **F10**. Implementaci by bylo dále možno vylepšit například o zobrazení přehledu částí a značek v celém dokumentu (nikoliv pouze v aktuálně otevřeném souboru).

2.5 Testování

Pro řádné otestování realizovaného rozšíření bylo využito jednotkových a integračních testů. Testy jsou umístěny ve složce `src/spec` a jejich výstup po kompilaci pak ve složce `spec`. Jsou spouštěny s pomocí komunitního *test runner* atom-jasmine3-test-runner, který byl zvolen na základě provedené rešerše 1.9.1. Testy tak mají (stejně jako kód samotného rozšíření) plný přístup k API Atomu a mohou pracovat s webovými APIs.

Pro přehlednost byl při tvorbě testů použit takzvaný AAA vzor – jednoduchý vzor, kdy je každý test členěn do třech částí. V první (*arrange*) části jsou dle [41] nejprve inicializovány objekty a očekávané výstupy. V další (*act*) části je pak dle [41] zavolána testovaná metoda či funkce. V poslední (*assert*) části je nakonec dle [41] ověřeno, že testovaná metoda či funkce funguje dle očekávání.



Obrázek 2.8: Navigační podokno u otevřeného souboru zdroje studijního textu k BI-PKM [2]

Především jednotkové testy pak používají *mock*, *stub* a *spy* objekty. Pro zjednodušení jejich tvorby byla využita populární knihovna Sinon.JS, která se na tuto oblast dle [42] specializuje a funguje s libovolným frameworkem pro jednotkové testování. Knihovna Sinon.JS je dle [42] dostupná pod permisivní open-source licencí (mimo jiné) z npm [32] a nabízí také *types* pro TypeScript.

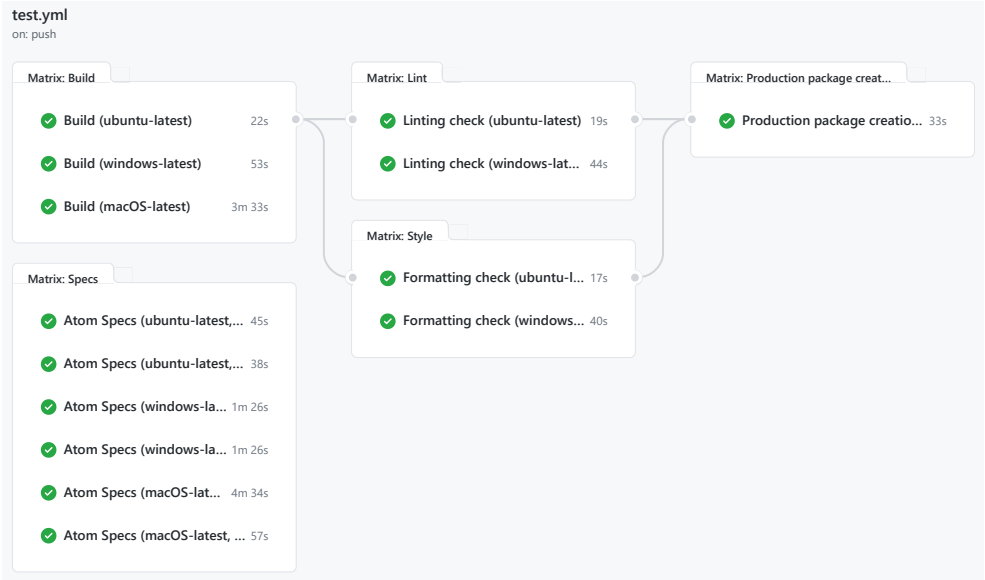
Tyto testovací objekty pak spolu s použitím interfaces a *inversion of control* techniky *dependency injection* ve zdroji rozšíření umožňují docílit (mimo jiné) izolace chování tříd. Další pomocné interfaces jsou vytvořeny pro abstrakci funkcionality vybraných tříd z Atom API (jejichž instance jsou dostupné v globální proměnné `atom`). Například třída `Workspace` z Atom API má totiž dle [5] desítky metod, z nichž realizované rozšíření jich využívá pouhé jednotky. Díky vlastním interfaces jako jsou například `WorkspaceItemManager` nebo `WorkspaceObserver` je tak docíleno *low coupling*. Jednou komplikací při vytváření *mock* objektů na základě interfaces v jazyce TypeScript je, že *mock* objekty není možno vytvářet s pomocí generujících funkcí/metod Sinon.JS. Je tomu z důvodu absence typových informací (mezi něž interfaces patří) v Ja-

vaScript výstupu (typové informace jsou dle [38] volitelně přidány pouze jako samostatné *types* soubory pro využití v dalším TypeScript kódu).

