VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

TEXTOVÝ EDITOR PRO SENIORY

TEXT EDITOR FOR SENIORS

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — Ondřej Kudela

AUTHOR

VEDOUCÍ PRÁCE — prof. Ing. Dan Komosný, Ph.D.

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Telekomunikační a informační systémy**

Ústav telekomunikací

Student: Ondřej Kudela ***ID:*** 230610

Ročník: 3 ***Akademický rok:*** 2022/23

NÁZEV TÉMATU:

Textový editor pro seniory

POKYNY PRO VYPRACOVÁNÍ:

Vytvořte základní textový editor, který bude přizpůsobený pro seniory ve věkové skupině 90 let a více. Editor bude jednoduše ovladatelný a bude podporovat pouze základní formátování pomocí speciálních znaků, například dle pravidel Markdown. Zvolte formu správy dokumentů a jejich archivace, která bude vhodná pro seniory. Editor zhotovte v programovacím jazyce Python. Výsledky práce publikujte na repozitáři GitHub pod licencí MIT.

DOPORUČENÁ LITERATURA:

Podle pokynů vedoucího práce.

Termín zadání: 6.2.2023 ***Termín odevzdání:*** 26.5.2023

Vedoucí práce: prof. Ing. Dan Komosný, Ph.D.

prof. Ing. Jiří Mišurec, CSc.
předseda rady studijního programu

Abstrakt

Bakalářská práce se zabývá návrhem a implementací jednoduchého textového editoru přizpůsobeného pro seniory ve věkové skupině 90 let a více. Textový editor je jednou z aplikací operačního systému pro seniory, který slouží pro ulehčení práce na počítači. Cílem práce je sestavit jednoduše ovladatelný textový editor podporující základní formátování pomocí značkovacího jazyka Markdown a implementovat vhodnou formu správy dokumentů a jejich archivace. Práce popisuje zhotovený programový kód, překlad jazyka Markdown do dokumentu PDF, formátování textu, práci se soubory, vícejazyčný překlad aplikace, hlasovou asistenci a vývoj grafického prostředí. Výsledky práce jsou publikovány na repozitáři GitHub.

Klíčová slova

Textový editor, Operační systém, Senior, Markdown, PDF, Python, PyQt5

Abstract

The thesis deals with the design and implementation of a simple text editor adapted for seniors in the age group of 90 years and more. The text editor is one of the applications of the operating system for the seniors, which is used to facilitate the work on the computer. The goal of this work is to build an easy to use text editor supporting basic formatting using Markdown markup language and to implement a suitable form of document management and archiving. The work describes the created source code, the translation of the Markdown language into a PDF document, text formatting, file handling, multilingual translation of the application, voice assistance, and the development of the graphical environment. The results of the work are published on the GitHub repository.

Keywords

Text editor, Operating system, Senior, Markdown, PDF, Python, PyQt5

Bibliografická citace

KUDELA, Ondřej. Textový editor pro seniory [online]. Brno, 2023 [cit. 2023-05-16]. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/151061. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce Dan Komosný.

Prohlášení autora o původnosti díla

Jméno a příjmení studenta:	*Ondřej Kudela*
VUT ID studenta:	*230610*
Typ práce:	*Bakalářská práce*
Akademický rok:	*2022/23*
Téma závěrečné práce:	*Textový editor pro seniory*

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

.

V Brně dne: ----------

podpis autora

Poděkování

Rád bych poděkoval vedoucímu bakalářské práce panu prof. Ing. Danovi Komosnému, Ph.D. za účinné vedení, kvalitní konzultace, trpělivost, a hlavně podnětné návrhy k práci.

Obsah

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM VÝPISŮ

Úvod

Svět internetu, webových a desktopových aplikací může být pro dnešního seniora velmi složitý. Na popud těchto problémů vznikl projekt s názvem Senior Operating System, který má seniorovi zjednodušit práci s počítačem pro používání na denní bázi. Textové editory jako je např. Microsoft Word mohou být pro seniora příliš komplikované. Proto byl vytvořen jednoduchý textový editor, který je součástí zmiňovaného operačního systému pro seniory, který má práci s textovými dokumenty zjednodušit. Pro zajištění jednoduché manipulace s textovým editorem byly vytvořeny pouze základní funkce sloužící pro práci se soubory a editaci textu, kdy formátování textu zajišťuje značkovací jazyk Markdown. Text ve formě Markdown je možné uložit do formátu PDF jako oficiální výstup textového editoru. Ke zvýšení uživatelské přívětivosti byl dále implementován hlasový asistent, vícejazyčný překlad aplikace a algoritmus pro automatické průběžné ukládání dokumentů.

V teoretické části práce je popsáno, co je to značkovací jazyk Markdown, jak funguje jeho překlad do HTML, jak jsou formáty Markdown a HTML propojeny, jaká je motivace k použití jazyka Markdown a syntaxe základních značek Markdown. Dále pojednává o formátu PDF, jeho historii, struktuře souboru a dokumentu, objektech, základní grafice, motivaci použití a omezení PDF a programových knihoven, které byly použity při řešení bakalářské práce.

V praktické částí je podrobně popsána vize a implementace textového editoru, vývoj grafického rozhraní, překlad značkovacího jazyka Markdown do formátu PDF pomocí knihoven *Python-Markdown* a *xhtml2pdf*. Popis jednotlivých funkcí pro správu souborů, formátování textu, algoritmus automatického průběžného ukládání dokumentů, vícejazyčný překlad aplikace, hlasová asistence a možná vylepšení aplikace.

V samotném závěru bakalářské práce jsou shrnuty její výsledky, popis řešené problematiky a navrženy cíle pro možná vylepšení aplikace.

1. ZNAČKOVACÍ JAZYK MARKDOWN

Markdown je značkovací jazyk, který byl vytvořen Johnem Gruberem v roce 2004 a je jedním z nejpopulárnějších značkovacích jazyků na světě, hlavně díky své jednoduchosti. Je to svobodný software dostupný za podmínek BSD-style open source licence. [1]

Markdown je oficiálně nástrojem pro převod textu do značkovacího jazyka HTML neboli Hypertext Markup Language. Umožňuje psát pomocí snadno čitelného a zapisovatelného formátu prostého text a převést jej do jazyka HTML, či XHTML. [2]

Používání jazyka Markdown je jiné, než používání editorů jako je třeba Microsoft Word, kde stačí kliknout na odpovídající tlačítko formátování a změna formátu textu se provede okamžitě [1]. V případě souboru formátu Markdown je postup jiný. Je zapotřebí přidat do textu znaménka formátovaní prostého textu, jak by formátovaní mělo vypadat, kdy se formát textu změní až po vykreslení do příslušného formát, například HTML.

Cílem pro syntaxi formátování značkovacího jazyka Markdown je, aby byla co nejčitelnější. Vize návrhu syntaxe byla taková, že dokument ve formátu Markdown by měl být publikován jako prostý text, aniž by vypadal, že je označen tagy k jeho formátování. To znamená, že text ve formátu Markdown lze číst, i když není vykreslen. Na první pohled se může zdát, že syntaxe Markdownu byla ovlivněna převážně filtry pro převod textu do HTML, jako jsou Setext, atx, Textile, reStructuredText aj. Největším zdrojem inspirace autora pro syntaxi Markdown byl však formát e-mailu prostého textu. [2]

Za tímto účelem se syntaxe jazyka Markdown skládá především z interpunkčních znaků, které vypadají tak, jako co znamenají. Např. hvězdičky, které kolem slova zvýrazňují text, mají opravdu úkol syntaxe text zvýraznit tučně *text* (*bold*). Tedy Markdown seznamy vypadají opravdu jako seznamy, blokové uvozovky vypadají opravdu jako citované pasáže textu. [2]

1.1 Překlad Markdown do jazyka HTML

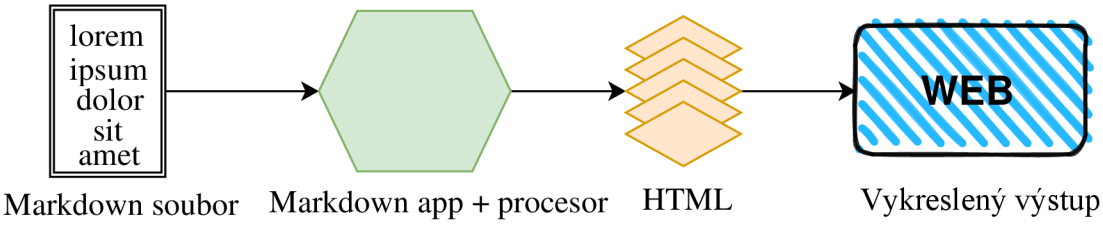
Text psaný ve značkovacím jazyce Markdown je uložen v souboru prostého textu, který ještě není nijak vykreslen a má příponu .md nebo .markdown. Pro požadované vykreslení textu je zapotřebí použít některou z aplikací, která umí zpracovat Markdown na HTML nebo textový editor podporující práci s Markdown. Těchto aplikací existuje mnoho a vypadají podobně jako Microsoft Word. Všechny z nich ale dělají totéž, a to překlad formátu Markdown na HTML, jež lze zobrazit ve webovém prohlížeči jako vykreslený text. [1]

Aplikace pracující s Markdown běžně používají procesor Markdown, který umožňuje překlad textu ve formátu Markdown do formátu HTML. Celkový proces vytvoření textu Markdown až k finálnímu vykreslení je sestaven ze čtyř kroku [1]:

1. Vytvoření textového souboru Markdown s příponou .md nebo .markdown pomocí textového editoru nebo speciální aplikace.
2. Aplikace otevře soubor Markdown.
3. Aplikace provede překlad souboru Markdown na formát HTML, pomocí Markdown procesoru.
4. HTML soubor je možné zobrazit na webovém prohlížeči nebo převést do jiného formátu souboru, např. PDF. [1]

Obr. 1.1: Princip překladu souboru Markdown do jazyka HTML [1]

Daný proces se může lišit v závislosti použité aplikace. Většina aplikací totiž kombinuje kroky 1-3 do jednoho kroku, tedy celý proces je pro uživatele méně viditelný. [1]



1.2 Motivace a výhody použití Markdown

Rychlost psaní a jednoduchost: Celý text je možné psát, včetně formátování, bez jakéhokoliv používání myši, či klávesových zkratek [3]. U editorů, jako je např. Microsoft Word, je nutné formátování volit myší, či klávesovými zkratkami přiřazených formátovacích funkci. Toto u značkovacího jazyka Markdown neplatí. Stačí jednoduše vepsat do prostého textu značky, které chce uživatel k formátování použít, tedy není zapotřebí ztrácet čas, a hlavně pozornost potvrzováním formátování (myší) textu v navigačním panelu.

Volně přenositelný: Textový soubor psaný ve formátu Markdown lze otevřít pomocí téměř všech aplikací (textových editorů, webových prohlížečů aj.). Jeho kompatibilnost nabízí jednoduchý přenos souborů Markdown mezi textovými editory, pokud např. uživatelovi používaný editor už nevyhovuje nebo nesplňuje jeho priority [1]. Toto je v rozporu z ostatními formáty textových souborů jako je třeba Word, které mají svůj speciální syntax formátování a nejsou lehko přenositelné.

Nezávislý na platformě: Text ve formátu Markdown je možné vytvořit na jakémkoliv zařízení (např. PC, telefon) s jakýmkoliv operačním systémem (např. Windows, Linux, Android) [1].

Má široké použití: Značkovací jazyk Markdown lze použit pro řadu případů. Nejpoužívanější je pro webové stránky, díky snadnému převedení Markdown na HTML [3]. Dále se ale také používá pro psaní dokumentů, poznámek, prezentací, e-mailových zpráv, knih či technické dokumentace [1].

Čitelný v budoucnosti: Díky tomu, že soubory Markdown jsou psány pomocí prostého textu, nemůže se stát, že v budoucnu nebude existovat aplikace, která by nedokázala text zobrazit nebo přečíst. Tato vlastnost je především důležitá, pokud se jedná např. o univerzitní práce, knihy, které je třeba uchovat na dobu neurčitou. [1]

Výhoda prostého textu: Text psaný pomocí jazyka Markdown je čistý text, díky čemuž nehrozí, že skryté formátování nějakým způsobem poškodí psaný text nebo programovaný kód [3].

Používaný všude: Značkovací jazyk Markdown podporován mnoha webovými stránkami jako jsou Reddit nebo pro programátory známý GitHub [3]. Také je podporovaný mnoha programy, či webovými aplikace.

1.3 Využití Markdown v HTML

Syntaxe Markdown je podle autora určena k jedinému účelu, a to k použití formátu pro psaní webových stránek (jeho obsahu), avšak není náhradou HTML, ani se k tomu neblíží. Kvůli tomu, že jeho syntaxe je velmi omezená, odpovídá pouze velmi malému množství HTML značek. Cílem syntaxe nebylo usnadnit vkládání HTML značek, ty už se podle autora snadno vkládaly, ale usnadnit čtení a psaní prostého textu (obsahu webové stránky). Je si třeba uvědomit, že Markdown je formát zápisu, HTML je publikační formát, tedy formátovací syntaxe Markdownu má pouze za úkol řešit problémy, které lze vyjádřit jako prostý text. [4]

Pokud není nějaké označení pokryto syntaxí Markdown, jednoduše je možné použít samostatný HTML syntax. Není třeba jej nějak speciální uvádět či označovat, že syntax z Markdown přechází na HTML [4], Markdown si se syntaxí HTML jednoduše poradí při použití adekvátních HTML značek.

Jediným omezením tohoto značkovacího jazyka je, že prvky HTML na blokové úrovni (<p>, , <table> atd.) musí být odděleny od ostatního obsahu prázdnými řádky, kdy počáteční a koncové značky bloku nemohou být odsazeny tabulátorem, či mezerou [4]. Příklad blokových elementů:

Výpis 1.1: Příklad blokových elementů

```
Zde se nachází text.

<table>
<tr>
                <td>Markdown</td>
                <td>HTML</td>
</tr>
</table>

Zde se nachází další text.
```

Je třeba si uvědomit, pokud už jsou použity značky HTML ve formátu Markdown, už není možné použít do těchto HTML značek značky Markdownu. Toto nekorektní použití značek by mohlo vypadat nějak takto: `<p>Toto slovo je *zvýrazněno*</p>`

Korektním použití značek v tomto případě vypadá následovně: `<p>Toto slovo je <b>zvýrazněno</b></p>`

Únik speciálních znaků

V jazyce HTML existují dva znaky < a &., které vyžadují speciální zacházení, protože levá lomená závorka slouží k zahájení tagů (značek) a ampersandy se používají k označení entit HTML. K použití těchto znaků v HTML je potřeba tyto znaky escapovat pomocí zápisu *<* a *&amp* [4].

Zvláštní opatrnost je třeba dbát především při použití ampersandu. Například text *A&B* je třeba psát v HTML jako *A&B*. Obzvlášť zvýšené opatrnosti je třeba dbát při zapisování URL adres, kde se nachází ampersandy. Jedná se o nejčastější zdroj chyb validace HTML stránek. [4]

Markdown umožňuje používat tyto speciální znaky přirozeně a únik vykonává za uživatele. Markdown automaticky překládá speciální znaky, jak bylo popsáno výše, bez nutnosti manuálního zásahu uživatele. [4]

1.4 Značky Markdown

1.4.1 Formátování textu

Nadpisy

Typickým znakem Markdown pro vytvoření nadpisu je # před slovem (mezi slovem a hashtagem je vyžadována mezera), který má být nastaven jako nadpis [5]. Podle počtu hashtagů překladač určí, o jakou úroveň nadpisu se jedná. Celkově je možné využít 6 různých úrovní, od přidání jednu hashtagů až šesti. Z hlediska kompatibility syntaxe nadpisů je doporučené vložit prázdný řádek před a za nadpisem. Tento zápis odpovídá zápisu HTML nadpisů h1-h6.

Tab. 1.1: Syntaxe nadpisů Markdown s překladem do HTML a následného vykreslení [5]

Markdown	HTML	Vykreslený výstup
# Úroveň nadpisu 1	Úroveň nadpisu 1	**Úroveň nadpisu 1**

## Úroveň nadpisu 2	Úroveň nadpisu 2	**Úroveň nadpisu 2**
### Úroveň nadpisu 3	Úroveň nadpisu 3	**Úroveň nadpisu 3**
#### Úroveň nadpisu 4	Úroveň nadpisu 4	**Úroveň nadpisu 4**
##### Úroveň nadpisu 5	Úroveň nadpisu 5	**Úroveň nadpisu 5**
###### Úroveň nadpisu 6	Úroveň nadpisu 6	**Úroveň nadpisu 6**

Alternativní zápis nadpisů lze provést přidáním libovolný počet znaků == pro úroveň nadpisu jedna a znaků -- pro úroveň nadpisu dva pod řádek textu, který chce uživatel určit jako nadpis [6]. Tento alternativní zápis pracuje pouze s úrovní nadpisu jedna a dva (h1 a h2).

Tab. 1.2: Syntaxe alternativních nadpisů Markdown s překladem do HTML a následného vykreslení [5]

Markdown	**HTML**	**Vykreslený výstup**
Úroveň nadpisu 1 ==========	Úroveň nadpisu 1	**Úroveň nadpisu 1**
Úroveň nadpisu 2 ----------	Úroveň nadpisu 2	**Úroveň nadpisu 2**

Tučné písmo (zvýraznění)

Pro tučné zvýraznění textu se využívá speciální znaků dvou hvězdiček ** nebo dvou podtržítek __ před a za slovo nebo frázi. Pokud uživatel chce zvýraznit pouze některá písmena ve slově, je nutné použít dvě hvězdičky před a za tyto písmena. V tomto případě nelze použít dvě podtržítka, ty se využívají pouze pro celá slova nebo fráze.

Tab. 1.4: Syntaxe tučného textu Markdown s překladem do HTML a následného vykreslení. [5]

Markdown	**HTML**	**Vykreslený výstup**
Toto je tučný **text**.	Toto je tučný text.	Toto je tučný **text**.

Toto je tučný __text__.	Toto je tučný text.	Toto je tučný **text**.
Toto je tučný **ex**t	Toto je tučný text	Toto je tučný **ex**t.

Zápis syntaxe hvězdiček a podtržítek se neshodují. Podtržítka nemají uplatnění při zvýraznění konkrétních písmen v textu, protože je vyžadována mezera před použití podtržítek. Kvůli tomu je doporučené používat převážně hvězdičky pro zvýraznění textu díky variabilitě a kompatibilnosti (zajišťuje méně syntaxových chyb).

Kurzíva

Kurzíva v Markdown je stylem zápisu syntaxe téměř stejná jako zvýraznění textu. Je třeba namísto dvou hvězdiček nebo podtržítek použít pouze jednu hvězdičku * nebo jedno podtržítko _ [5].