V průběhu realizace bylo dále využito jednoduchých uživatelských testů, kdy byla nově přidávaná funkcionalita vyzkoušena na WooWoo zdrojích existujících studijních textů autorem práce na operačním systému Windows 10 a vedoucím práce na Linuxové distribuci Arch Linux.

Pro automatizaci jednotkových a integračních testů bylo využito platformy pro kontinuální integraci GitHub Actions. GitHub Actions umožňují dle [43] tvorbu vlastních *actions*, které lze dále v oficiálním „obchodě“ sdílet s komunitou. Jednou z takto dostupných *actions* je Setup Atom [35], která umožňuje jednoduché nastavení Atomu pro použití v (nejen) tomto nástroji pro kontinuální integraci. Setup Atom dle [35] umožňuje použití nejnovější stabilní verze Atomu nebo jedné z vývojových verzí. Dále dle [35] umožňuje použití konkrétní verze Atomu (tato funkcionalita byla přidána na podnět autora této práce). Díky těmto možnostem tak jde docílit zajištění korektního fungování na všech *minor* verzích Atomu mezi nejstarší testovanou verzí a nejnovější stabilní verzí bez nutnosti testování na všech těchto *minor* verzích (to platí dokud nevyjde nová *major* verze a za předpokladu, že jsou řádně dodržována pravidla semantického verzování).

Minimální takto podporovaná verze Atomu byla stanovena na 1.54, což byla nejnovější verze v době začátku realizace. Testy jsou v kontinuální integraci spouštěny na nejnovějších verzích operačních systémů macOS, Ubuntu a Windows 10. Dále je kontinuální integrace využita pro automatickou kontrolu sestavení, kontrolu formátování (za pomoci nástroje Prettier), provedení statické analýzy (za pomoci nástroje ESLint) a vytvoření produkčního *package*. Průběh kontinuální integrace je ilustrován na obrázku 2.9.



Obrázek 2.9: Úspěšný běh GitHub Actions *workflow*

Závěr

Cílem této práce bylo seznámit se s nově vzniklým formátem WooWoo, prozkoumat možnosti moderních editorů či vývojových prostředí, zejména co se týče jejich rozšiřitelnosti a navrhnout, implementovat a otestovat rozšíření vybraného editoru, jehož účelem má být právě poskytnutí chybějící okamžité zpětné vazby při tvorbě WooWoo dokumentů a tím zjednodušení práce s nimi. Zejména jde pak o přehlednou prezentaci logické struktury dokumentů, zobrazování matematických výrazů a různých grafických objektů (TikZ diagramy), navigaci mezi různými částmi dokumentu a vyhledávání pomocí značek objektů, a to s využitím stylů založených na existujících šablonách.

Na základě provedené rešerše bylo vytvořeno rozšíření editoru Atom, protože poskytuje ze zkoumaných editorů největší prostor pro modifikaci a zároveň poskytuje kvalitní dokumentaci a širokou komunitu pro řešení případných problémů. Atom je také volně dostupný na všech potřebných platformách.

Toto rozšíření aktuálně nabízí živý náhled otevřeného kusu dokumentu, který přesně a přehledně prezentuje jeho logickou strukturu. Náhled také zobrazuje v dokumentu obsažené matematické výrazy s využitím knihovny MathJax 2.6, která poskytuje dostatečnou podporu LaTeX maker využitých v existujících studijních materiálech psaných s pomocí WooWoo. Náhled dále zobrazuje TikZ diagramy s využitím nativní TeX distribuce. Rozšíření poskytuje okno s přehledem částí a značek v otevřeném kusu dokumentu a fuzzy vyhledávání v jejich názvech. Bylo využito stylů založených na existujících šablonách. Rozšíření také doplňuje Atom o zvýrazňování syntaxe WooWoo formátu.