Tab. 1.5: Syntaxe kurzívy Markdown s překladem do HTML a následného vykreslení [5]

Markdown	**HTML**	**Vykreslený výstup**
Toto je *kurzíva*.	Toto je kurzíva.	Toto je *kurzíva*.
Toto je _kurzíva_.	Toto je kurzíva. >	Toto je *kurzíva*.
Toto je ku*rzí*va.	Toto je kurzíva	Toto je ku*rzí*va.

Kombinace kurzívy a tučného textu

Kombinace kurzívy a tučného textu zároveň lze aplikovat na text použitím znaků tří hvězdiček ***, tří podtržítek ___, anebo jejich kombinace (např. *__ nebo **_). Syntaxe tří hvězdiček lze použít pro konkrétní písmena v daném slově [5]. Opět obecným doporučením, jako u používání kurzívy a zvýraznění textu, je využívat především zápisu tří hvězdiček díky zvýšené kompatibilnosti a variabilitě.

Tab. 1.6: Syntaxe kombinace kurzívy a tučného textu Markdown s překladem do HTML a následného vykreslení [5]

Markdown	**Vykreslený text**
Kombinace ***kurzívy a zvýraznění***	Kombinace ***kurzívy a zvýraznění***

Kombinace ___kurzívy a zvýraznění___	Kombinace ***kurzívy a zvýraznění***
Kombinace *__kurzívy a zvýraznění__*	Kombinace ***kurzívy a zvýraznění***
Kombinace ***kurzívy* a zvýraznění**	Kombinace ***kurzívy* a zvýraznění**
Kombinace kurzívy a z***výrazně***ní	Kombinace kurzívy a z***výrazně***ní
HTML	
Kombinace kurzívy a zvýraznění	
Kombinace kurzívy a zvýraznění	
Kombinace kurzívy a zvýraznění	
Kombinace kurzívy a zvýraznění	
Kombinace kurzívy a zvýraznění	

1.4.2 Odstavce

K vytvoření odstavce není nutnost využít žádného speciálního znaku. Odstavce prostého textu se převedou automaticky do HTML jako odstavce . Doporučený postup při zápisu odstavců je neodsazování prostého textu mezery nebo tabulátory. V případě nutnosti odřádkování textu (v HTML znakem <br>) je doporučeno ukončit řádek dvěma či více mezery a poté se přesunovat na nový řádek odřádkováním pomocí tlačítka *Enter*. Další možností odřádkování je možné použít HTML tag <br> místo dvou mezer, ale jenom pokud daný překladač Markdown podporuje použití HTML syntaxe do syntaxe Markdownu, [5]

Tab. 1.3: Syntaxe odstavců Markdown s překladem do HTML a následného vykreslení [5]

Markdown	**Vykreslený text**
Odstavce v Markdown jsou velice jednoduché. Příklad nepoužití dvou mezer na konci minulého řádku pro odřádkování. Zde už bylo použito dvou mezer na minulém řádku pro odřádkování. Zde byl naopak použit tag br na minulém řádku pro odřádkování.	Odstavce v Markdown jsou velice jednoduché. Příklad nepoužití dvou mezer na konci minulého řádku pro odřádkování. Zde už bylo použito dvou mezer na minulém řádku pro odřádkování. Zde byl naopak použit tag br na minulém řádku pro odřádkování. Mezera mezi odstavci bez nutnosti použití dvou mezer.

Mezera mezi odstavci bez nutnosti použití dvou mezer.	
HTML	
Odstavce v Markdown jsou velice jednoduché. Příklad nepoužití dvou mezer na konci minulého řádku pro odřádkování. Zde už bylo použito dvou mezer na minulém řádku pro odřádkování. Zde byl naopak použit tag br na minulém řádku pro odřádkování. Mezera mezi odstavci bez nutnosti použití dvou mezer.	

1.4.3 Odsazení bloku textu

K odsazení bloku je přidělen speciální znak >, který se používá vždy na začátku řádku bloku textu oddělený mezerou od prvního písmene v řádku. Pro odsazení více bloků textu je třeba označit tímto znakem každý blok (řádek) samostatně. Tímto způsobem mohou být odsazeny jednotlivé prvky Markdown, jako jsou např. nadpisy nebo seznamy. [6]

Tab. 1.7: Syntaxe k odsazení bloku textu Markdown s překladem do HTML a následného vykreslení [5]

Markdown	**Vykreslený text**
Standardní odstavec > Použití blokových uvozovek. > > ### Nadpis 3 > - odrážka 1 > - odrážka 2	Standardní odstavec Použití blokových uvozovek. Nadpis 3 • odrážka 1 • odrážka 2
HTML	
Standardní odstavec <blockquote> Použití blokových uvozovek. Nadpis 3 - odrážka 1 - odrážka 2 </blockquote>	

Pro odsazení textu je možné použít více úrovní, jako tomu bylo třeba u nadpisů (h1-h6). Víceúrovňové odsazení lze jednoduše aplikovat zápisem vyššího počtu znaků >, dle úrovně, kterou chce uživatel použít. [5]

Tab. 1.8: Syntaxe víceúrovňového odsazení bloku textu Markdown s překladem do HTML a následného vykreslení [5]

Markdown	Vykreslený text
Standardní odstavec > Použití blokových uvozovek. > >> Vnořené blokové uvozovky >>> Vnořené blokové uvozovky	Standardní odstavec Použití blokových uvozovek. Vnořené blokové uvozovky Vnořené blokové uvozovky
HTML	
Standardní odstavec <blockquote> Použití blokových uvozovek. <blockquote> Vnořené blokové uvozovky <blockquote> Vnořené blokové uvozovky </blockquote> </blockquote> </blockquote>	

Kvůli kompatibilitě zápisu je obecným doporučením vložit prázdný řádek před a za blok textu.

1.4.4 Seznamy

Seznamy obecně slouží k přehlednému uspořádání textu. Markdown podporuje uspořádané (číslované) a neuspořádané (odrážkové) seznamy. Tyto znaky se umisťují na začátek řádku textu, který chce uživatel označit jako seznam [5]. I zde je možné využít více úrovní seznamu, kdy mezerou před speciálním znakem seznamu lze nastavit danou úroveň.

Pro použití **neuspořádaného seznamu**, lze použít zápis jednoho ze tří znaků – hvězdičky, znaménka plus nebo pomlčky [5]. Obecným doporučením je používat pouze jeden typ znaku (např. pomlčku) a vyvarovat se jakékoliv kombinace těchto tří znaků.

Tab. 1.9: Syntaxe neuspořádaného seznamu Markdown s překladem do HTML a následného vykreslení [5]

Markdown	HTML	Vykreslený výstup

- První položka
- Druhá položka
  - Třetí položka
  - Čtvrtá položka
	
	 - První položka
	 - Druhá položka
	 - Třetí položka
	 - Čtvrtá položka
	
		• První položka
		• Druhá položka
		  ○ Třetí položka
		  ▪ Čtvrtá položka

Pro použití **uspořádaného seznamu** je nutno použit jakékoliv číslo zakončené tečkou a oddělné mezerou před textem, který chceme uvést do číslovaného seznamu [5]. Číslo může být opravdu jakékoliv, Markdown totiž čísluje vždy od jedničky postupně nahoru. Je dokonce možné použít pokaždé stejné číslo na začátku řádku, což ale není doporučené použití, hlavně pokud se seznamem je naplánována další práce na HTML stránce. Jestliže z nějakého důvodu uživatel potřebuje začít seznam číslovat od jiného čísla než je jedna, tak tuto změnu musí provést až v HTML kódu po překladu.

Tab. 1.10: Syntaxe uspořádaného seznamu Markdown s překladem do HTML a následného vykreslení [5]

Markdown	**HTML**	**Vykreslený výstup**
1. První položka 2. Druhá položka 3. Třetí položka	 - První položka - Druhá položka - Třetí položka 	1. První položka 2. Druhá položka 3. Třetí položka
1. První položka 1. Druhá položka 1. Třetí položka	 - První položka - Druhá položka - Třetí položka 	1. První položka 2. Druhá položka 3. Třetí položka
5. Pátá položka 6. Šestá položka 7. Sedmá položka	<ol start="5"> - Pátá položka - Šestá položka - Sedmá položka 	5. Pátá položka 6. Šestá položka 7. Sedmá položka
1. První položka 2. Druhá položka 1. Druhá úroveň 2. Druhá úroveň 3. Třetí položka	 - První položka - Druhá položka - Druhá úroveň - Druhá úroveň - Třetí položka 	1. První položka 2. Druhá položka 1. Druhá úroveň 2. Druhá úroveň 3. Třetí položka

1. První položka 2. Druhá položka - Odrážka - Odrážka 3. Třetí položka	 - První položka - Druhá položka - Odrážka - Odrážka - Třetí položka 	1. První položka 2. Druhá položka ○ Odrážka ○ Odrážka 3. Třetí položka

Do seznamu lze přidávat i ostatní prvky Markdown. Daným prvkem je třeba ale odsadit čtyřmi mezerami nebo tabulátorem od začátku nového řádku a zároveň oddělit řádkem před a za tímto prvkem od bodů seznamu. [5]

Tab. 1.11: Syntaxe kombinace seznamu ostatních prvků Markdown s překladem do HTML a následného vykreslení [5]

Markdown	**HTML**	**Vykreslený výstup**
1. První položka. Odstavec 3. Druhá položka. > B. uvozovky 5. Třetí položka.	 - První položka. Odstavec - Druhá položka. <blockquote> B. uvozovky </blockquote> - Třetí položka. 	1. První položka. Odstavec 2. Druhá položka. B. uvozovky 3. Třetí položka.

1.4.5 Odkazy

Pro vytvoření odkazu, je nutné uzavřít text odkazu do hranatých závorek a ihned po hranatých závorek vložit URL adresu do klasických závorek [7].

Tab. 1.12: Syntaxe odkazů Markdown s překladem do HTML a následného vykreslení [5]

Markdown	Oficiální dokumentace [Markdown](https://daringfireball.net/projects/markdown/).
HTML	Oficiální dokumentace Markdown

Vykreslený výstup	Oficiální dokumentace Markdown.

Podobným způsobem lze jednoduše vkládat i celou URL adresu nebo e-mailovou adresu, kdy požadovanou URL stačí pouze vložit do hranatých závorek [5]. URL adresu lze také vložit volně bez hranatých závorek a bude běžně funkční, ale není to doporučený postup, hlavně z hlediska přehlednosti.

Tab. 1.13: Syntaxe URL adres Markdown s překladem do HTML a následného vykreslení [5]

Markdown	<https://daringfireball.net/projects/markdown/>
HTML	 https://daringfireball.net/projects/markdown/
Vykreslený výstup	https://daringfireball.net/projects/markdown/

Je také třeba podotknout, že i odkazy jsou jakkoliv kompatibilní s formátováním textu (odkaz lze zvýraznit, či napsat kurzívou).

1.4.6 Obrázky

Pro vložení obrázku pomocí Markdown je využívána podobná syntaxe jako u odkazů. Syntaxe začíná vykřičníkem, za nímž se v bezprostřední vzdálenosti nachází hranaté závorky, v nichž je napsaný alternativní text, který se zobrazí při nenačtení obrázku. Za hranatými závorkami se nachází jednoduché závorky, v nichž na první pozici je definována cesta k obrázku, buď URL odkazem nebo cestou k souboru na disku, a na druhé pozici oddělen mezerou je vepsán v uvozovkách titulek textu obrázku, který se zobrazí při podržením kurzoru myši na obrázku. Tento titulek je ale volitelným parametrem. Popřípadě lze ještě za celý zápis do kulatých závorek vepsat URL adresu, na kterou je uživatel přesměrován po kliknutí na daný obrázek. [7]

Tab. 1.14: Syntaxe obrázku Markdown s překladem do HTML a následného vykreslení [5]

Markdown	![Markdown Text] (/images/markdown.jpg "Markdown")
HTML	<img src="/images/markdown.jpg" alt="Markdown Text" title="Markdown">

Vykreslený výstup	Markdown

1.4.7 Tabulky

K přidání tabulky slouží zápis, kdy se využívá tří nebo více pomlček k vytvoření záhlaví tabulky a každý sloupec je definován oddělovací čarou | (každý sloupec je ohraničen touto dělicí čarou). Šířka buněk se může lišit, podle vloženého textu v buňce. Z hlediska přehlednosti lze tedy text v buňce doplnit mezerami, aby každá buňka řádku sloupce byla stejně rozložená [8]. Tyto mezery pro vyplnění buňky jsou zobrazeny pouze v textovém souboru Markdown, po jeho přeložení do HTML se tyto mezery v buňkách tabulky již nezobrazují. Toto doplnění mezer ovšem není povinnost, jedná se pouze o doporučení pro zachování přehlednosti.

Tab. 1.15: Syntaxe tabulky Markdown s překladem do HTML a následného vykreslení [8]

Markdown	HTML	Vykreslený výstup
| Markdown | Text | | --- | --- | | Formátovací | Tučný | | Jazyk | Kurzíva |	<table> <thead> <tr> <th>Markdown</th> <th>Text</th> </tr> </thead> <tbody> <tr> <td>Formátovací</td> <td>Tučný</td> </tr> <tr> <td>Jazyk</td> <td>Kurzíva</td> </tr> </tbody> </table>	**Markdown** **Text** Formátovací Tučný Jazyk Kurzíva
|Markdown|Text| |---|---| |Formátovací|Tučný| |Jazyk| Kurzíva|		**Markdown** **Text** Formátovací Tučný Jazyk Kurzíva

Jak si lze všimnout, vygenerovaná tabulka z Markdown není nijak přehledná, spíše vypadá jako prostý text, který je pod sebe zarovnán. Proto se většinou využívají v zápisu Markdown ještě CSS styly, které tento nedostatek potlačují.

Také lze pro lepší přehlednost navolit zarovnávání textu v každém sloupci zvlášť pomocí přidání dvojteček na začátek, konec, anebo jako ohraničení tří pomlček záhlaví. [8]

Tab. 1.16: Syntaxe tabulky se styly CSS a zarovnáním textu Markdown s překladem do HTML a následného vykreslení [8]

Markdown	**HTML**

<style>
table {
  border-collapse: collapse;
  width: 90%;
  font-family: arial;
}
 	
td, th {
  border: 1px solid #aaaaaa;
  padding: 10px;
}
 	
tr:nth-child(even) {
  background-color: #dddddd;
}
</style>
 	
| Markdown | Text | Bloky |
| :--- | :---: | ---: |
| Formátovací | Tučný | Tabulka |
| Značkovací | Kurzíva | Kód |
| Jazyk | Kombinace | Odstavec |
| MD | Zarovnání | Citace |
	<table>
	<thead><tr>
	<th style="text-align:left">Markdown</th>
	<th style="text-align:center">Text</th>
	<th style="text-align:right">Bloky</th>
	</tr></thead>
	<tbody><tr>
	<td style="text-align:left">Formátovací</td>
	<td style="text-align:center">Tučný</td>
	<td style="text-align:right">Tabulka</td>
	</tr><tr>
	<td style="text-align:left">Značkovací</td>
	<td style="text-align:center">Kurzíva</td>
	<td style="text-align:right">Kód</td>
	</tr><tr>
	<td style="text-align:left">Jazyk</td>
	<td style="text-align:center">Kombinace</td>
	<td style="text-align:right">Odstavec</td>
	</tr><tr>
	<td style="text-align:left">MD</td>
	<td style="text-align:center">Zarovnání</td>
	<td style="text-align:right">Citace</td>
	</tr></tbody>
	</table>

Vykreslený výstup	

Markdown	Text	Bloky
Formátovací	Tučný	Tabulka
Značkovací	Kurzíva	Kód
Jazyk	Kombinace	Odstavec
MD	Zarovnání	Citace

1.4.8 Únik speciálních znaků

Někdy je potřeba speciální znaky jazyka Markdown *escapovat*, k čemuž se používá zpětné lomítko \. Nejčastěji se používá únik speciálních znaků pro řadové číslovky na začátku věty na novém řádku, aby nevznikl omylem seznam. Využitích se najde ale mnohem více. [5]

Tab. 1.17: Únik znaků ve značkovacím jazyce Markdown [5]

Markdown	HTML	Vykreslený výstup
Příklad *hvězdiček*	Příklad *hvězdiček*	Příklad *hvězdiček*
\# Toto není nadpis	# Toto není nadpis	# Toto není nadpis
3\. místo obsadil...	3. místo obsadil...	3. místo obsadil...

2. FORMÁT PDF

Portable document format, také nazývaný jako PDF, je univerzální formát souborů vytvořený společností Adobe, který umožňuje spolehlivý a snadný způsob prezentace a výměny dokumentů nezávisle na použitém software hardwaru nebo operačním systému. Soubory PDF zachovávají formátování dokumentu, což zajišťuje, že sdílený soubor se u každého uživatele zobrazuje totožně. [16]

Dokumenty PDF zpravidla obsahují text, hypertextové odkazy, tlačítka, pole formulářů, ale také zvuky a videa. Tyto dokumenty lze elektronicky podepsat a snadno prohlížet nezávisle na operačním systému, například pomocí prohlížečů (webových) nebo odpovídajícímu softwaru (Adobe Acrobat Reader) sloužícího pro práci s PDF. Avšak u mnohých PDF prohlížečů nejsou aktivní typy obsahu, jako jsou například videa, zvukové nahrávky, grafika, podporované. Soubory PDF obsahují příponu .pdf. Jedná se o jeden z nejpoužívanějších typů souborů v dnešní době. [19], [16]

2.1 Historie

Před vytvořením formátu PDF byla myšlenka zasílání textových a grafických dokumentů e-mailem nepředstavitelná. V té době byl email považován vůči dnešním standardům za primitivní [17]. Spoluzakladatel společnosti Adobe Dr. John Warnock přišel v roce 1991 s nápadem zvaným The Camelot Project, který měl za úkol převést papírové dokumenty do digitální formy. Warnock se rozhodl vytvořit způsob, jakým by bylo možné dokumenty z jakékoliv aplikace jednoduše ukládat bez ztráty kvality a přebytečného množství dat, sdílet mezi ostatními uživateli, prohlížet a tisknout na kterémkoli počítači (s jakýmkoliv operačním systémem či softwarem). [16]

Nový formát PDF, vytvořený sloučením a vylepšením dvou již existujících technologií PostScript a Adobe Illustrator, byl dokončen v roce 1993 a představen světu [17]. PDF se však ve společnosti v prvních letech od svého vydání neujalo, protože v té době byly počítače považovány za luxus. Jeho popularita se rozšířila až v roce 1996, kdy společnost IRS poskytovala formuláře daňových přiznání ke stažení právě ve formátu PDF, což zjednodušovalo práci v daňovém období. Po vzoru IRS začalo více společností dělat totéž, což mělo za následek zvyšování povědomí o existenci PDF. V roce 2008 Mezinárodní organizace pro standardizaci (ISO) formát PDF standardizovala, což z něj umožnilo stát se otevřeným formátem elektronických dokumentů. [18]

V dnešní době je formát PDF používán celosvětově miliony lidí, napříč průmyslovými odvětvími a stále roste na popularitě. V roce 2015 se zjistilo, že jen na internetu existovalo zhruba 2,5 bilionů PDF souborů. [16]

2.2 Struktura souboru

PDF formát má více funkcí než jen vložení textu, může také soubor zabezpečovat heslem, spouštět JavaScript, přidávat multimediální prvky aj.

Jádro formátu PDF vychází z jazyka zvaného PostScript, avšak bez jeho programovacích funkcí, jako jsou například příkazy podmínek *if* a cyklů *loop*, a některé jiné elementy byly implementovány odlišně. Na rozdíl od programovacího jazyku PostScript je PDF založen na binárním formátu souboru optimalizovaného pro vysoký výkon při jeho prohlížení. Navíc formát PDF obsahuje technologii umožňující uložit více různých částí dokumentu do jednoho souboru za použití komprese. [22]

Obsah souboru PDF jsou tvořeny následujícími prvky: **záhlaví**, **tělo**, **tabulka odkazů**, **závěrečná sekce** [21].

Záhlaví

Záhlaví (*header*) je první řádek souboru PDF skládající se z 5 znaků *%PDF-* následované čísly ve tvaru 1.N, kde *N* je v rozsahu 1-7, které určují jakou verzi PDF soubor využívá. [21]

Tělo

Tělo (*body*) souboru se skládá ze sekvence nepřímých objektů, které představují obsah dokumentu. Objekty představují součásti dokumentu, jsou to textové proudy, fonty, obrázky, multimediální prvky [22]. Používá se k uložení všech dat dokumentu, které se uživateli následně zobrazí [20].

Tabulka odkazů

Tabulka odkazů (*cross-reference table*) obsahuje informace umožňující náhodný přístup k nepřímým objektům v souboru, k nalezení konkrétního objektu, tedy není třeba číst celý soubor. Obsahuje odkazy na všechny objekty v dokumentu. [21]

Každá sekce odkazu začíná řádkem obsahující slovo *xref*, kdy následuje jeden nebo víc podsekcí odkazů, které mohou být v libovolném pořadí [22]. Podsekce začíná dvojící čísel oddělených od sebe mezerníkem, první číslo odpovídá číslu objektu (ID) a druhé číslo uvádí počet objektů v aktuální podsekci. Každý objekt představuje jednu položku v tabulce odkazů dlouhou vždy 20 B. Prvních 10 B značí posun objektu od začátku dokumentu PDF k danému objektu. Následuje pětimístné generační (unikátní) číslo objektu v dané podsekci doplněné písmenem „f“ nebo „n“ oddělené mezerou. Existují totiž dva druhy záznamů odkazů: pro volný záznam – písmeno „f“; pro používaný záznam – písmeno „n“ [20]. Je pravidlem, že první záznam v tabulce odkazů má ID 0, musí být vždy volný a jeho generační číslo náleží hodnotě 65535 (jedná se o hlavičku celé tabulky odkazů), poslední záznam v tabulce je také volný (konec tabulky odkazů) a odkazuje zpět na objekt ID 0. Kromě objektu ID 0 musí mít všechny objekty v tabulce první generační číslo rovné 0. Pokud je objekt vymazán, jeho záznam se

v tabulce označí jako volný, číslo generování se poté zvýší o hodnotu 1, aby se následující generační číslo nekrylo s minulým generačním číslem. Maximální počet generací objektů je roven 65535. [22]

Závěrečná sekce

Závěrečná sekce (*trailer*) umožňuje prohlížeči PDF souborů rychle najít tabulku odkazů a určité speciální objekty. Odkazuje na pozici, na které začíná tabulka odkazů [21]. Daný prohlížeč by měl začít číst PDF od konce souboru. Poslední řádek souboru musí vždy obsahovat značku konce souboru *%%EOF*, před níž se nacházejí dva řádky, kde na prvním je řetězec *startxref*, který určuje přesun od začátku souboru k tabulce odkazů vyjádřený hodnotou nacházející se v druhém řádku. Před řádkem *startxref* se nachází slovník uzavřeného do dvojice lomenými závorek (<<…>>). Slovník má následující strukturu [20]:

Size [integer]: Určuje celkový počet položek v tabulce odkazů.

Prev [integer]: Definuje posun od začátku souboru k předchozímu oddílu odkazu.

Root [slovník]: Určuje referenční objekt pro objekt katalogů dokumentů.

Encrypt [slovník]: Určuje šifrovací slovník dokumentu.

Info [slovník]: Specifikuje referenční objekt pro informační slovník souboru.

ID [pole]: Povinná, pokud je přítomná položka *Encrypt*. Určuje pole dvoubajtových nešifrovaných řetězců, které tvoří identifikátor souboru. [22]

2.3 Objekty

V PDF formátu mohou být objekty označeny takovým způsobem, aby na ně bylo možné odkazovat pomocí jiného objektu. Takto označený objekt se nazývá nepřímý. [22] Soubor PDF obsahuje osm základních typů objektů:

- **Booleovské (logické) hodnoty (Boolean)** – představují hodnotu vyjádřenou pomocí klíčových slov *true* nebo *false*. Bývají zejména využívány v polích a slovnících.
- **Čísla (Numbers)** – PDF rozlišuje dva základní typy číselných objektů: *integer* a *real*. Objekt *integer* představuje matematická celá čísla. *Real* prezentuje matematická reálná čísla s omezeným rozsahem. [22]
- **Řetězce (Strings)** – *String* objekt se skládá z řady bajtů hodnot obklopenými jednoduchými nebo lomenými závorkami. Jakýkoliv znak může být reprezentován znaky ASCII nebo osmičkovou či hexadecimální hodnotou. Řetězce mohou být maximálně 65535 bajtů dlouhé. [20]
- **Jména (Names)** – *Name object* je reprezentován posloupností znaků ACII, s určitými výjimkami speciálních znaků, kterému musí předcházet lomítko. Lomítko však není součástí jména, je to pouze předpona indikující, že se jedná o objekt jméno. Alternativní zápis může být také v hexadecimální formě, kde

se místo lomítka před názvem používá hashtag. Délka jména je maximálně 127 bajtů. [20]

- **Pole (Arrays)** – Pole je reprezentováno jako jednorozměrná sekvence objektů uspořádaných za sebou, které mohou být různých typů (může obsahovat slovníky, řetězce, čísla nebo další pole) [20]. V PDF sice nejsou podporovány vícerozměrné pole, ale pole vyšších rozměrů lze sestrojit vnořením pole do jiného jednorozměrného pole jakožto jeho prvek. Pole může mít také nula prvků. Pole se vyznačuje hranatými závorkami. [22]
- **Slovníky (Dictionaries**) – Slovník je reprezentován jako asociativní tabulka obsahující páry *klíč*, *hodnota*. Klíčem je vždy název objektu a hodnotou může být libovolný objekt. Maximální počet položek ve slovníku je 4096 hesel. Slovník se implementuje pomocí dvojice lomených závorek. [20]
- **Streamy (Streams)** – Objekt *streamu* je stejně jako řetězec posloupností bajtů, ale může mít neomezenou délku, z toho důvodu jsou obrázky a multimédia reprezentovány jako proudy. *Stream* se skládá ze slovníku, za nímž následuje posloupnost bajtů v závorce mezi klíčovými slovy (každé na novém řádku) *stream* a *endstream*. Všechny streamy jsou zpravidla nepřímé objekty a slovník musí být vždy přímým objektem. Slovník zde určuje přesný počet bajtů ve streamu. [22]
- **Prázdný (null) objekt** – má typ a hodnotu, které se nerovnají typu ani hodnotě jiného objektu. [22]

Nepřímé objekty

Jakýkoli objekt je možné označit jako objekt nepřímý v PDF souboru. Tím získá objekt jedinečný identifikátor objektu, podle kterého se mnohou ostatní objekty na něj odkazovat (např. jako prvek pole, hodnota slovníkové položky). Odkazy na nepřímé objekty se nacházejí v tabulce odkazů a označeny klíčovým slovem *xref*. [20]

2.4 Struktura dokumentu

Dokument PDF se skládá z objektů nacházejících se v těle souboru, jedná se především o slovníky. Každá stránka dokumentu je definována slovníkem obsahující odkazy na obsah stránky a další atributy, jako je například miniatura (náhled) stránky. Objekty stránek (slovníky) jsou vzájemně propojeny a tvoří tím takzvaný strom stránek *page tree*, který je deklarován s nepřímým odkazem v katalogu dokumentů. Katalog dokumentů odkazuje konkrétně na pět sekcí – *page tree*, *outline hiearchy*, *article threads*, *named destinations*, *interactive form*. [22]

Katalog dokumentů

Kořenem hierarchie objektů dokumentu PDF je katalogový slovník, který se nachází v závěrečné sekci (trailer) souboru v sekci */Root*. Obsahuje odkazy na jiné objekty, které definují obsah dokumentu. Definuje, jak bude dokument vypadat a jak bude zobrazen na obrazovce (např. zda se mají zobrazit miniatury stránek a obsah dokumentu). [20]

Strom stránek

Stránky dokumentu jsou přístupné pomocí stromu stránek, který definuje, v jakém pořadí se stránky v dokumentu nacházejí. Pomocí stromové struktury mohou prohlížeče PDF dokumentů rychle otevřít soubor obsahující až několik tisíc stránek. Strom obsahuje uzly, které představují stránky dokumentu PDF. Rozlišujeme dva typy uzlů: mezilehlé uzly (uzly stromu stránek) a listové uzly (objekty stránek). Každý uzel ve stromu stránek musí obsahovat následující položky [20]:

- **Type** [jméno] – Specifikuje typ objektu PDF (stránky)
- **Parent** [slovník] – Určuje bezprostředně svého rodiče.
- **Kids** [pole] – Definuje všechny bezprostřední potomky tohoto uzlu.
- **Count** [integer] – Určuje počet objektů stránek, které jsou potomky tohoto uzlu. [22]

Listy stromu stránek neboli objekty stránek, jsou slovníky definující atributy jedné jediné stránky [20].

2.5 Obsahové streamy

Obsahové streamy PDF souboru jsou primárním prostředkem, který obsahuje instrukce popisující vzhled stránky nebo jiných grafických elementů (formulářů, fontů, vzorů), které mají být na stránce vykresleny. Tyto instrukce jsou reprezentovány ve formě objektů, avšak jsou pojmově odlišné oproti objektům, které definují strukturu dokumentu, jsou popsány samostatně. Zatímco dokument je statická datová struktura s náhodným přístupem, objekty (instrukce) v obsahových streamech se interpretují dynamicky. Každá jedna stránka musí být reprezentována alespoň jedním obsahovým proudem. [22]

Grafické operátory

Grafické operátory jsou klíčové pro popis vzhledu stránek, které jsou zobrazeny na výstupním zařízení (zobrazovací aplikace, tiskárna). Grafické operátory je možné rozdělit na šest hlavních skupin [21]:

- **Operátory grafického stavu** manipulují s datovou strukturou, která se nazývá grafický stav, v němž se provádí operace ostatních grafických operátorů. V této datové struktuře jsou uloženy informace o aktuální transformační matici (CTM), která sleduje souřadnice uživatelského prostoru v obsahovém streamu na

souřadnice výstupního zařízení. Dále obsahuje aktuálně používanou cestu, aktuální ořezovou cestu a další parametry operátorů malování. [22]

- **Operátory konstrukce cesty** určují cesty, které definují trajektorie čar a jejich tvar různých typů cest. Součástí jsou také operátory zajišťující vytvoření nové cesty, přidání úsečky a křivek nebo také jejich uzavření. [21]
- **Operátory malování cesty** se zaměřují na vyplnění cesty barvou, malování tahů podél ní nebo použití této cesty jako hraničního oříznutí. [22]
- **Ostatní operátory malování** provádí např. geometricky definované stínování, vzorkování obrázků. [22]
- **Textové operátory** vybírají a zobrazují glyfy znaků písem. V PDF jsou glyfy považovány za grafické útvary, a proto mnoho textových operátorů spadá pod operátory malování a grafického stavu. [22]
- **Operátory označeného obsahu** spojují logické informace vyšší úrovně s objekty ve streamu obsahu. Tyto operátory nemají přímo vliv na vykreslený vliv vzhledu (mohou ale určovat, zda se daný obsah má vůbec zobrazit), jsou ale užitečnými pro aplikace, které využívají formát PDF pro výměnu dokumentů. [22]

2.6 Motivace použití

Konzistentní: zobrazuje přesně stejný obsah a jeho rozvržení bez ohledu na použitém softwaru, operačním systému nebo zařízení. Ukládá obsah bez ztráty kvality a přebytečného množství dat (vhodný pro přenos pomocí e-mailu). [19]

Všestranný: umožňuje sjednotit různé typy obsahu jako je text, obrázky, 3D modely, vektorovou grafiku, zvuk, video, odkazy anebo tlačítka. Tento obsah lze v rámci souboru PDF uložit například jako prezentaci, zprávu nebo dokument. [19]

Bezpečnost: nabízí zabezpečení obsahu nebo celého dokumentu pomocí například hesla, digitální podpisy anebo vodoznaky. [19]

2.7 Omezení PDF

Cílem formátu PDF je zachovat a chránit obsah a rozvržení dokumentu, bez ohledu na použitou platformu nebo software. Kvůli tomu je těžké soubory PDF upravit a v některých případech z nich i složitě extrahovat informace. [19]

Všechny soubory PDF nejsou stejné. Různé typy PDF vyžadují odlišené způsoby práce, například při extrahování nebo vyhledávání informací. [19]

3. PROGRAMOVÉ KNIHOVNY

Tato kapitola popisuje programové knihovny, které byly použity při realizaci textového editoru.

Python-Markdown

Knihovna Python-Markdown je téměř totožná Python implementace Markdownu Johna Grubera. Existují pouze velmi malé rozdíly mezi touto knihovnou a referenční implementací [9]. Knihovna byla především vytvořena pro použití v prostředích webových serverů, kde dochází překladu Markdown na HTML. Kromě základní syntaxe Markdown je dále knihovna rozšířena o funkce mezinárodního vstupu, kdy knihovna dokáže přijímat vstupy v jakémkoli jazyce podporovaném Unicode, o různá rozšíření základní syntaxe (viz. dokumentace), o možnost volby výstupního formátu mezi HTML nebo XTHML, anebo o rozhraní příkazového řádku, kdy lze pohodlně využít skript příkazového řádku. [10]

Pravidla Markdownu Johna Grubera jasně stanovují, že pokud se text např. skládá z více odstavců, musí být každý odstavec odřádkován minimálně 3 mezery nebo tabulátorem pro správné formátování. Toto vývojáři knihovny Python-Markdown považují za chybu, tedy toto pravidlo nevynucují [10]. Knihovna dále disponuje např. rozšířením *Sane List Extansion*, které umožňuje míchání typů seznamů, kdy uspořádaný seznam nebude dále pokračovat (překrývat neuspořádaný seznam), pokud je v kombinaci s neuspořádaným seznamem. [11]

xhtml2pdf

Knihovna *xhtml2pdf* umožnuje uživatelům převod obsahu HTML na dokument PDF pomocí Python, ReportLab Toolkit, html5lib a PyPDF3. Knihovna je kompletně napsána v jazyce Python, tedy je nezávislá na použité platformě [12]. Převod mezi formáty HTML a PDF zařizuje automatizované řízení toku dat, jako je udržování textu pohromadě nebo stránkování. Knihovna lze použít pro samostatný převod HTML obsahu nebo společně s knihovnou Django. Je možné použit také nástroje příkazového řádku knihovny (skripty příkazového řádku). [13]

PyQt5

Qt vychází ze sady multiplatformních a vývojových C++ knihoven, které zahrnují tvorbu grafických prostředí, databází SQL, XML, uživatelských a aplikačních nastavení, polohovací a lokalizační služby a mnoho dalšího [14]. Pyt5 je sada více než 1000 tříd rozšiřujících modulů, které slouží jako alternativa jazyka C++ pro vývoj aplikací na všech podporových platformách včetně Android a iOS [15]. Je jedním z nejpoužívanějších a nejpopulárnějších frameworků pro tvoření grafických prostředí GUI, díky své unikátnosti, velké variabilitě a širokému nastavení vzhledu jednotlivých prvků. Designer

QtDesigner, který je součástí balíčku PyQt5, umožňuje jednoduše vytvářet kostru grafického rozhraní přetažením jednotlivých komponent do oken programu bez nutnosti zdlouhavého psaní layoutu aplikace v kódu. Použití frameworku PyQt5 spadá pod license Riverbank Commercia License a GPL v3. [14]

Beautiful Soup 4

Beautiful Soup je knihovnou Python, která usnadňuje výčet informací, dat z webových stránek nebo dokumentů HTML a XML. Disponuje jednoduchým a přehledným rozhraním pro procházení těchto stránek či dokumentů a vyhledávání konkrétních tagů. Poskytuje různé metody pro práci s extrahovanými daty z tagů a atributů, včetně možnosti filtrování dat na základě předem stanovených podmínek uživatele (programátora) a kombinování extrahovaných dat a jejich úpravu. Knihovna Beautiful Soup je užitečným nástrojem pro *data mining*, *web scraping* nebo další podobné práce vyžadující práci s daty webových stránek a dokumentů. [23]

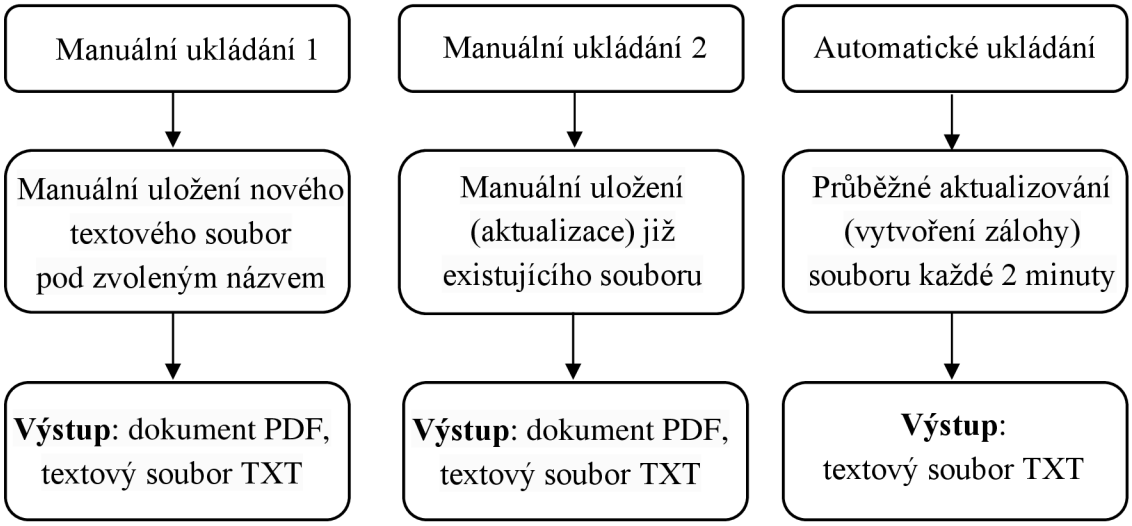
4. TVORBA TEXTOVÉHO EDITORU

4.1 Vize textového editoru

Cílem bakalářské práce je vytvořit základní textový editor, který bude přizpůsobený pro seniory ve věkové kategorii 90 let a více. Textový editor proto musí být lehce ovladatelný, bude podporovat pouze základní formátování pomocí značkovacího jazyka Markdown a přívětivou formu správy dokumentů. Celá aplikace je pak zhotovena v jazyce Python.