Funkčnost rozšíření byla řádně otestována s využitím jednotkových testů, integračních testů a jednoduchého uživatelského testování. Dále byly prozkoumány možnosti rozšíření editoru pro podporu více šablon a návrh i implementace probíhaly tak, aby toto případné další rozšíření šlo provést jen s drobnými modifikacemi.

Vzniklé rozšíření usnadní tvorbu studijních textů ve formátu WooWoo díky poskytnutí okamžité zpětné vazby a nabízí navíc mnoho možností pro další

budoucí rozšíření, jako je podpora zobrazení celého dokumentu (ne jen otevřeného kusu), synchronní posuv náhledu, lišta nástrojů nebo podpora více šablon a potenciální vznik celého ekosystému šablonových rozšíření. Zde implementovaný parser by se navíc mohl potencionálně využít i ve zcela jiných projektech, a to i díky použité permisivní open-source licenci a kódu dostupnému na portálu GitHub[1].

[1] GitHub: Wootom (`https://github.com/davidstraka2/wootom`)

Bibliografie

1. KALVODA, Tomáš. *WooWoo Specs* [online]. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021 [cit. 2021-05-03]. Dostupné z: `https://kam.fit.cvut.cz/deploy/woowoo-specs/woowoo-specs.pdf`.
2. KALVODA, Tomáš. *Přípravný kurz matematiky* [online]. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021 [cit. 2021-04-22]. Dostupné z: `https://kam.fit.cvut.cz/deploy/bi-pkm/mirror/`.
3. CONSORTIUM, World Wide Web. *HTML & CSS* [online]. 2016 [cit. 2021-04-24]. Dostupné z: `https://www.w3.org/standards/webdesign/htmlcss`.
4. FOUNDATION, OpenJS. *Electron Documentation* [online]. 2021 [cit. 2021-04-24]. Dostupné z: `https://www.electronjs.org/docs`.
5. GITHUB, Inc. *Atom Flight Manual* [online]. 2021 [cit. 2021-04-25]. Dostupné z: `https://flight-manual.atom.io`.
6. GITHUB, Inc. *GitHub: Atom* [online]. 2021 [cit. 2021-05-12]. Dostupné z: `https://github.com/atom/atom`.
7. CORPORATION, Microsoft. *Documentation for Visual Studio Code* [online]. 2021 [cit. 2021-05-08]. Dostupné z: `https://code.visualstudio.com/docs`.
8. CORPORATION, Microsoft. *GitHub: Visual Studio Code* [online]. 2021 [cit. 2021-05-12]. Dostupné z: `https://github.com/microsoft/vscode`.
9. KODOWA, Inc. *GitHub: Light Table* [online]. 2021 [cit. 2021-05-12]. Dostupné z: `https://github.com/LightTable/LightTable`.
10. KODOWA, Inc. *Light Table docs* [online]. 2016 [cit. 2021-05-12]. Dostupné z: `http://docs.lighttable.com`.

11. BRUNSFELD, Max. *Tree-sitter* [online]. 2021 [cit. 2021-05-08]. Dostupné z: https://tree-sitter.github.io/tree-sitter/.

12. SUBLIME HQ PTY, Ltd. *Sublime Text 3 Documentation* [online]. 2021 [cit. 2021-05-08]. Dostupné z: https://www.sublimetext.com/docs/3/.

13. ROLFS, Christopher. *Atom packages: Tasks Package* [online]. 2018 [cit. 2021-05-08]. Dostupné z: https://atom.io/packages/tasks.

14. KLING, Felix. *AST explorer* [online]. 2021 [cit. 2021-05-09]. Dostupné z: https://astexplorer.net.

15. MATHJAX CONSORTIUM, The. *MathJax 3.1 Documentation* [online]. 2021 [cit. 2021-05-09]. Dostupné z: http://docs.mathjax.org/en/v3.1-latest/.

16. MATHJAX CONSORTIUM, The. *MathJax* [online]. 2021 [cit. 2021-05-11]. Dostupné z: https://www.mathjax.org.

17. MATHJAX CONSORTIUM, The. *MathJax 2.7 Documentation* [online]. 2020 [cit. 2021-05-11]. Dostupné z: http://docs.mathjax.org/en/v2.7-latest/.