Značkovací jazyk Markdown zpřístupňuje velmi jednoduché formátování textu, obrázků, seznamů nebo například tabulek pomocí speciálních znaků. Pro seniora je poměrně snadné se naučit pouze sadu nejpoužívanějších znaků (pro formátování nadpisů, zvýraznění textu, popřípadě seznamů) a ty opakovaně aplikovat. Jazyk Markdown má ale jednu zásadní nevýhodu, výsledek nelze vidět okamžitě. Požadované změny formátování se projeví až po vykreslení (většinou v HTML), v případě této práce až v PDF dokumentu, což může být pro uživatele matoucí. Proto byly vytvořeny formátovací funkce editoru, které dokáží formátovat text okamžitě, bez použití značek Markdown. Tyto funkce ale stále pracují s jazykem Markdown na svém pozadí. Text z textového pole editoru je pomocí formátovacích funkcí do PDF souboru (jako hlavní výstup textového editoru) vykreslen ve své přesné podobě, avšak do textového souboru .txt je uložen ve formě prostého textu s kombinací značek jazyka Markdown. Znalý uživatel (např. asistent seniora) jazyka Markdown může upravovat tento textový soubor přímo pomocí značek bez nutnosti otevřít tento soubor v textovém editoru.

O správu dokumentů a jejich archivace se starají 3 funkce. Jedna z funkcí obstarává automatické průběžné ukládání, popřípadě vytváření záloh. Pokud uživatel manuálně neuložil soubor, je mu vytvořena záloha. Tato záloha je aktivní do té doby, až uživatel manuálně soubor neuloží nebo neotevře nějaký jiný soubor. Jestli však uživatel uloží dokument pod konkrétním názvem nebo otevře nějaký jiný dokument v počítače, je průběžně aktualizován právě tento dokument. Průběžné ukládání probíhá každé 2 minuty. Manuální ukládání dokumentů uživatelem je řešeno dvěma dalšími funkcemi, které ukládají obsah textového pole do formátu PDF a textové souboru .txt. Automatické průběžné ukládání obsah textového pole ukládá pouze do textového souboru .txt. První funkce manuálního ukládání zprostředkovává uložení nového souboru pod názvem zadaným uživatelem. Druhá funkce manuálního ukládá (přepisuje) pouze soubory, které už předtím byly uživatelem uloženy nebo otevřeny v textovém editoru. Všechny funkce spravující dokumenty a jejich archivace ukládají dokumenty do domovské složky /home.



Obr. 4.1: Princip fungování funkcí pro správu a archivaci souborů

Ke zvýšení uživatelské přívětivosti byl implementován hlasový asistent, vícejazyčný překlad aplikace nebo například možnost přibližování a oddalování textového pole. Hlasový asistent zajišťuje snadnější orientaci v programu pro uživatele, především se zrakovým postižením. Umožňuje zvukovou signalizaci při přejetí kurzoru myši na tlačítko nebo štítek programu. Po přejetí myši se zpravidla ozve zvuk oznamující název tohoto tlačítka či štítku. Vícejazyčný překlad aplikace umožňuje kompletně přeložit aplikaci do jiného jazyka. Momentálně podporuje pouze dva základní jazyky češtinu a angličtinu. Hlasový asistent a vícejazyčný překlad využívají souborů JSON, kde jsou uloženy cesty k nahrávkám TTS nebo kompletní překlad aplikace.

Textový editor se skládá ze dvou hlavních oken – úvodní okno a hlavní okno. Úvodní okno se zobrazuje při startu aplikaci, kde uživatel má možnost si buďto načíst jeden z naposledy uložených textových dokumentů (soubory) nebo vytvořit textový dokument nový. Po potvrzení jedné z možností je uživatel přesměrován do hlavního okna, kde se soubor načte nebo vytvoří nový. Hlavní okno je jádrem celého textového editoru, které slouží pro práci s textovým dokumentem, formátování textu, ukládání souborů a obsahuje funkce ulehčující tuto práci. Tyto dvě okna nejsou samostatná, ale přímo na sebe navazují. Pro tento účel byla vytvořena třída MoreWindows zděděná ze třídy QStackedWidget, která umožňuje vkládat více oken, přesněji více widgetů, pod jedno konkrétní. Vloženým oknům tato třída následně přiděluje specifický index, podle něhož může dané okno zobrazovat nebo s ním dělat určité operace. Toto okno třídy MoreWindows je následně zobrazeno v režimu celé obrazovky (fullscreen), aby se uživatel nemohl překliknout do okna aplikace *srun*, zprovozňující spouštění aplikací operačního systému včetně textového editoru, a tím se tak ztratit mezi zapnutými okny. Textový editor také disponuje skupinou dialogů pro načtení textových dokumentů a obrázků, anebo pro jejich uložení.

Celková vizáž textového editoru je koncipována do barevného podání odstínů šedi k zachování přehlednosti i pro uživatele se zrakovým postižením. Jednotlivé tlačítka, štítky a ostatní prvky byly designována v co nejpřijatelnější velikosti, aby je mohl přečíst i starší člověk. Tomu odpovídají i funkce pro ovládání textového editoru, které jsou intuitivní i pro neznalého uživatele.

Úvodní okno

Úvodní okno bylo navrženo z důvodu jednoduchého načítání již dříve uložených textových dokumentů uživatelem. Okno obsahuje tlačítko pro vytvoření nového textového souboru a sadu štítku, které odpovídají textovým souborům uložených uživatelem v textovém editoru. Tyto štítky obsahují název souboru a datum posledního uložení a jsou uloženy sestupně právě podle tohoto data. Štítky jsou prvky typu QLabel, kterým bylo třeba nastavit funkci kliknutí, která otevře v textovém editoru, přesněji v textovém poli, daný soubor nebo vytvoří nový. Prvek QLabel byl upřednostněn oproti tlačítkům QPushButton díky vyšší variabilitě nastavení textu. Text bylo potřeba naformátovat pomocí HTML, kdy název souboru byl zarovnán na levou stranu prvku (štítku) a datum poslední změny na pravou stranu, což u tlačítek QPushButton není možné.

Sada štítku je importována z textového souboru *filesToHomePage.txt* do prvku QScrollArea ve svislé poloze, který dokáže v případě většího počtu štítku mezi nimi rolovat. V tomto textovém souboru dvojice řádku odpovídá jednomu zobrazovaném štítku, kdy lichý řádek značí cestu daného souboru a sudý řádek datum jeho posledního uložení. Kvůli tomu, že textový editor ukládá všechny soubory do jednotné složky /home, by nebylo třeba ukládat celé cesty souborů, ale pouze jejich názvy. Díky tomu, že je možné otevírat textové dokumenty z různých složek v systému, je lepší nápad uchovávat i jejich cesty, aby se zachovala celková variabilita programu. Kompletní cesty souborů se už ale nevyskytují na štítcích v úvodním okně. Ty jsou vyfiltrované takovým způsobem, aby se zobrazoval pouze název souboru.

Obr. 4.2: Úvodní okno textového editoru

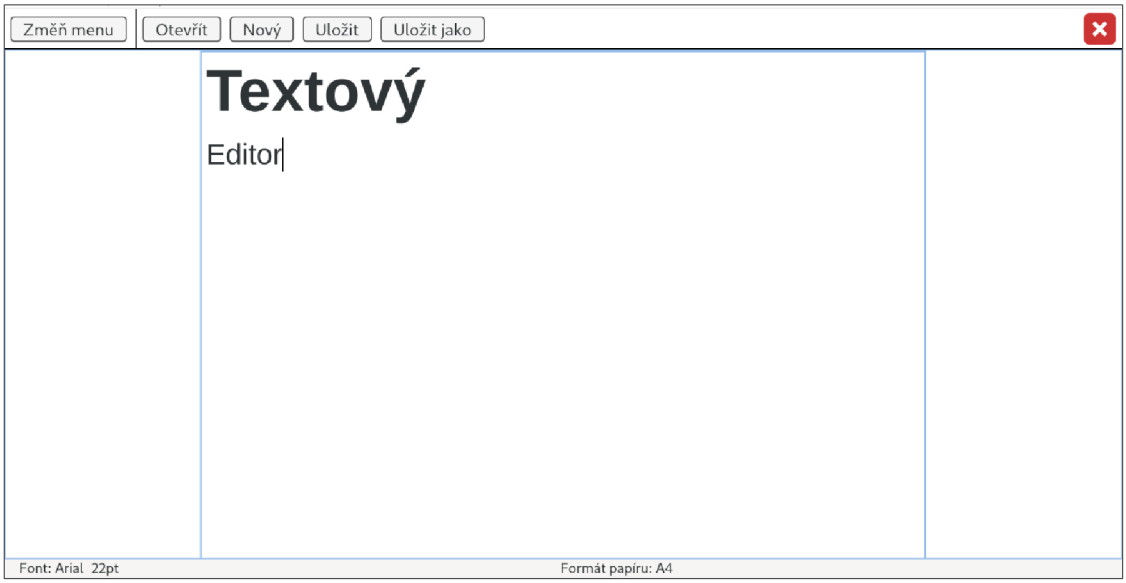
Po kliknutí na příslušný štítek se spustí funkce *openFileInTextEditor*, která obsahuje dvojici parametrů *path* a *objectName*. Pokud proměnná *path* neobsahuje žádnou hodnotu,

tedy *None*, pak je vytvořen nový soubor v textovém editoru. Tato část funkce odpovídá implementaci tlačítka „Vytvořit nový soubor“ v úvodním okně. V ostatních případech proměnná *path* obsahuje konkrétní cestu daného souboru. Následuje důležitá část, při které se kontroluje existence souboru pod touto konkrétní cestou. Jestli soubor existuje, textový editor tento textový soubor načte. Pokud však ne, program vyhledá štítek odpovídající cestě souboru pomocí proměnné *objectName*, tento štítek vymaže, zobrazí varovné okno, které upozorní uživatele, že nebylo možné tento soubor otevřít, protože nebyl nalezen. Následně také vymaže záznam souboru (cesty a data poslední změny) z textového souboru *filesToHomePage.txt*.

O vytváření záznamů uložených souborech z textového editoru (soubor *filesToHomePage.txt*) se stará funkce *saveLastFileName()*, která je vyvolána vždy při ukládání textových souborů. Tato funkce ukládá záznamy do souboru principem, aby záznamy byly seřazeny dle data uložení od nejnovějšího po nejstarší.

Hlavní okno

Hlavní okno textového editoru se skládá z navigačního panelu, textového pole a stavového řádku. Navigační panel zprostředkovává kompletní ovládání textového editoru, jako je práce se souborem (dokumentem), formátování textu nebo základní operace s textem (vkládání, kopírování atd.). Z důvodu, že v aplikaci byly implementovány funkce podporující formátování textu v reálném čase (ne pouze po vykreslení do PDF) běžící na značkovacím jazyce Markdown, bylo vybráno textové pole třídy QTextEdit, která umožňuje formátování textu, vkládání obrázků, listů, tabulek aj. Pro standardní Markdown syntaxi by plně dostačovalo textové pole třídy QPlainTextEdit podporující pouze prostý text. Textové pole je dále vloženo do grafického zobrazení, které umožňuje toto textové pole přibližovat nebo oddalovat. Stavový řádek nese informace o nastavení textového dokumentu – font, velikost písma a formát papíru.



Obr. 4.3: Hlavní okno textového editoru

4.2 Grafické rozhraní

Pro tvorbu grafického rozhraní textového editoru byla zvolena knihovna PyQt5. Druhou alternativou byla známá knihovna Tkinter, která je již součástí Pythonu. Knihovna PyQt5 byla vybrána na základě její univerzálnosti, komplexnosti, a především kvůli modernosti a variabilitě prvků (widgetů), kde Tkinter zaostává. U Tkinter je omezená možnost úprav, variabilita a pružnost prvků, tedy programy postavené na této knihovně nejsou většinou složité a mohou vypadat zastarale, zde knihovna PyQt5 naopak vyniká. Na druhou stranu výhodou Tkinteru je jeho rychlost, především díky své jednoduchosti. PyQt5 je o něco pomalejší, ale u programu jako je textový editor není tento rozdíl znatelný. Další výhodou PyQt5, především pro designery, je QtDesigner, který je nainstalován společně s knihovnou. Pomocí QtDesigneru lze jednoduše vytvářet grafická rozhraní přetažením potřebných widgetů do okna programu, nastavovat jejich vlastnosti, zarovnání nebo akce a popřípadě tyto prvky upravovat pomocí šablon stylů QSS (stejný způsob jako CSS). Výhodou QtDesigneru je také, že výsledek práce lze vidět okamžitě, kdežto u manuálního psaní kódu grafického prostředí lze vidět výsledek až po spuštění programu. Téměř všechny okna grafického rozhraní programu byla tvořena pomocí zmiňované QtDesigneru, díky němuž bylo ušetřeno mnoho času, který byl využit v jiných částech programování textového editoru.

Výstupem QtDesigneru není Python soubor se syntaxí PyQt5, ale souboru s koncovkou .ui, ve kterém jsou uložený všechny konfigurační vlastnosti a design daného okna GUI pomocí jazyka XML (velmi podobná syntaxe jako u jazyka HTML). Import tohoto XML soubor je možný pomocí metody *loadUi()* v konstruktoru konkrétní třídy, která specifikuje její vzhled. Možným řešením je také soubor *ui* převést rovnou na python

kód pomocí některého z konvertorů nebo QtDesigneru, který tuto funkci podporuje, a tento kód přímo zasadit do konstruktoru dané třídy.

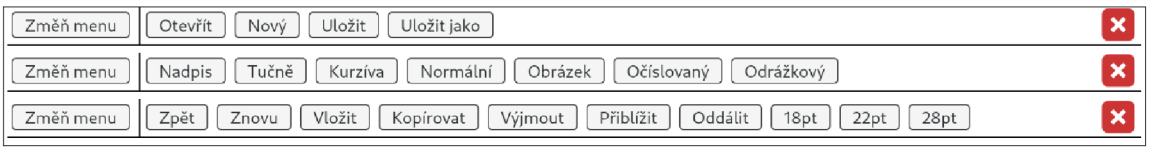
Výpis 4.1: Ukázka konfiguračního souboru ui grafické rozhraní (jazyk XML)

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>3436</width>
    <height>762</height>
   </rect>
  </property>
        ...             ...             ...             ...
<property name="styleSheet">
   <string notr="true">/* Style for first widget - main_menu_widget*/

#main_menu_widget {

        background-color: white;
        border-bottom: 2px solid black;
}
        ...             ...             ...             ...
   </string>
</property>
  <widget class="QWidget" name="centralwidget">
   <layout class="QGridLayout" name="gridLayout">
    <property name="leftMargin">
     <number>0</number>
    </property>
        ...             ...             ...             ...
```

4.3 Navigační panel

Pro co nejjednodušší ovladatelnost a nejlepší uživatelskou přívětivost vznikl jednořádkový navigační panel. Zpravidla jednořádkový navigační panel nemůže disponovat příliš obsáhlou nabídkou. To je ale vyřešeno tlačítkem „Změň menu“, které přepíná jednotlivé sekce nabídky – práce se souborem, formátování, jednoduchá práce s textem. Toto tlačítko funguje na principu stavového automat, který přepíná mezi 3 hlavními stavy, resp. mezi 3 nabídkami. Stavový automat je tvořen funkcí *changeNavbar()* a konkrétně se stará o zobrazení jedné specifické nabídky v podobě třídy QWidget a skrytí všech ostatních. Výchozí stav (první) zobrazuje pouze nabídku (*file_widget*) pro práci se souborem (dokumentem). Po kliknutí na tlačítko „Změň menu“ se uživateli zobrazí již jiná nabídka pro formátování textu (*format_widget*). Pokud tedy uživatel chce z druhého stavu přejít zpět na první výchozí, musí se nabídkou proklikat.

Třetí stav zobrazuje základní možnosti práce s textem (*edit_widget*) a poslední zobrazuje pouze nabídku pro možný překlad aplikace. Poslední stav je pouze provizorní, protože tato nabídku bude v budoucnu smazána a přemístěna do aplikace *srun* zprovozňující spouštění všech ostatních aplikací, včetně textového editoru, ve které bude možné nastavit výchozí jazyk nebo vypnout, či zapnout hlasového asistenta. Kvůli tomu, že textový editor je zobrazen přes celou obrazovku (fullscreen), navigační bar obsahuje v pravé části vypínací tlačítko.



Obr. 4.4: Navigační panel – tři základní stavy

Jednotlivé nabídky jsou naplněny skupinou tlačítek vykonávají určitou funkci. Tyto tlačítka mají jednotný styl, který platí i pro tlačítka v úvodním okně. Celá aplikace je designovaná v barvách šedi, tlačítka nejsou výjimkou. Jejich pozadí je mírně šedé, aby je bylo možné rozeznat od bílého navigačního panelu, a po přejetí myší se ještě více ztmaví. Docílením tohoto stylu bylo pomocí jazyka QSS (Qt Style Sheet), který funguje na velmi podobném principu jako jazyk CSS.

Výpis 4.2: Ukázka stylů QSS pro nastavení design tlačítek navigačního panelu

```
#main_menu_widget QPushButton {
        background-color: rgb(245, 245, 245);
        border-radius: 7;
        border: 2px solid #464646;
        padding: 2 20 2 20px;
}

#main_menu_widget QPushButton:hover {
        background-color: rgb(204, 204, 204);
}
```

Jednotlivým tlačítkům byly také přiřazeny klávesové zkratky umožňující rychlejší spouštění jejich funkcí a celkově svižnější manipulaci s textovým editorem. K vytvoření klávesových zkratek byla použita třída QShortcut a klávesové kombinace třída QKeySequence.

Výpis 4.3: Ukázka kódu pro nastavení klávesových zkratek