18. ACADEMY, Khan. *KaTeX* [online]. 2021 [cit. 2021-05-11]. Dostupné z: https://katex.org.

19. ACADEMY, Khan. *KaTeX Docs* [online]. 2021 [cit. 2021-05-11]. Dostupné z: https://katex.org/docs/.

20. MACKINLAY, Dan. *Mathematics without LaTeX* [online]. 2021 [cit. 2021-05-11]. Dostupné z: https://danmackinlay.name/notebook/latex_free_mathematics.html.

21. FOWLER, Jim. *GitHub: TikZJax* [online]. 2021 [cit. 2021-05-11]. Dostupné z: https://github.com/kisonecat/tikzjax.

22. MALYSHEV, Basil. *CTAN: Package bakoma-fonts* [online]. 2021 [cit. 2021-05-11]. Dostupné z: https://www.ctan.org/pkg/bakoma-fonts.

23. CORPORATION, Microsoft. *Word help & learning* [online]. 2021 [cit. 2021-05-01]. Dostupné z: https://support.microsoft.com/en-us/word.

24. GITHUB, Inc. *Atom packages: Symbols View package* [online]. 2020 [cit. 2021-05-01]. Dostupné z: https://atom.io/packages/symbols-view.

25. HIEBERT, Darren. *Exuberant Ctags* [online]. 2009 [cit. 2021-05-01]. Dostupné z: http://ctags.sourceforge.net.

26. GITHUB, Inc. *GitHub: Symbols View package* [online]. 2020 [cit. 2021-05-01]. Dostupné z: https://github.com/atom/symbols-view.

27. CHEN, Yongkang. *GitHub: Atom Ctags Package* [online]. 2021 [cit. 2021-05-01]. Dostupné z: https://github.com/yongkangchen/atom-ctags.

28. CORPORATION, Microsoft. *Language Server Protocol* [online]. 2021 [cit. 2021-05-02]. Dostupné z: https://microsoft.github.io/language-server-protocol/.

29. GITHUB, Inc. *GitHub: Atom Language Server Protocol Client* [online]. 2021 [cit. 2021-05-02]. Dostupné z: https://github.com/atom-community/atom-languageclient.

30. LABS, Pivotal. *GitHub: Jasmine Releases* [online]. 2021 [cit. 2021-04-25]. Dostupné z: https://github.com/jasmine/jasmine/releases.

31. STICKA-JONES, Kepler. *GitHub: atom-test-runner-jest* [online]. 2021 [cit. 2021-05-06]. Dostupné z: https://github.com/keplersj/atom-test-runner-jest.

32. NPM, Inc. *npm* [online]. 2021 [cit. 2021-05-06]. Dostupné z: https://www.npmjs.com.

33. TILLEY, Michelle. *GitHub: Atom Mocha Test Runner* [online]. 2021 [cit. 2021-05-06]. Dostupné z: https://github.com/atom/atom-mocha-test-runner.

34. BRIX, Tony. *GitHub: Atom Jasmine 3.x Test Runner* [online]. 2021 [cit. 2021-05-06]. Dostupné z: https://github.com/UziTech/atom-jasmine3-test-runner.

35. BRIX, Tony. *GitHub: Setup Atom and APM* [online]. 2021 [cit. 2021-05-06]. Dostupné z: https://github.com/UziTech/action-setup-atom.

36. GITHUB, Inc. *Atom packages: Autocomplete+ package* [online]. 2021 [cit. 2021-05-06]. Dostupné z: https://atom.io/packages/autocomplete-plus.

37. WHITNEY, Will. *Atom packages: Hydrogen* [online]. 2021 [cit. 2021-05-06]. Dostupné z: https://atom.io/packages/hydrogen.

38. CORPORATION, Microsoft. *TypeScript Documentation* [online]. 2021 [cit. 2021-05-10]. Dostupné z: https://www.typescriptlang.org/docs/.

39. MACROMATES, Ltd. *TextMate 1.x Manual: Language Grammars* [online]. 2021 [cit. 2021-05-12]. Dostupné z: https://macromates.com/manual/en/language_grammars.

40. RISK, Kirollos. *Fuse.js* [online]. 2021 [cit. 2021-05-10]. Dostupné z: https://fusejs.io.

41. CORPORATION, Microsoft. *Unit testing fundamentals* [online]. 2019 [cit. 2021-05-12]. Dostupné z: https://docs.microsoft.com/en-us/visualstudio/test/unit-test-basics.