```
self.btnOpenFile.clicked.connect(self.openFile)
self.shortcut = QShortcut(QKeySequence("Ctrl+O"), self)
self.shortcut.activated.connect(self.openFile).
```

4.4 Překlad Markdown do PDF

Způsobů, jak přeložit značkovací jazyk Markdown do textového formátu PDF je vícero. Prostý text napsaný v Markdown lze přeložit buďto přímo do PDF nebo s mezikrokem (doporučený způsob), kdy Markdown je přeložen do HTML a následně do PDF. První způsob překladu Markdown přímo do PDF je složitější a náročnější na paměť zařízení, protože překlad využívající například knihovnu Pandoc potřebuje doinstalovat nějaký z dostupných LaTeX enginů, které zabírají poměrně dost místa. Samozřejmě existují zredukované (zabírají mnohem méně místa) knihovny, které se specializují pouze na překlad mezi Markdown a PDF, ale ty nejsou většinou volně dostupné. Pro tuto práci byl vybrán druhý způsob, kdy je prostý Markdown text převeden do HTML a následně přeložen do PDF. Tento způsob je doporučený, protože jazyk Markdown byl primárně vytvořen pro jednoduché formátování a překlad textu do HTML.

Pro převod textu Markdown do HTML byla vybrána knihovna pro *Python-Markdown*, která je implementací Markdownu Johna Grubera s několika velmi malými rozdíly, kde syntaxe a překlad je stejný jako tomu bylo popsáno v teorii o jazyce Markdown. Implementace překladu knihovny a jejího rozšíření je velmi jednoduché, bez žádných komplikací – `markdown.markdown(markdownText)`.

Pokud se jedná o převod z HTML do PDF, zde jsou hlavní dvě možnosti, jak tento překlad uskutečnit. Buď pomocí nástrojů příkazového řádku *wkhtmltopdf* nebo pomocí python knihovny *xhtml2pdf*. Pro prvotní řešení překladu z HTML do PDF bylo využito nástrojů příkazového řádku *wkhtmltopdf*. Převod na PDF pomocí *wkhtmltopdf* je sice jednodušší než *xhtml2pdf*, ale obsahuje zásadní chybu, a to nemožnost převést přesnou velikost fontu z HTML do PDF. Tedy velikost textu ve formátu HTML nikdy neodpovídá velikosti fontu v PDF, většinou je text o 1-4 bodů větší, podle použité velikosti textu u HTML. K řešení tohoto problému by bylo možné udělat slovník, který by obsahoval velikosti fontu v HTML jako klíče slovníku a na druhé straně by byly hodnoty odpovídající velikosti fontu pro PDF. Toto řešení lze ale použít pouze pro některé konkrétní fonty a tyto slovníky by musely být vytvořeny pro každý font zvlášť, což není praktické. Kvůli tomuto bylo třeba přejít na knihovnu *xhtmltopdf*, která jakékoliv problémy ohledně překladu velikosti fontu z HTML do PDF vyřešila.

Knihovna *xhtml2pdf* se využívá většinou s Python knihovnou Django pro překlad webových stránek. Použít tuto knihovnu lze bez problému i pro prostý HTML text. O překlad se stará metoda knihovny *pisa*, kde povinnými parametry je zdroj HTML (stránka nebo text) a cesta k PDF souboru pro jeho vytvoření a následný překlad. Nepovinným parametrem je dále použité kódování, které bylo nastaveno ve formátu UTF-8. Pro překlad HTML do formátu PDF byla vytvořena funkce *convert_html_to_pdf()*.

Výpis 4.4: Ukázka kódu pro překlad HTML do PDF pomocí knihovny *xhtml2pdf*

```
def convert_html_to_pdf(self, source_html, output_filename):
    # open output file for writing (truncated binary)
    result_file = open(output_filename, "w+b")

    # convert HTML to PDF
    pisa_status = pisa.CreatePDF(
           source_html,          # the HTML to convert
           dest=result_file,
           encoding="UTF-8")# file handle to recieve result

    # close output file
    result_file.close()           # close output file

    # return False on success and True on errors
    return pisa_status.err
```

Bohužel nastavením nepovinného parametru kódování není vyřešen problém správného vykreslování znaků české diakritiky jako je třeba písmeno „ů“. Toto je nevýhodou knihovny *xhtml2pdf* oproti *wkhtmltopdf*, hlavně pro národnosti, které používají speciální znaky ve svém jazyce. Tuto nepříjemnost lze ale vyřešit doplněním proměnné typu *string* s názvem *style* k přeloženému textu Markdown do HTML, která bude obsahovat CSS styly pro importování externích fontů podporující vykreslení speciálních znaků. Proměnná *style* obsahuje základní elementy webu, jako jsou *doctype*, *html*, *head*, *meta charset*, kde je taktéž nastaveno kódování UTF-8, a v neposlední řádě *style*, kde je řešeno jádro tohoto problému a přidáno vhodné formátování textu do PDF dokumentu. Do tohoto stylu byly vloženy jednotlivé atributy @font-face pro importování jednotlivých fontů. Poté je třeba vložit povinné parametry, pro správné fungování, jako je název fontu (*font-family*) a jeho zdroj, neboli cesta k danému fontu (*src*). Je třeba podotknout, že vložení pouze regulárního fontu například Arial, problém s diakritikou vyřeší, ale vznikne nový problém a to ten, že knihovna *xhtml2pdf* nebude moct přeložit daný text do PDF s využitím zvýraznění fontu (*bold*) nebo kurzívy. Dále je tedy třeba importovat ještě daný font pro *bold*, *italic* a také jejich kombinaci *bold* + *italic*. Pro načtení *bold* fontu je také nutnost nastavit navíc *font-weight* jako *bold* a u *italic* vložit *font-style* jako *italic*, *font-family* je stejný jako u regulárního fontu. Poslední věcí pro správné vykreslování diakritiky v dokumentu PDF je nastavit font pro tělo HTML (body), tedy pro překládaný text, název *font-family* daného importovaného fontu, popřípadě velikost fontu.

Výpis 4.5: Příklad nahrávání různých fontů a jejich nastavení pro překládaný text

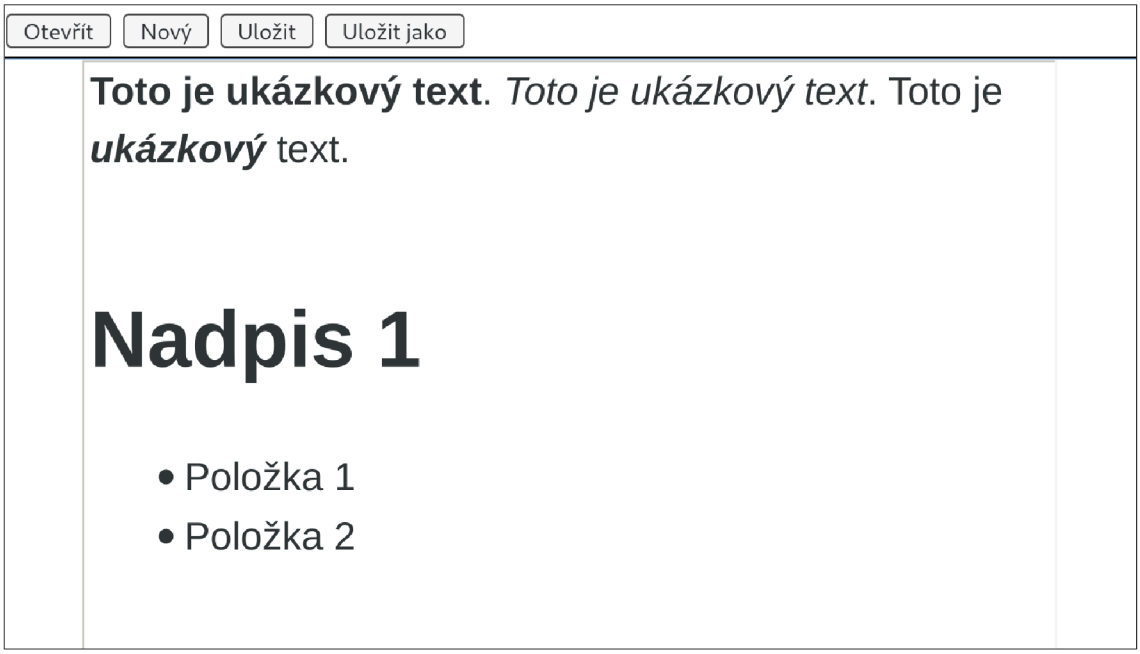
```
@font-face {
    font-family: Arial;
    src: url("Fonts/ArialBoldItalic.ttf");
    font-weight: bold;
    font-style: italic;
```

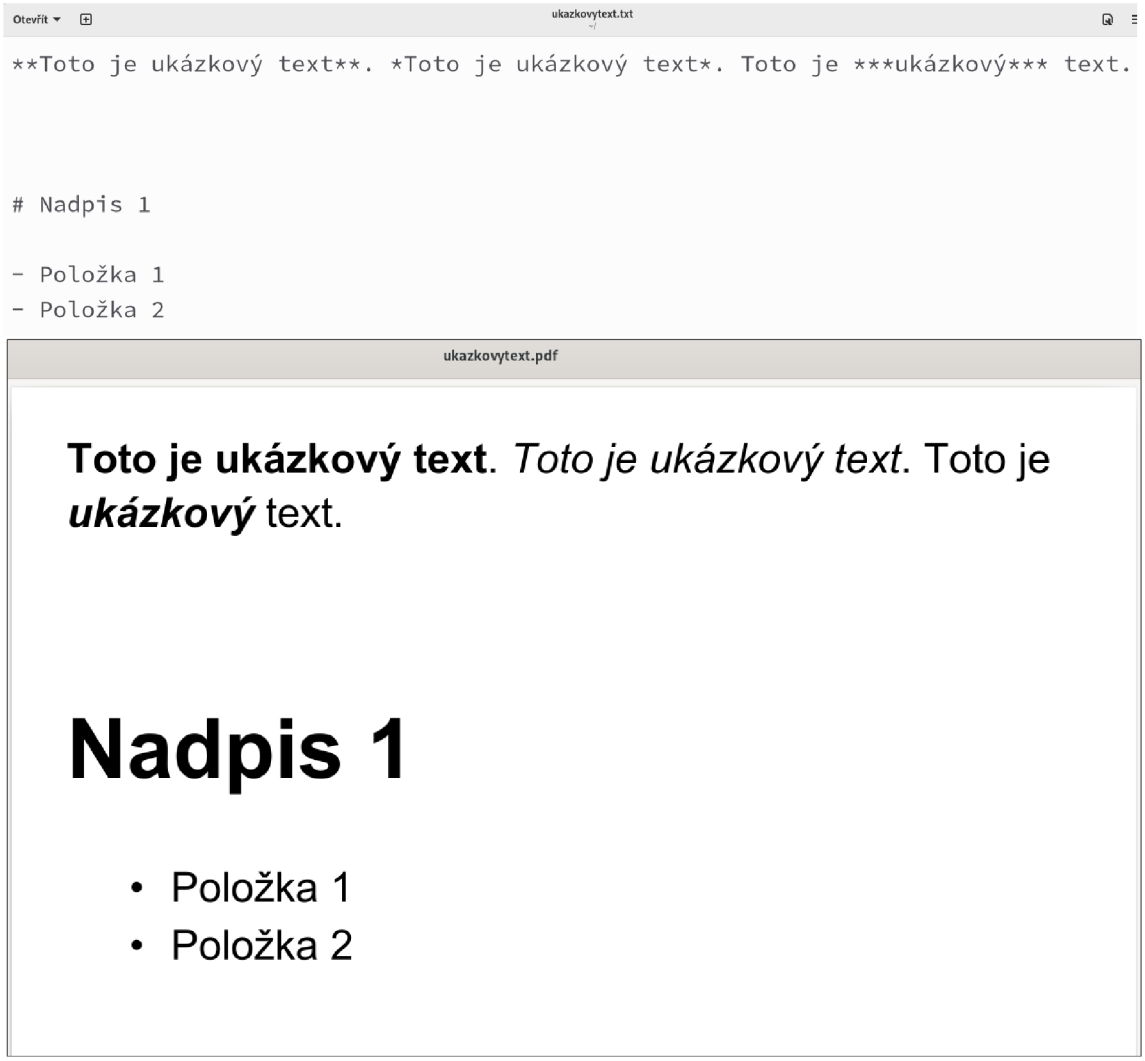
```
    }

    body {
        font-family: Arial;
        font-size: ''' +str(self.textEdit.font().pointSize())+'''pt;
    }
```

Přeložený text z Markdown do HTML je třeba sjednotit s proměnnou style pro správný překlad do PDF. Dále je zapotřebí nastavit cestu k uložení tohoto PDF dokumentu. Tato cesta pro PDF dokument je vytvořena z aktuální používané cesty uložené v proměnné *currentPath*. Protože proměnná *currentPath* obsahuje cestu k textovém souboru, většinou typu *txt* nebo *md*, tak je nutností koncovku této cesty přepsat na .pdf pomocí knihovny os. PDF dokument bude tedy uložen do složky /home pod stejným názvem jako textový soubor *txt*, ale jiného typu. Poté už stačí tyto parametry vložit do funkce *convert_to_hmtl_pdf()* a knihovna *xhtml2pdf* se o překlad daného textu do formátu PDF postará.

Výpis 4.6: Sjednocení přeloženého textu s proměnnou style a nastavení cesty výstupu

```
markdownText = self.markdownWithEmptyLines().replace("file://", "")
markdownCorrectly = self.loadWithEmptyLinesPDF(markdownText)
htmlText = style + "\n<body>\n" + markdown.markdown(markdownCorrectly)
+ "\n</body>\n</html>"
pathNamePDF = os.path.splitext(pathName)[0] + ".pdf"

pisa_status = self.convert_html_to_pdf(htmlText, pathNamePDF)
```





Obr. 4.5: Překlad textu z textového pole editoru do dokumentu PDF a souboru TXT

4.5 Práce se soubory

Vytvoření nového souboru

K vytvoření nového souboru slouží funkce *newFile()*, která je velmi jednoduchá. Pouze vymaže konkrétní cestu pro ukládání souborů, nastaví výchozí formátování textu a bloků a vymaže obsah textového pole.

Otevření souboru

O otevírání souborů se stará funkce editoru *openFile()*, která využívá metoda *getOpenFileName()* třídy QFileDialog umožňující pomocí průzkumníka souborů uživateli daný soubor vyhledat a zjistit jeho cestu. Metoda přímo soubor neotevírá, to je třeba provést manuálně. V metodě *getOpenFileName()* jsou vepsány parametry určující název okna průzkumníku, počáteční složku, ve které je průzkumník spuštěn (/home),

formáty podporovaných souborů k otevření a v neposlední řádě možnosti *options*, kde bylo nutno nastavit QFile.Dialog.DontUseNativeDialog, které přiřazují průzkumníkovi dané práva, pro hledání vyhledávání souborů. Tato možnost je povinností pro operační systémy s Linuxovým jádrem, jinak by se v průzkumníkovi neobjevovaly žádné soubory, pouze jejich složky. Podle cesty souboru, získané z této metody, je soubor otevřen a jeho obsah je přepsán do textového pole v textovém editoru. Následně je cesta tohoto souboru vložena do současné cesty proměnné *currentPath*, obsah textového pole vhodně upraven a vymazán zásobník historie úprav. Pokud cesta k souboru je prázdný string, tedy uživatel nevybral žádný soubor k otevření, dojde k přerušení funkce otevření *openFile()*.

Výpis 4.7: Ukázka kódu funkce pro otevření souboru

```
def openFile(self):
    fname = QFileDialog.getOpenFileName(self,
self.translate[f"stext_{self.btnChangeLanguage.text()}_openFileDialogT
itle"], QDir().homePath(),

(self.translate[f"stext_{self.btnChangeLanguage.text()}_openFileDialog
TextFiles"]+" (*.txt *.md);;
"+self.translate[f"stext_{self.btnChangeLanguage.text()_openFileDialog
AllFiles"]+" (*.*)"),
        options=QFileDialog.DontUseNativeDialog)
    if fname[0] != "":
        self.textEdit.setHtml(self.styleDocument +
self.sizePic(self.loadWithEmptyLines(fname[0])))
        self.textEditContent = self.markdownWithEmptyLines()
        self.setWindowTitle(fname[0])
        self.currentPath = fname[0]
        self.deleteSpaces()
        self.textEdit.document().clearUndoRedoStacks()
```

Uložení souboru

Funkce textového editoru *saveFile()* zajišťující uložení souboru je velmi jednoduchá. Pouze zkontroluje, jestli existuje nějaká konkrétní cesta pro uložení souboru a v případě že ano, uloží pod touto cestou obsah textového editoru do textového souboru TXT a dokumentu PDF. Pokud konkrétní cesta neexistuje, vyvolá funkci *saveFileAs()*.

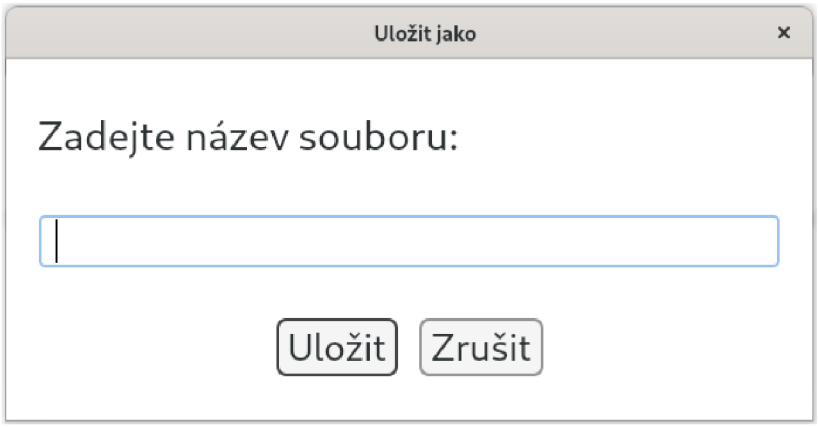
Pro připomenutí, funkce starající se o automatické průběžné ukládání soubory ukládá pouze ve formátu textového souboru TXT. Kdežto funkce *saveFile()* a *saveFileAs()* se starají o manuální ukládání souborů uživatelem ve formátu TXT a PDF.

Výpis 4.8: Ukázka kódu pro uložení souboru

```
    def saveFile(self):
        if self.currentPath != None:
            self.saveFileTXT(self.currentPath)
            self.saveFilePDF(self.currentPath)
        else:
            self.saveFileAs()
```

Uložení souboru jako

K „uložení souboru jako“ se využívá funkce pojmenována *saveFileAs()*. Při jejím vyvolání je automaticky vytvořen dialog uložení SaveDialog, ve kterém se vyplní pouze název souboru. Dialog následně kontroluje po potvrzení tlačítka „uložit“, zda byl název souboru vyplněn. Pokud ano, je název souboru testován, jestli neobsahuje zakázané znaky. Tyto znaky mohou být například tečka, lomítko, středník a jiné, které by mohly ovlivnit kompatibilitu a funkčnost textového souboru. V opačném případě je uživatel upozorněn, že nesplnil podmínky pro uložení souboru. Dialog uložení taktéž podporuje hlasovou asistenci a překlad, jako tomu je u zbylého textového editoru.



Obr. 4.6: Dialog k uložení souboru

Po potvrzení dialogu a splnění uvedených podmínek je vytvořena cesta souboru s odpovídajícím jménem souboru zadaným uživatelem, nastavena konkrétní cesta, vymazána průběžná záloha souboru (nahrazena tímto souborem), vymazána průběžná cesta a uložen obsah textového editoru do textového souboru TXT a dokumentu PDF. Tyto soubory jsou uloženy do složky /home pod uvedeným názvem.

Výpis 4.9: Ukázka kódu pro uložení souboru jako

```
    def saveFileAs(self):
        dlg = SaveDialog(self.btnChangeLanguage.text())
        dlg.exec_()
        fileName = dlg.getResults()

        if fileName != None:
            pathName = QDir().homePath() + "/" + fileName + ".txt"
            self.currentPath = pathName
            self.saveFileTXT(pathName)
            self.saveLastFileName()
            self.setWindowTitle(pathName)
            if self.continuosPath != None:
                if os.path.exists(self.continuosPath):
```

```
                os.remove(self.continuosPath)
            self.continuosPath = None

            self.saveFilePDF(pathName)
```

Výzva k uložení souboru při zavření hlavního okna

Senior může velmi snadno zapomenout uložit textový soubor před ukončením hlavního okna textového editoru. Proto byla přepsaná metoda *closeEvent()*, která vyzívá uživatele, jestli chce daný text v editoru uložit do souboru, či nikoliv. Tato metoda se z principu vždy spustí automaticky při pokusu o vypnutí okna, byla však modifikována, aby sekvenci svých příkazů provedla pouze, pokud je momentálně spuštěno hlavní okno textového editoru (nikoliv domovská stránka) a pokud od minulého uložení byly provedeny nějaké změny v textovém poli. V opačném případě se vytvoří jednoduchý dialog třídy QMessageBox vyzívají uživatele, jestli chce textový soubor uložit. Po stisknutí jednoho ze tří tlačítek „Ano“ (uložit), „Ne“ (neukládat) nebo „Zrušit“, provede metoda danou akci.

Výpis 4.10: Ukázka kódu fungování dialogu při zavření hlavního okna

```
dialog.setWindowTitle(translate[f"stext_{language}_closeDialogTitle"])
dialog.setText(translate[f"stext_{language}_closeDialogText"])
dialog.addButton(QPushButton(translate[f"stext_{language}_closeDialogY
esRole"]), QMessageBox.YesRole) #value - 0
dialog.addButton(QPushButton(translate[f"stext_{language}_closeDialogN
oRole"]), QMessageBox.NoRole) #value - 1
dialog.addButton(QPushButton(translate[f"stext_{language}_closeDialogR
ejectRole"]), QMessageBox.RejectRole) #value - 2

answer = dialog.exec_()
self.currentIndex()
if answer == 0:
    self.currentWidget().saveFile()
    event.accept()
elif answer == 2:
    event.ignore()
else:
    if self.currentWidget().continuosPath != None:
        if os.path.exists(self.currentWidget().continuosPath):
            os.remove(self.currentWidget().continuosPath)
```

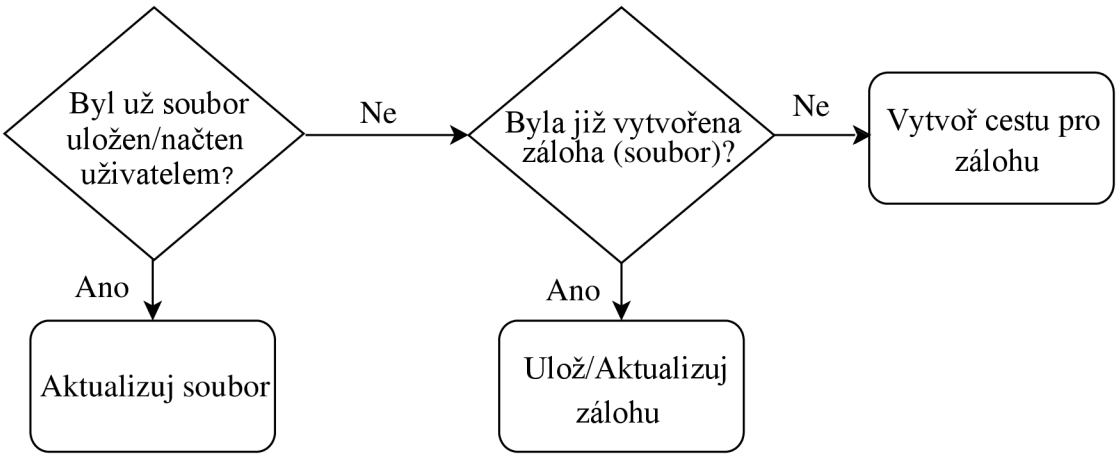
Průběžné ukládání souborů

Funkce textového editoru zajišťující průběžné ukládání textových souborů, vznikla za účelem předcházení ztracení dat, kvůli nedbalosti průběžného manuálního ukládání textových souborů uživatelem, popřípadě proti pádu PC, operačního systému, či celé aplikace.

Automatické průběžné ukládání probíhá každé 2 minuty a pouze pro textové soubory TXT (dokumenty PDF nejsou aktualizovány). Toto je zajištěno třídou QTimer, která při nastaveném čase, spouští (vykonává) v cyklech danou funkci. V tomto případě třída

QTimer vykonává funkci *automaticSave()*, která jak už z názvu vypovídá, se stará o automatické ukládání.

Funkce *automaticSave()* nejprve zkontroluje, jestli textový editor disponuje nějakou konkrétní cestou souboru. Pokud textový editor disponuje konkrétní cestou, jedná se buďto o soubor již uložený uživatelem nebo o soubor načtený. Tento soubor bude tedy funkce průběžné aktualizovat (ukládat), co dvě minuty. Pokud textový editor nedisponuje konkrétní cestou, jedná se o nový soubor, který ještě nebyl uložen uživatelem. Funkce následně zkontroluje, zda už vytvořila nějakou průběžnou cestu pro zálohy. V případě že průběžná cesta je již vytvořená, funkce průběžné ukládá text z textového pole editoru do této zálohy, v opačném případě vytvoří novou průběžnou cestu. Průběžná cesta je tvořena cestou domovské složky /home uživatele a názvem souboru, který je ve tvaru *file_08_15__15_02_2023.txt,* kde první skupina číslic odpovídá času a druhá konkrétnímu datu. Textový editor maže automaticky tyto zálohy v případě, že uživatel uloží soubor pod konkrétní cestou, nebo pokud soubor neuloží vůbec a při vypínání programu potvrdí, že nechce daný soubor uložit.



Obr. 4.7: Vývojový diagram automatického průběžného ukládání souborů

4.6 Formátování

Formátování textu je implementováno pomocí značkovacího jazyka Markdown, který je pro uživatele intuitivní, jednoduchý a rychlý. V textovém editoru může uživatel buďto formátovat text pomocí formátovacích značek jazyka Markdown, kdy výsledek formátování je možné vidět až po vykreslení do PDF dokumentu, nebo formátovat text pomocí vestavěných funkcí formátování v podobě tlačítek nacházejících se v navigační liště. Tyto vestavěné funkce formátování upravují text v reálném čase. Uživatel může tedy vidět změny okamžitě, nikoliv až v dokumentu PDF. Je nutné podotknout, že tyto vestavěné funkce pracují na pozadí stále se značkovacím jazykem Markdown. To, co uživatel vidí na obrazovce při formátování, není přesná podoba textu, který je uložen do textového souboru. V textovém souboru je uložen obsah textového editoru pouze jako

prostý text bez jakéhokoli viditelného formátování, uvozen do Markdown značek. V následujících kapitolách jsou popsány pouze tyto vestavěné funkce.

V knihovně PyQt5 se nabízí ještě jedna možnost jak formátovat text v textovém editoru, a to je pomocí jazyka HTML. Jazyk HTML nabízí mnohem větší škálu možností formátovaní textu – barevný text, podtržení textu, zarovnávání textu, možnost dvou velikostí fontu na jednom bloku textu, jednoduché řádkování aj. Na druhou stranu surový HTML text není tolik přehledný jako prostý text Markdown. Tedy pro ukládání textových dokumentů, jiného formátu než PDF, je vhodnější jazyk Markdown. Je také nutno podotknout, že nasazení jazyka CSS s HTML posouvá schopnost formátování textu úplně na jinou úroveň. Markdown samozřejmě umožnuje použít jazyk HTML v dokumentu Markdown, ale jeho čitelnost je poté horší. Proto textový editor využívá schopností jazyka HTML pouze pokud je to nutností.

Tučný text a kurzíva

Funkce formátování editující text jako tučný nebo psaný kurzívou, fungují stejně jako i u jiných textových editorů. Při jejich aplikaci nejprve funkce vyhodnocuje, zda selektovaný text nebo text v místě umístění kurzoru, je formátován jako tučný nebo kurzívou. V případě že nikoliv, funkce naformátuje text dle přání uživatele. Pokud ano, je text přeformátován na normální, vychází formát textu (bez jakýchkoliv stylistických úprav). Tlačítka spouštějící tyto formátovací funkce, bylo nutné nastavit jako zaškrtávací, fungující jako přepínače, protože porovnávají daný text ve dvou stavech, jestli je upraven touto funkcí či nikoliv. Je tedy možné v budoucnu navíc implementovat signál knihovny PyQt5, který bude kontrolovat tyto stavy a následně graficky upravovat vizuál tlačítka. Například jestli by text v místě umístěný kurzoru byl tučný, tlačítko „Tučný“, odpovídající této funkci, by se rozsvítilo jinou barvou než všechny ostatní neaktivní tlačítka.

Výpis 4.11: Ukázka kódu funkcí formátování tučného a písma kurzíva

```
self.boldAction = QAction("Bold")
self.boldAction.setCheckable(True)
self.boldAction.triggered.connect(lambda x:
self.textEdit.setFontWeight(QFont.Bold if x else QFont.Normal))

self.italicAction = QAction()
self.italicAction.setCheckable(True)
self.italicAction.toggled.connect(self.textEdit.setFontItalic)
```

Nadpis

U formátování nadpisů je třeba dbát na několik okolností, které nejsou na první pohled viditelné, ale dokáží výrazně ovlivnit rozdíl vykreslení textu mezi PDF a textovým editorem. V knihovně PyQt5 se pro označení bloku textu jako Nadpis 1 používá třída QTextBlockFormat a její metoda *setHeadingLevel(1)*, která před daný blok vloží hashtag s mezerou (značení Nadpisu 1 v Markdown). Tento způsob analogicky probíhá například

pro HTML. Tyto tagy, značky (hashtag) nelze v textovém editoru vidět, ty se projeví až v uloženém textovém souboru. Hlavní je si uvědomit, co je to blok textu. Blok textu je základní jednotka textu v QTextDocument (jednoduše textové pole), který obsahuje jeden nebo více řádků oddělenými koncovým znakem, např. novým řádkem. Blok textu může být například nadpis, standardní odstavec, seznam nebo tabulka. Pokud uživatel chce označit pouze vybraný text v bloku jako Nadpis 1 (to stejné platí i pro seznamy, tabulky atd.), nelze tuto akci provést, protože se tato aplikace vždy provede na celý text v bloku. V případě pokud je *headingLevel(1)* nastaven na konkrétní blok, ještě neznamená, že text se naformátuje, vizuálně změní do podoby Nadpisu 1 v textovém editoru. Tato změna bude provedena pouze na pozadí a projeví až v konečném vykreslení do dokumentu PDF nebo textového souboru. Formátovaní textu ve stylu Nadpisu 1 se provádí pomocí třídy QTextCharFormat podle nastavených parametrů (velikost textu, váha nebo barva písma), nebo pomocí HTML tagů. Toto formátování je však aplikováno na označený text kurzorem (nemusí se nutně jednat o celé textové bloky). Například pokud selektovaná část textu v bloku by byla nastavena jako blok s *headingLevel(1)* (Nadpis 1) a formátována pomocí HTML tagů *h1*, tak by se sice tento vybraný text v textovém editoru zvětšil na požadovanou velikost *h1* vůči ostatnímu textu v bloku, ale při vykreslení do PDF by byl celý text bloku vykreslen stejně, tedy na velikost *h1* i s ostatními částmi textu v bloku, které nebyly označeny, právě díky nastavení *setHeadingLevel(1)*, aplikující se na celý blok. Je třeba opětovně dbát na to, co se jeví jako možné formátování v textovém editoru, nemusí být nutně vykresleno stejně v PDF dokumentu.

Implementace funkce *setHeader()*, která má za úkol upravit vybraný text do formátu Nadpisu 1, řeší všechny problémy popsané výše. Tato funkce obsahuje poměrně dost podmínek, které kontrolují, v jaké části bloku se vybraný text uživatelem nachází. Nejprve zkontroluje, jestli text, který chce uživatel formátovat jako Nadpis 1, již není pod tímto formátem upraven. Poté ověří, zda je vůbec nějaký text uživatelem vybrán. V případě, že není žádný text uživatelem vybrán, textový editor za něj automaticky označí celý blok textu nacházející se v aktuálním umístění kurzoru. Následně je automaticky (textovým editorem) i manuálně (uživatelem) selektovaný text „smazán“ (vytržen) a pomocí HTML tagů upraven v následující podobě: *formattedText* = f"<h1>{text}</h1> ". Totéž by bylo možné implementovat i pomocí třídy QTextCharFormat, což by ale bylo mírně náročnější. Dvojice tagů *h1* formátují text do vizuální podoby Nadpisu 1 a další dvojice tagů p má za úkol odřádkovat text a pokračovat ve standardním formátování (normální odstavec textu). Pokračuje se ověřováním, jestli se vybraný text nachází uprostřed bloku nebo končí jeho selekce na konci bloku a začíná někde uprostřed. Pokud označený text odpovídá první podmínce, textový editor tento text odřádkuje na začátku selekce, označí tento blok jako *headingLevel(1)*, do tohoto bloku vloží již naformátovaný text pomocí výše uvedených HTML znaků a vytvoří nový blok (nový řádek) s *headingLevel(0)*, který odpovídá

klasickému odstavci. V druhé podmínce je text formátován velmi podobně, pouze není po vloženém naformátovaném textu vytvořen nový blok, ale následující blok je nastaven jako standardní odstavec. V ostatních případech je pouze vytvořen blok s *headingLevel(1)*, vložen naformátovaný text a následujícímu bloku je zase přiřazeno výchozí nastavení textu. Tyto jednotlivé posloupnosti úkonů jsou provedeny v jednom kroku pomocí metod textového kurzoru *beginEditBlock()* a *endEditBlock()*. Tedy uživatel si těchto několika změn jdoucích po sobě nemůže všimnout, vidí pouze výsledek. Také je zajištěna kompatibilita s funkcí vrácení změn *undo()* v jednom kroku, tudíž se uživatel nemusí proklikávat pomocí této funkce zpět do stavu před formátováním nadpisu. Těmito podmínkami bylo dosaženo, že označený text je vždy umístěn na novém bloku s nastavením *headingLevel(1)* a odpovídajícímu formátování, čímž bylo zajištěno stejné vykreslování nadpisů v textovém editoru i následnému PDF dokumentu.

Výpis 4.12: Ukázka kódu funkce formátování nadpisů

```
def setHeader(self):
    cursor = self.textEdit.textCursor()
    if cursor.blockFormat().headingLevel() != 1:
        cursor.beginEditBlock()

        # Get selected text or current line
        if not cursor.hasSelection():
            cursor.select(cursor.LineUnderCursor)
        text = cursor.selectedText().strip()
        formattedText = f"<h1>{text}</h1><p></p>"

        selectionAtStart = cursor.selectionStart() ==
cursor.block().position()
        selectionAtEnd = cursor.selectionEnd() ==
cursor.block().position() + cursor.block().length() - 1

        # Replace text with formatted text
        if not selectionAtEnd and not selectionAtStart:
          if not cursor.atStart() or not cursor.selectionStart() == 0:
                cursor.insertText("\n")
            cursor.setBlockFormat(self.formatHeader1)
            cursor.insertHtml(formattedText)
            cursor.insertText("\n")
            cursor.setBlockFormat(self.formatParagraph)
        elif selectionAtEnd and not selectionAtStart:
            cursor.insertText("\n")
            cursor.setBlockFormat(self.formatHeader1)
            cursor.insertHtml(formattedText)
            cursor.setPosition(cursor.NextBlock)
            cursor.setBlockFormat(self.formatParagraph)
        else:
            cursor.setBlockFormat(self.formatHeader1)
            cursor.insertHtml(formattedText)
            cursor.setBlockFormat(self.formatParagraph)

        cursor.endEditBlock()
```

Normální text

Definovat blok textu jako normální text (klasický odstavec) lze pomocí metody *setHeadingLevel(0)*. O toto nastavení se stará funkce textového editoru *setParagraph()*, která nejprve zkontroluje dvě základní podmínky a to, jestli se při vyvolání funkce nachází textový kurzor na bloku Nadpisu 1 nebo jestli je kurzorem vybrán nějaký text. V případě, že je první podmínka splněna, tedy kurzor se nachází na bloku Nadpis 1, je tento blok označen celý. Pokud v druhé podmínce není vybrán žádný text kurzorem, textový editor automaticky vybere slovo nacházející se pod tímto kurzorem. Následně se selektovaný text formátuje dle parametrů objektu *paragraphCharFormat* – velikost fontu výchozí nebo zvolená uživatelem a váha fontu normální. Také je pro blok tohoto textu nastaven *headingLevel(0)* odpovídající standardnímu odstavci.

U tohoto řešení bylo pro formátování textu mnohem efektivnější využít třídu QTextCharFormat, kvůli tomu, že zde formátovaný text není nijak přesouván, ani s ním nebylo pokročile manipulováno.

Výpis 4.13: Ukázka kódu funkce formátování normálního/standardního textu