42. JOHANSEN, Christian. *Sinon.JS* [online]. 2021 [cit. 2021-05-12]. Dostupné z: https://sinonjs.org.

43. GITHUB, Inc. *GitHub Features: GitHub Actions* [online]. 2021 [cit. 2021-05-12]. Dostupné z: `https://github.com/features/actions`.

Příloha **A**

Seznam použitých zkratek

AAA Arrange, Act, Assert

API Application Programming Interface

apm Atom Package Manager

AST Abstraktní syntaktický strom (z anglického Abstract Syntax Tree)

BI-PKM Přípravný kurz matematiky (předmět na FIT ČVUT)

BI-ZMA Základy matematické analýzy (předmět na FIT ČVUT)

CSON CoffeeScript Object Notation

CSS Kaskádové styly (z anglického Cascading Style Sheets)

CST Derivační strom (z anglického Concrete Syntax Tree)

CDN Content Delivery Network

DOM Document Object Model

DVI DeVice Independent

EPUB Electronic Publication

HTML Hypertext Markup Language

JSON JavaScript Object Notation

Less Leaner Style Sheets

npm Node Package Manager

PDF Portable Document Format

SVG Scalable Vector Graphics

YAML YAML Ain't Markup Language

Příloha **B**

Instalační příručka

Instalace rozšíření vyžaduje Atom 1.54 a novější (starší verze mohou fungovat, není to však otestováno).

Zobrazení TikZ obrázků dále vyžaduje lokální instalaci TEX distribuce, která musí poskytovat příkazy `latex` a `dvisvgm`. Dále musí TEX distribuce poskytovat používané *packages* (v případě použití výchozí preambule dokumentu jde o následující packages: amsmath, amssymb, bbding, fontenc, inputenc, libertine, pgfplots a standalone). Pokud TikZ obrázky nebudou použity, instalace TEX distribuce není nutná.

Rozšíření je možno nainstalovat buď z terminálu s pomocí příkazu `apm install wootom`, nebo přímo prostřednictvím uživatelského prostředí Atomu (File → Settings → Install → Vyhledejte „wootom“ → Install).

Při práci se zdroji existujících studijních textů (BI-PKM, BI-ZMA) je dále výrazně doporučeno upravit nastavení *package* kvůli podpoře vlastních LATEX maker (a přidaných LATEX *packages*). Nastavení *package* je možno upravit v File → Settings → Packages → Najděte *package* wootom → Settings. Do textového pole položky *MathJax Macros* vložte obsah souboru `settings/mathjax-macros.txt` z přiloženého CD. Do textového pole položky *TikZ Preamble* pak obdobně vložte obsah souboru `settings/tikz-preamble.txt`. Po změně nastavení je doporučeno znovu načíst Atom (pro to může být využito klávesové zkratky Ctrl + Shift + F5).

Nastavení pro vývoj

Pro vývoj rozšíření je navíc k závislostem z úvodu této přílohy vyžadována instalace Node.js (spolu s npm).

1. Získejte zdroj rozšíření (GitHub[2], přiložené CD, ...)
2. `cd` do adresáře se zdrojem

[2]GitHub: Wootom (`https://github.com/davidstraka2/wootom`)

3. `apm link`
4. `npm install`
5. `npm run build`

Užitečné příkazy

- `npm test` – Spuštění jednotkových a integračních testů
- `npm run build` – Kompilace zdroje
- `npm run format:check` – Kontrola formátování
- `npm run format:fix` – Oprava formátování
- `npm run lint:check` – Statická analýza
- `npm run lint:fix` – Statická analýza s pokusem o nápravu
- `npm run pack` – Vytvoření produkčního *package*

Příloha C

Obsah přiloženého CD

- `README.md` ..... stručný popis obsahu CD
- `impl`
  - `wootom.bundle` ..... Git *bundle* se zdrojem implementace
  - `wootom` ..... zdroj implementace
  - `wootom-dist` ..... produkční *package*
- `settings`
  - `mathjax-macros.txt` ..... doporučené nastavení MathJax Macros
  - `tikz-preamble.txt` ..... doporučené nastavení TikZ Preamble
- `thesis`
  - `BP_Straka_David_2021.bundle` ..... Git *bundle* se zdrojem práce ve formátu LaTeX
  - `BP_Straka_David_2021.pdf` ..... text práce ve formátu PDF
  - `BP_Straka_David_2021` ..... zdroj práce ve formátu LaTeX