```
def setParagraph(self):
    cursor = self.textEdit.textCursor()
    cursor.beginEditBlock()
    if cursor.block().blockFormat().headingLevel() == 1:
        cursor.select(QTextCursor.LineUnderCursor)
    elif not cursor.hasSelection():
        cursor.select(QTextCursor.WordUnderCursor)

    paragraphCharFormat = QTextCharFormat()
paragraphCharFormat.setFontPointSize(self.textEdit.font().pointSize())
    paragraphCharFormat.setFontWeight(QFont.Normal)

    cursor.setBlockFormat(self.formatParagraph)
    cursor.setCharFormat(paragraphCharFormat)
    cursor.endEditBlock()
```

Obrázky

Vkládání obrázku do textového pole editoru má v kompetenci funkce *insertPicture()*. Při vyvolání této funkce je uživatel dotázán, aby pomocí souborového dialogu vybral obrázek, který si přeje vložit do textového pole. Obrázek je následně nahrán pomocí třídy QPixmap a přeformátován na šířku o 40 pixelů menší než je šířka textového pole. Tímto je zajištěno, že obrázky s příliš velkým rozlišením nepřetečou až za hranice tohoto textového pole. Funkce dále uplatňuje podobné podmínky jako u formátování nadpisů, kde kontroluje, zda se kurzor nenachází na začátku textového pole nebo naopak, zda se nachází ve středu bloku textu dat. Pokud ano, obrázek je vložen na nový řádek pomocí HTML tagu *img* s odpovídající šířkou a výškou. Poté je textový kurzor přemístěn na nový blok pod obrázkem.

Značkovací jazyk Markdown nenabízí možnost nastavit velikost obrázků, zatímco jazyk HTML tuto funkčnost podporuje. Při importování kompletního dokumentu v Markdownu však není jednoduché ovlivnit velikost všech obrázků. Proto byla vytvořena funkce *sizePic()* pracující s knihovnou BeautifulSoup, která umožňuje nastavit jednotlivé parametry tagů HTML, třeba požadovanou šířku obrázku. Tato funkce nejprve přeloží *markdownText*, který odpovídá obsahu Markdown dokumentu, do HTML a vyhledává v něm všechny obrázky, kterým nastaví stejnou, jako u importování samostatných obrázků, šířku o 40 pixelů menší, než je šířka textového pole. Tím je zajištěno, aby všechny obrázky importované z Markdown souboru byly v souladu s rozměry dokumentu a nevyčnívaly mimo text.

Výpis 4.14: Ukázka kódu funkce pro vkládání obrázků do textového pole

```
def insertPicture(self):
    filePath, _ = QFileDialog.getOpenFileName(self,
self.translate[f"stext_{self.btnChangeLanguage.text()}_insertPictureDi
alogTitle"],
            QDir().homePath(),
self.translate[f"stext_{self.btnChangeLanguage.text()}_insertPictureDi
alogFiles"] +" (*.png *.PNG *.jpg *.JPG *.jpeg *.gif)",
options=QFileDialog.DontUseNativeDialog)

    if filePath:
        pixmap = QPixmap(filePath)
        # Setting the maximum width of the label corresponding to the
width of the QTextEdit
        maxWidth = int(self.textEdit.width() -
self.textEdit.document().documentMargin() * 2) - 40
        pixmap = pixmap.scaledToWidth(maxWidth,
Qt.SmoothTransformation)

        cursor = self.textEdit.textCursor()
        cursor.beginEditBlock()

        if cursor.hasSelection():
            cursor.removeSelectedText()

        if (not cursor.atBlockEnd() and not cursor.atBlockStart()) or
(cursor.atBlockEnd() and not cursor.atStart()):
            cursor.insertText('\n')
        # Insert image into QTextEdit
        cursor.insertHtml(f'<img
src="{QUrl.fromLocalFile(filePath).toString()}"
width="{pixmap.width()}" height="{pixmap.height()}">')
        cursor.insertText('\n')

        cursor.endEditBlock()
```

Seznamy

Uživatel má možnost si vybrat mezi dvěma seznamy – očíslovaný a odrážkový. Vkládání těchto listů do textového pole provádí funkce *addList()* s parametrem *style*, který určuje, jaký typ seznamu bude vytvořen. K vytvoření konkrétního listu se využívá třídy QTextListFormat, která obsahuje až 8 možných typů seznámů. Tyto typy seznamů lze volit dle metody *setStyle()* s parametrem typu int odpovídající číselnému označení daného seznamu. Očíslovaný seznam QTextListFormat.ListDecimal odpovídá číselnému označení -4, odrážkový seznam QTextListFormat.ListDisc zase číselnému zobrazení -1. Tyto číselné označení seznamů jsou zapsaný do funkce *addList()* pomocí tlačítek *btnOrdered* a *btnUnordered*, které se starají o vyvolání funkce, odpovídající parametru style.

Funkce *addList()* je vyvolána pomocí jednoho z tlačítek k vytvoření seznamu. Zkontroluje, zda text, pod kterým se nachází textový kurzor není blok typu Nadpis 1. V případě že ano, tento blok textu změní na normální text. Následně nastaví bloku textu odsazení od levého okraje textového pole, aby text byl lépe čitelný a přehledný, a vytvoří pro tento blok příslušný seznam určitého typu (stylu).

Výpis 4.15: Ukázka kódu funkce formátování seznamů

```
def addList(self, style):
    cursor = self.textEdit.textCursor()
    cursor.beginEditBlock()

    if cursor.block().blockFormat().headingLevel() == 1:
        cursor.setBlockFormat(self.formatParagraph)

        paragraphCharFormat = QTextCharFormat()

paragraphCharFormat.setFontPointSize(self.textEdit.font().pointSize())
        paragraphCharFormat.setFontWeight(QFont.Normal)
        cursor.select(QTextCursor.LineUnderCursor)
        cursor.setCharFormat(paragraphCharFormat)

    blockFmt = cursor.blockFormat()
    listFmt = QTextListFormat()

    if cursor.currentList():
        listFmt = cursor.currentList().format()
    else:
        listFmt.setIndent(1)
        blockFmt.setIndent(1)
        cursor.setBlockFormat(blockFmt)

    listFmt.setStyle(style)
    cursor.createList(listFmt)
    cursor.endEditBlock()
```

Doplňující funkce formátování

Aby docházelo ke korektnímu formátování textu a nastavování vlastností bloků při manuální úpravě textu uživatelem (mazání bloků, odřádkování začátku bloku, přesouvání textu myší), byly vytvořeny doplňkové funkce. Tyto doplňkové funkce pracují na pozadí textového editoru a jsou vyvolány například změnou počtu bloků v dokumentu, stisknutí určité klávesy nebo přetažení textu myši. Problém například spočívá v tom, že pokud uživatel smaže (začátek) blok textu, který následně přeteče do odlišného bloku nad ním, nastane situace, že se formáty textu v tomto bloku budou lišit. Text tedy nebude následně vhodně vykreslen do dokumentu PDF. Další komplikací může být například skutečnost, že pokud uživatel odřádkuje blok textu Nadpis 1 na jeho začátku, tak tento blok sice bude stále typu Nadpis 1, ale vykreslen (formát) v textovém editoru bude stejně jako normální text. Pro řešení těchto zásadních problémů, je třeba nainstalovat pro textové pole *event filtr*, který kontroluje, jaké klávesy uživatel stisknul. Konkrétně bude kontrolovat stisknutí kláves *Enter* a *Backspace*, které souvisejí s popsanými záležitostmi výše.

V případě, že je stisknuta klávesa *Backspace*, filtr zkontroluje, jestli se momentálně textový kurzor nachází na začátku bloku, tedy v pozici 0. Při splnění tohoto předpokladu filtr pokračuje dále. Nejprve zkontroluje, zda předchozí blok, resp. blok nacházející se nad blokem umístěný kurzoru, vůbec existuje. Pokud ano a předchozí blok je typu Nadpis 1, nastaví se jemu odpovídající QTextCharFormat pro celý blok v umístění textového kurzoru, tedy pro blok, který má být vymazán. Jestli je předchozí blok standardní odstavec, aplikuje je se na něj QTextCharFormat pouze s velikostí fontu odpovídající velikosti normálního textu, což má využití například, pokud chce uživatel smazat blok typu Nadpis 1. který má následně přetéct do standardního bloku.

Pokud je stisknuta klávesa *Enter*, filtr ověření, jestli textový kurzor se nachází na začátku bloku a je tento blok typu Nadpis 1. Jestli ano, je tento blok nastaven na standardní a proměnné *enterBlock* je přiřazen stav *True*, s kterou následně pracuje funkce *handleBlockCountChanged()* vyvolaná při každé změně počtu bloků v textovém dokumentu. Při stisknutí klávesy *Enter* nelze nastavit parametry pro budoucí vytvořený blok, proto je třeba využít signálu mapující změnu počtu bloků. Funkce *handleBlockCountChanged()* je implementována, aby prováděla své úkony pouze za předpokladu, že došlo k vytvoření nového bloku textu, nikoliv při jeho smazání, a pokud proměnná *enterBlock* nabývá hodnoty *True*. Tato funkce nastaví blok jako Nadpis 1 v umístěné textového kurzoru, tedy pro blok, který byl odřádkován, a přechozí blok, na kterém se text předtím nacházel, nakonfiguruje jako standardní odstavec s odpovídajícím formátem. Poté je do proměnné *enterBlock* vepsán stav *False*.

Upravena byla také metoda *dropEvent()* pro přetahování textu pomocí kurzoru myši v textovém editoru. V základním nastavení tato metoda funguje na principu, že zachovává formát textu, který byl přetažen. Pokud například uživatel chce přesunout text z bloku typu Nadpis 1 do standardního bloku textu, bude tento blok obsahovat dva odlišné formáty – standardní text a text stylu *h1*, což se opět projeví nekorektním vykreslením

v dokumentu PDF. Metoda *dropEvent()* byla přepsána, aby vracela vždy text, který bude odpovídat formátu bloku, do kterého je tento text přetažen. Tuto funkcionalitu zajišťuje třída QMimeData, která se nezabývá formátováním textu, ale pouze jeho obsahem. Tudíž přetažený text automaticky dědí vlastnosti formátování bloku, do kterého byl přetažen.

Prázdné řádky

Jazyk Markdown sám o sobě nepodporuje prázdné (volné) řádky mezi odstavci, ale využívá pro dosažení této vlastnosti jazyk HTML, konkrétně tag <br>, což ale pro seniora je velmi nepraktické. Byla teda vytvořena skupina funkcí zajišťující kompatibilitu prázdných řádků se značkovacím jazykem Markdown. Uživatel by však bez použití těchto funkcí v textovém editoru mohl vytvořit prázdný řádek, ale ten by se neprojevil v uloženém textovém souboru, ani v PDF dokumentu.

Metoda *toMarkdown()*, která překládá text z textového pole, nevypisuje přímo prázdné řádky, pouze řádky na kterých se nachází alespoň jeden znak (např. mezera). Funkce *markdownWithEmptyLines()* využívá této vlastnosti, kdy na jednotlivé prázdné řádky přidá mezeru a následně vrací Markdown text již se zdánlivě prázdnými řádky, který je uložen do textového souboru nebo dokumentu PDF (dle funkce uložení). Následně, pokud došlo k přidání těchto mezer, textový editor vrátí provedené změny, tedy vymaže přidané mezery. Funkce *markdownWithEmptyLines()* také obsahuje cyklus starající se o správné vykreslení mezer na prázdných řádcích, které navazují na formát textu tučný, kurzíva nebo jejich kombinace (tzv. tučné mezery) a regulární výraz, který opravuje někdy nesmyslné odřádkování dlouhých textů jednoho textové bloku metody *toMarkdown()*. Výstup (text) funkce je použita pro export do textového nebo PDF dokumentu.

Výpis 4.16: Ukázka funkce zajišťující vykreslení prázdných řádku v jazyce Markdown

```
def markdownWithEmptyLines(self):
    cursor = self.textEdit.textCursor()
    cursor.beginEditBlock()
    emptyLinesChecker = False
    for block in range(self.textEdit.document().blockCount()):
        current_block =
self.textEdit.document().findBlockByNumber(block)
        if current_block.length() == 1:
            cursor.setPosition(current_block.position())
            cursor.insertText(" ")
            emptyLinesChecker = True
    # for correct indentation of long texts and saving lists
    text = re.sub(r'(?<!\n)\n(?!\n|\s*(\d+\.|-|\*)\s+)', '
',self.textEdit.toMarkdown())
    lines = text.splitlines()
    for i, line in enumerate(lines):
        if (re.search(r"^\*\*\s*\*\*$", line) or
re.search(r"^\*\s*\*$", line) or re.search(r"^\*\*\*\s*\*\*\*$",
line)):
            lines[i] = " "
```

```
    result = "\n".join(lines)
    if emptyLinesChecker:
        self.textEdit.undo()
    cursor.endEditBlock()

    return result
```

K zajištění zpětné kompatibility importu Markdown textu s prázdnými řádky z textového souboru do textového pole slouží funkce l*oadWithEmptyLines()*, která načte textový soubor zvolený uživatelem a každý řádek kontroluje, jestli se nejedná o standardní blok nebo o blok typu Nadpis 1. Standardní blok s mezerou je nahrazen řetězcem „ “ a blok Nadpis 1 řetězcem „<h1> </h1>“. Výstup funkce je následně předán do funkce *sizePic()*, která se stylem HTML s povinným parametrem *white-space: pre-wrap* u paragrafů *p* a nadpisů *h1*, zajišťuje správné vykreslení prázdných řádků. Následně po importu textu ze souboru je vyvolána funkce *deleteSpaces()* a metoda *clearUndoRedoStacks()*, pro vymazání přebytečných mezer a záznamů úprav, aby se uživatel nemohl vrátit do stavu s mezerami na zdánlivě prázdných řádcích.

Výpis 4.17: Ukázka kódu funkce zajišťující zpětnou kompatibilitu prázdných řádků

```
def loadWithEmptyLines(self, file):
    with open(file, "r") as f:
        text = f.read()
    lines = text.split('\n')

    for i, line in enumerate(lines):
        if re.match(r'^\s+$', line): #If the line contains only spaces
            # Replacing multiple spaces with a single spac
            line = re.sub(r'\s{1,}', '<p> </p>', line)
            lines[i] = line
        elif line.strip() == "#":
                lines[i] = "<h1> </h1>"

    result = "\n".join(lines)
    return result
```

Při exportování PDF dokumentu je nutností již používat tagy <br> pro vytvoření prázdného řádku. Proto funkce *loadWithEmptyLines()* byla modifikována s tagem <br> uvnitř řetězců „ “ a „<h1> </h1>“ jako nová funkce *loadWithEmptyFilesPDF()*.

Změna velikosti písma

Pro změnu velikosti fontu písma byla vytvořena trojice tlačítek – 18pt, 22pt, 28pt. O změnu velikosti písma se stará funkce *changeFontSize()* s parametrem odpovídající textu, které dané tlačítko obsahuje. Tedy tlačítko „18pt“ mění velikost písma na 18pt.

Tato funkce také mění text znázorňující velikost fontu ve stavovém řádku. Změna velikosti fontu je aplikována na celý text.

Další možností, jak nastavit velikost písma, by bylo možné použít pole seznamů neboli combo box, kde by měl uživatel podstatně více variant velikostí na jednom místě. Toto řešení by však mohlo být pro seniora matoucí.

Výpis 4.18: Ukázka kódu funkce měnící velikost fontu

```
    def changeFontSize(self, fontSize):
        self.textEdit.setFont(QFont("Arial", fontSize))
        self.lblFontSizeStat.setText(str(fontSize)+ "pt")
```

Funkce pro práci s textem

Funkce pro úpravu textu jsou velmi jednoduché využívající základní metody třídy textového pole pro prostý text QPlainTextEdit. Jedná se o metody pro kopírování, vložení, vyjmutí textu a pro vracení změn, či jejich zrušení. Metoda pro vrácení změn není moc kvalitně implementována knihovnou PyQt5, ale pro svůj účel jednoduše postačí. Tyto všechny metody již mají defaultně nastavené klávesové zkratky díky knihovně PyQt5, nebylo třeba tuto část manuálně implementovat.

Výpis 4.19: Ukázka kódu přiřazení základních funkcí pro práci s textem

```
self.btnRedo.clicked.connect(self.textEdit.redo)
self.btnUndo.clicked.connect(self.textEdit.undo)
self.btnPaste.clicked.connect(self.textEdit.paste)
self.btnCopy.clicked.connect(self.textEdit.copy)
self.btnCut.clicked.connect(self.textEdit.cut)
```

4.7 Přibližování a oddalování textového pole

Kvůli různému rozlišení jednotlivých displejů počítače, a hlavně malému DPI prvku QTextEdit (textového pole) bylo nutností do textového editoru implementovat možnost škálovatelnosti textového pole. Jednotlivé QWidgety nelze škálovat, dokonce ani měnit měřítko zobrazení těchto widgetů v rozložení QLayout, lze pouze měnit jejich velikost. Zdánlivého přibližování a oddalování by šlo sice dosáhnout změnou velikosti textového pole QTextEdit a velikosti jeho písma, ale tato implementace by byla realizačně náročná a využívala by manuálních přepočtů zvolené velikosti fontu uživatelem a zobrazovaného textu, který by byl ve skutečnosti odlišné velikosti kvůli zvolené míře měřítka. Proto byla vybrána třída QGraphicsView, která funkci škálovatelnosti svých prvků zcela umožňuje. Tato třída je převážné vytvořena pro vykreslování a práci s jednotlivými obrazci, čar, geometrických útvarů, vkládání obrázku, ale lze do ní vložit i widgety a standardně s nimi manipulovat.

Třída QGraphicsView je úzce spojena s třídou QGraphicsScene, která slouží jako prostředník pro vkládání jednotlivých widgetů. Před implementací funkce bylo tedy třeba vytvořit objekty těchto tříd s názvy *view* a *scene* a vložit do scény *scene* textové pole

textEdit (QTextEdit), pro zobrazení ve *view*. Následně byly vytvořeny funkce *zoomIn()* a *zoomOut()* starající se o přibližování a oddalování textového pole. Tyto funkce mění měřítko zobrazovaných prvků ve scéně QGraphicsScene pomocí třídy QTransform využívají metodu *scale()*, kterou se nastavuje poměr změny měřítka scény. Tento poměr je v textovém editoru nastaven na výchozí hodnotu 1.2, tedy scéna své zobrazení widgetů vždy přiblíží nebo oddálí o 20 % své velikosti. V programu byl také nastaven maximální počet přiblížení, či oddálení grafické scény, aby senior nebyl zmatený například příliš malou velikostí textového pole, které by již nemuselo být zřetelné.

Je nutné si uvědomit, že pokud dojde k přiblížení grafické scény, je textové pole přiblíženo natolik, že dojde k jeho přetečení (konkrétně výška textového pole) za hranice této scény. Tato skutečnost má za následek, že v programu textového editoru by se zobrazovaly zároveň dva vertikální posuvníky (*scroll bars*) - jeden pro textové pole a druhý pro grafickou scénu. Netřeba říkat, že dva vertikální posuvníky mají neblahý vliv na uživatelskou přívětivost a bylo nutností tento problém vyřešit. Jedním z řešeních, které bylo aplikováno pro program, bylo při změně měřítka scény také měnit výšku textového pole. Tedy při přibližování scény dochází ke zmenšování výšky textového pole o poměr 1,2 (jako u změny měřítka scény), resp. při oddalování scény dochází ke zvětšování výšky textového pole o poměr 1,2. Ne vždy se ale změnou měřítka scény vyrovná její vertikální posuvník na výchozí hodnotu (začátek), především u přibližování. O vyrovnávání tohoto vertikálního posuvníku se stará vytvořená funkce *scaleStabilization()*, která je velmi jednoduchá. Pouze zapíná vertikální posuvník grafické scény, který je ve výchozím stavu aplikace vypnut, aby se v textovém editoru nezobrazoval, nastaví hodnotu posuvníku na 0, což odpovídá začátku a posuvník znovu vypne.

Výpis 4.20: Ukázka kódu přibližování textového pole (grafické scény)

```
@QtCore.pyqtSlot()
def zoomIn(self):
    if self.zoomNum < 5:
        scaleTr = QTransform()
        scaleTr.scale(self.factor, self.factor)

        tr = self.view.transform() * scaleTr
        self.view.setTransform(tr)

        self.textEdit.setFixedHeight(int(self.textEdit.frameGeometry().
        height()/1.2))
        self.scaleStabilization()

        self.zoomNum = self.zoomNum + 1
```

4.8 Vícejazyčný překlad aplikace

Díky velkému mezinárodnímu zájmu o Operační systém pro seniory, byl celý textový editor přeložen do anglického jazyka. Kompletní překlad aplikace se nachází v JSON souboru *translate.json*, kde název klíčů odpovídá názvu daného tlačítka, štítku, či jiného prvku a jeho hodnota odpovídá konkrétnímu překladu. V tomto souboru jsou uloženy překlady jak do českého jazyka, tak i do anglického. Funkce textového editoru *changeLanguage()* následně s tímto JSON souborem pracuje a postupně přepisuje popisy všech prvků aplikace hodnotami z tohoto souboru.

Výpis 4.21: Ukázka části obsahu JSON souboru umožňující vícejazyčný překlad

```
{
    "stext_en_btnOpenFile": "Open",
    "stext_en_btnNewFile": "New",
    "stext_en_btnSaveFile": "Save",
    "stext_en_btnSaveFileAs": "Save as",
        ...             ...             ...
    "stext_cz_openFileDialogTitle": "Otevřít soubor",
    "stext_cz_openFileDialogTextFiles": "Textové soubory",
    "stext_cz_openFileDialogAllFiles": "Všechny soubory"
}
```

4.9 Hlasová asistence

Hlasová asistence umožňuje zvukovou signalizaci při přejetí kurzoru myši na tlačítko nebo štítek programu. Tento zvukový signál slouží ke zlepšení uživatelského zážitku a orientace v programu pro uživatele se zrakovým postižením.

Jádrem hlasové asistence je funkce *mouseOverAudio()*, při jejíž aktivaci se přehraje nahraná zvuková stopa TTS (Text to speech) odpovídající názvu tlačítka nebo štítku, na který uživatel kurzorem myši namířil. Jednotlivé nahrávky TTS se nacházejí v adresáři *AudioTTS*. Podobně jako u překladu aplikace se i zde pracuje s JSON souborem, konkrétně *audioTTS.json*, v němž jsou uloženy všechny cesty nahrávek TTS. Funkce *mouseOverAudio()* nahraje všechny cesty těchto nahrávek a přiřadí metodě *enterEver()* tlačítek i štítků funkci *playAudio()*. Metoda *enterEvent()* slouží k provedení určité akce (funkce) po přejetí kurzorem myši na daný prvek. O přehrání dané nahrávky se stará funkce *playAudio()*, která využívá především tříd QMediaPlayer a QBuffer. Teoreticky k samotnému fungování této funkce by stačila pouze knihovna QMediaPlayer, která by načítala zvukové signály přímo ze souboru, to ale není možné, protože GStreamer tyto nahrávky neumí přehrát z lokálního souboru. Proto je využita také třída QBuffer, která nejprve svůj obsah vymaže (aby bylo možné přehrávat vícero nahrávek), následně načte data zvukového signálu a poskytne je třídě QMediaPlayer, která nahrávku přehraje.

Audio nahrávky TTS byly pořízeny pomocí webové stránky Free Text to Speech (TTS) od společnosti Wideo.

Výpis 4.22: Ukázka kódu pro přehrávání nahrávek TTS

```
player = QMediaPlayer()
buff = QBuffer()

def playAudio(filename):
    buff.close()
    with open(filename, "rb") as f:
        data = f.read()
    ba = QtCore.QByteArray(data)
    buff.setData(ba)
    buff.open(QtCore.QIODevice.ReadOnly)
    player.setMedia(QMediaContent(), buff)
    player.play()
```

Vypnout hlasovou asistenci v textovém editoru není možné. Toto by se však mělo změnit v budoucnu, kdy nastavení hlasové asistence i překlad aplikací (včetně textového editoru) se bude nacházet v aplikaci *srun*, která zprovozňuje spouštění všech aplikací operačního systému.

4.10 Možná vylepšení

Možnými vylepšeními pro následující práci může být například vytvoření funkce pro jednoduché přidání tabulek, možnost stránkování, hledání v textu, vytvoření vlastního zásobníku historie úprav (QUndoStack), přidání více formátů papíru nebo fontů.

Manuální vytváření tabulek uživatelem pomocí značkovacího jazyka Markdown je velmi zdlouhavé a pro seniora těžko pochopitelné. Je ale možné vytvořit funkci, která by se starala o vkládání tabulek buďto pomocí HTML jazyka nebo třídy QTextTable knihovny PyQt5.

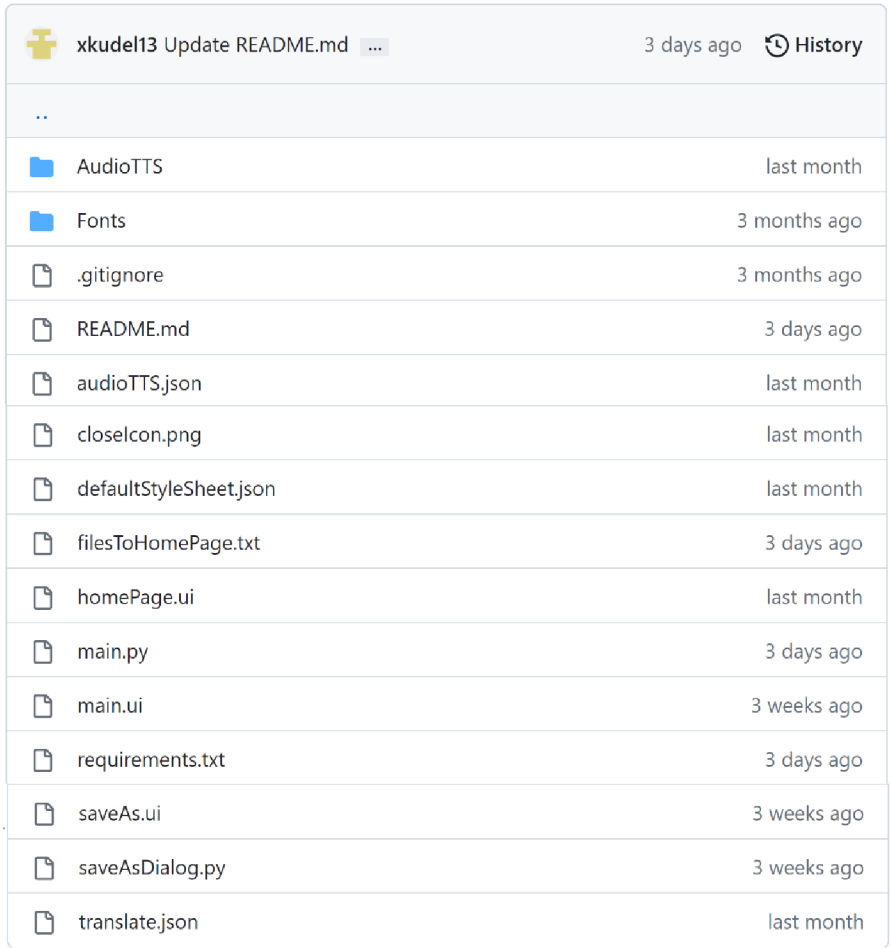
Možnost stránkování je také velmi vítaným vylepšením. V momentální verzi textového editoru se nachází jedno textové pole, které není rozděleno na určité úseky odpovídající jednotlivým stránkám, tedy uživatel přímo nemůže určit kde začíná a končí jednotlivá stránka, tudíž neví, jestli nějaká část textu, seznamu či celý obrázek, bude odřádkován na novou stránku. Toto by šlo vyřešit buďto definováním pevných hranic jednotného textového pole QTextEdit nebo vytvořením vícero těchto textových polí (fixní velikosti) pod sebou, odpovídajícím dílčím stránkám, v případě, že dojde k přetečení textu z jednoho textového pole do druhého. Poté už by bylo velice snadné přidat číslování stránek.

Vyhledávání jednotlivých slov či částí textů v textovém poli může být obzvlášť užitečné pro seniora, který se špatně orientu v obsáhlém textovém dokumentu. Z tohoto důvodu by bylo adekvátní implementovat funkci, při jejímž vyvolání např. pomocí tlačítka, se vytvoří malé textové pole v podobě řádků, díky němuž může uživatel vyhledat text v textovém poli.

Vytvoření vlastního zásobníku historie úprav je vhodné obzvlášť pro funkce formátování, které provádí dva kroky úprav místo jednoho nebo krok nadbytečný (pro uživatele neviditelný, ale pro funkčnost zásadní). Jedná se konkrétně o funkci *handleBlockCounrChanged()* pracující s funkcí, která se vyvolává po stisknutí tlačítka *Enter* a nebo funkci *markdownWithEmptyLines()* uložení prázdných řádků do textového souboru nebo PDF. Textové pole QTextEdit má svůj vlastní zásobník historie úprav, do kterého ale programátor nemá možnost zasahovat (např. vymazat jednotlivé kroky). Proto je potřeba vytvořit tento zásobník vlastní, třeba pomocí třídy QUndoStack.

5. ZVEŘEJNĚNÍ A INSTALACE TEXTOVÉHO EDITORU

Výsledky bakalářské práce a programový kód textového editoru jsou zveřejněny na platformě GitHub pod licencí MIT, konkrétně pod organizací *forsenior* v repozitáři *os* (Operating system for senios) v adresáři *stext*. Instalace textového editoru se realizuje primárně s celým operačním systémem pro seniory, lze však nainstalovat i samostatně. K instalaci kompletního operačního systému, včetně všech aplikací (textový editor součástí) a potřebných knihoven, byl vyvinut skript Tarikem Alkananem v rámci bakalářské práce Linuxová distribuce pro seniory s názvem *install_seniorOS.sh* v adresáři *sinstall*. Skript po jeho spuštění stáhne všechny potřebné soubory a knihovny, provede jejich instalaci a konfiguraci operačního systému. Po restartu systému se vykonají všechny potřebné změny a operační systém pro seniory se stane aktivním. Textový editor lze zprovoznit i samostatně stažením všech souborů z jeho adresáře a následného nainstalování potřebných knihoven pomocí příkazu *pip install -r requirements.txt* ve stažené složce (konkrétní knihovny lze nainstalovat i manuálně viz. GitHub). Pro spuštění aplikace slouží soubor *main.py*. Je ale nutné podotknout, že textový editor byl vyvinut pouze pro operační systémy s jádrem Linux.



Obr. 5.1: Struktura repozitáře GitHub textového editoru

6. ZÁVĚR

Cílem bakalářské práce bylo vytvořit textový editor přizpůsobený pro seniory ve věkové skupině 90 let a více, který bude lehce ovladatelný, bude podporovat základní formátování textu pomocí značkovacího jazyka Markdown a vhodnou správu dokumentů a jejich archivace.

Úkolem teoretické části bylo seznámení se s danou problematikou. Co je to značkovací jazyk Markdown, na jakém principu funguje jeho překlad do HTML, jaké má jazyk Markdown využití, použití základních značek Markdown s příklady překladu do formátu HTML a následného vykreslení. Dále byl popsán formát PDF, jeho historie, struktura, grafické prvky a omezení. V poslední kapitole teoretické části byly pospány všechny použité programové knihovny sloužící k překladu mezi formáty a k vytvoření přívětivého grafického prostředí textového editoru.

V praktické části byly podrobně vysvětleny konkrétní kroky k řešení daných problémů – převod prostého textu Markdown na formát PDF, funkce pro práci se soubory a formátování textu, tvorba grafického rozhraní, hlasový asistent, vícejazyčný překlad aplikace.

Pro překlad prostého textu z textového pole editoru do formátu HTML byla využita knihovna *Python-Markdown*, která je implementací Markdownu Johna Grubera. Následný převod HTML do dokumentu PDF byl realizován pomocí knihovny *xhtml2pdf*, kde ale vznikla největší překážka celé práce. Knihovna *xhtml2pdf* totiž defaultně neumí vykreslovat do formátu PDF diakritiku některých českých znaků. K řešení tohoto zásadního problému bylo třeba importovat do stylů HTML (CSS) font podporující českou diakritiku a také jeho variace *bold*, *italic* a jejich kombinace, pro jejich správné vykreslení. Tvorba formátovacích funkcí byla taktéž stěžejní částí bakalářské práce. Práce s bloky textu a jejich stylizace je složitá především se značkovacím jazykem Markdown a vyžaduje mnoho času na testování, jestli opravdu fungují správně (jestli text je vykreslen korektně do dokumentu PDF a textového souboru txt). Především při implementaci funkce pro formátování nadpisů a vytvoření prázdných řádků bylo třeba dbát zvýšené opatrnosti.

Grafické rozhraní úvodního hlavního okna textového editoru a dialogu pro ukládaní souborů bylo vytvořeno takovým způsobem, aby bylo dosaženo přehlednosti, snadné orientace a ovládání textového editoru. O jednoduché ovládání textového editoru se stará přehledný navigační panel.

Finální verze textového editoru je intuitivní, uživatelsky přívětivá a lehce ovladatelná. Přesto však možných vylepšení je několik, která jsou popsána výše v kapitole Možná vylepšení.

Zhotovený programový kód textového editoru je dostupný na stránkách GitHub: https://github.com/forsenior/os/tree/main/stext

LITERATURA

[1] MARKDOWN GUIDE. Getting Started. In: *Markdown Guide* [online]. [cit. 2022-11-25]. Dostupné z: https://www.markdownguide.org/getting-started/#how-does-it-work

[2] GRUBER, John. Markdown. In: *Daring Fireball* [online]. [cit. 2022-11-25]. Dostupné z: https://daringfireball.net/projects/markdown/

[3] PECKA, Miroslav. Markdown – aneb píšete články stále ještě ve Wordu?. In: Miroslav Pecka Blog [online]. [cit. 2022-11-30]. Dostupné z: https://miroslavpecka.cz/blog/markdown-editory/

[4] GRUBER, John. Markdown: Syntax. In: *Daring Fireball* [online]. [cit. 2022-11-25]. Dostupné z: https://daringfireball.net/projects/markdown/syntax

[5] Basic Syntax. In: *Markdown Guide* [online]. [cit. 2022-11-25]. Dostupné z: https://www.markdownguide.org/basic-syntax/

[6] GITHUB. Basic writing and formatting syntax. In: GitHub [online]. [cit. 2022-11-30]. Dostupné z: https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax

[7] IONOS. Markdown: Guide for the simple markup language. In: Digital Guide [online]. 2022 [cit. 2022-11-30]. Dostupné z: https://www.ionos.com/digitalguide/websites/web-development/markdown/

[8] MARKDOWN GUIDE. Extended Syntax. In: *Markdown Guide* [online]. [cit. 2022-11-25]. Dostupné z: https://www.markdownguide.org/extended-syntax/

[9] PYPI. Python-Markdown. In: PyPi [online]. [cit. 2022-11-30]. Dostupné z: https://pypi.org/project/Markdown/#description

[10] LIMBERG, Waylan. Python-Markdown 3.4.1 documentation. In: Python-Markdown documentation [online]. [cit. 2022-11-30]. Dostupné z: https://python-markdown.github.io/

[11] LIMBERG, Waylan. Using Markdown as a Python Library. In: Python-Makdown documentation [online]. [cit. 2022-11-30]. Dostupné z: https://python-markdown.github.io/reference/#tab_length

[12] HOLTWICK, Dirk. XHTML2PDF. In: PyPi [online]. [cit. 2022-11-30]. Dostupné z: https://pypi.org/project/xhtml2pdf/

[13] XHTML2PDF. Welcome to xhtml2pdf's documentation!. In: Xhtml2pdf [online]. [cit. 2022-11-30]. Dostupné z: https://xhtml2pdf.readthedocs.io/en/latest/#

[14] RIVER BANK COMPUTING. PyQt Documentation v5.15.4 - Introduction. River Bank Computing [online]. [cit. 2022-11-30]. Dostupné z: https://www.riverbankcomputing.com/static/Docs/PyQt5/introduction.html

[15] THOMPSON, Phil. PyQt5. In: PyPi [online]. [cit. 2022-11-30]. Dostupné z: https://pypi.org/project/PyQt5/

[16] ADOBE. About Adobe PDF. In: Adobe [online]. Adobe [cit. 2023-01-31]. Dostupné z: https://www.adobe.com/acrobat/about-adobe-pdf.html

[17] SCAN TO PDF, Scan to PDF. A Brief History Of The PDF. In: Scan to PDF [online]. [cit. 2023-01-31]. Dostupné z: https://scantopdf.com/blog/brief-history-of-the-pdf/
[18] SIMPSON. A Brief Look at the History of PDF. In: SwifDoo [online]. Deutsch, 2022 [cit. 2023-01-31]. Dostupné z: https://www.swifdoo.com/news/what-is-pdf-and-its-history
[19] ABBY FINEREADER PDF. What is a PDF?. In: ABBY FineReader PDF [online]. [cit. 2023-01-31]. Dostupné z: https://pdf.abbyy.com/learning-center/what-is-pdf/
[20] LUKAN, Dejan. PDF file format: Basic structure. In: Infosec [online]. 2020 [cit. 2023-02-01]. Dostupné z: https://resources.infosecinstitute.com/topic/pdf-file-format-basic-structure/
[21] FILEFORMAT. PDF File Format - What is a PDF file?. In: Fileformat [online]. [cit. 2023-02-01]. Dostupné z: https://docs.fileformat.com/pdf/
[22] ADOBE. Document management — Portable document format — Part 1: PDF 1.7 [online]. 1.7. 2008n. l. [cit. 2023-02-01]. Dostupné z: https://opensource.adobe.com/dc-acrobat-sdk-docs/pdfstandards/PDF32000_2008.pdf
[23] READ THE DOCS. Beautiful Soup Documentation. Read The Docs [online]. [cit. 2023-04-26]. Dostupné z: https://beautiful-soup-4.readthedocs.io/en/latest/

SEZNAM SYMBOLŮ A ZKRATEK

Zkratky:

MD	Markdown
HTML	Hypertext Markup Language
XHTML	Extensible Hypertext Makkup Language
CSS	Cascading Style Sheets
QSS	Qt Style Sheets
PDF	Portable Document Format
GUI	Graphic User Interface
TTS	Text to Speech