VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

MULTIPLATFORMNÍ NÁSTROJ PRO GENEROVÁNÍ TECHNICKÉ DOKUMENTACE Z XML

MULTI-PLATFORM TOOL FOR GENERATION OF TECHNICAL DOCUMENTATION FROM XML

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MICHAL UHRECKÝ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JOSEF STRNADEL, Ph.D.**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

154270

Ústav: Ústav počítačových systémů (UPSY)
Student: **Uhrecký Michal**
Program: Informační technologie
Název: **Multiplatformní nástroj pro generování technické dokumentace z XML**
Kategorie: Algoritmy a datové struktury
Akademický rok: 2023/24

Zadání:

1. Proveďte rešerši v oblasti ukládání informací ve formátu XML pro různé případy užití; zaměřte se zejména na využití XML pro uložení projektů vývojových prostředí.
2. Proveďte rozbor požadavků kladených na technickou dokumentaci; identifikujte třídu požadavků, jejichž splnění je možno automatizovat a předpoklady pro tuto automatizaci.
3. Proveďte rešerši metod, nástrojů, technologií a přístupů v oblasti automatizovaného generování technické dokumentace ze zdrojových souborů, zejména pak z XML.
4. Navrhněte multiplatformní nástroj, který bude schopen maximálně automatizovaně generovat čitelnou technickou dokumentaci z XML vstupu, případně doplněného o uživatelskou anotaci, konfiguraci apod. před spuštěním generování.
5. Navržený nástroj implementujte, proveďte jeho základní testování a připravte podmínky pro jeho zhodnocení.
6. Na základě uživatelského, popř. dalšího, testování zhodnoťte vlastnosti implementovaného nástroje, především co se týká jeho schopnosti generovat čitelnou dokumentaci z XML (přinejmenším předpokládejte SVD vstup, XML vstup pro popis projektu nástroje UPPAAL a jeden vlastní vstup) a z případných doplňujících informací dodaných uživatelem nástroje.
7. Zhodnoťte vlastnosti implementovaného nástroje a výsledky pomocí něj dosažené; identifikujte silné a slabé stránky implementace, navrhněte možné směry jejího vylepšení a rozveďte ty, které považujete za nejvíce perspektivní.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 4 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Strnadel Josef, Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1.11.2023
Termín pro odevzdání: 9.5.2024
Datum schválení: 30.10.2023

Abstrakt

Cieľom tejto práce je navrhnúť a vyvinúť generátor technickej dokumentácie, ktorý spracuje všeobecný XML vstup. Práca sa podrobne venuje analýze štruktúry XML dokumentov a definuje kritériá, ktoré by mala spĺňať kvalitná technická dokumentácia. Ďalej sa zameriava na generovania dokumentácie, pričom kladie dôraz na možnosti užívateľského anotovania XML dát.

Abstract

The aim of this thesis is to design and develop a technical documentation generator that processes general XML input. The work thoroughly examines the structure of XML documents and defines the criteria that quality technical documentation should meet. Furthermore, it focuses on the generation of documentation, emphasizing the possibilities for user annotation of XML data.

Kľúčové slová

XML, dokumentácia, generátor, HTML, Python, lxml, Flask, SVD, UPPAAL, Draw.io

Keywords

XML, documentation, generator, HTML, Python, lxml, Flask, SVD, UPPAAL, Draw.io

Citácia

UHRECKÝ, Michal. *Multiplatformní nástroj pro generování technické dokumentace z XML.* Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Josef Strnadel, Ph.D.

Multiplatformní nástroj pro generování technické dokumentace z XML

Prehlásenie

Prehlasujem, že som tuto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Josefa Strnadela, Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Michal Uhrecký
6. mája 2024

Poďakovanie

V prvom rade by som chel poďakovať vedúcemu tejto práce za cenné rady a tipy. Ďalej by som chcel poďakovať mojim rodičom a priateľke Tereze Hermanovej za morálnu podporu počas tvorenia bakalárskej práce a veľká vďaka taktiež patrí všetkým testerom generátora.

Obsah

Kapitola 1

Úvod

V dnešnom rýchlo sa meniacom technologickom prostredí vznikajú neustále nové produkty a technológie, čo nevyhnutne vedie k zvýšenej potrebe ich kvalitného dokumentovania. Vzhľadom na svoju hierarchickú a detailne štruktúrovanú povahu, XML často obsahuje množstvo dát, ktoré poskytujú komplexný prehľad o štruktúrovaných informáciách. Dáta uložené v XML formáte môžu uchovávať rôzne typy informácií, napríklad sú v ňom uložené konfiguračné súbory pre softvérové aplikácie. Mať prehľadne zdokumentované XML dáta poskytuje čitateľom možnosť rýchlej orientácie v informáciách.

Technická dokumentácia je nevyhnutná súčasť každého vývojového projektu, produktu alebo služby, pretože zabezpečuje, že všetky aspekty technického procesu sú jasne a presne dokumentované. Prezeranie technickej dokumentácie by malo byť pre jej čitateľa príjemným zážitkom, ktorý umožňuje efektívne a pohodlné získavanie potrebných informácií. Aby sa to dosiahlo, dokumentácia by mala byť jasne štruktúrovaná, s logicky organizovaným obsahom, ktorý je rozdelený do prehľadných sekcií.

Problém s generovaním technickej dokumentácie spočíva v tom, že vstupný XML súbor môže obsahovať rôznu štruktúru a informácie, čo značne komplikuje automatizovaný proces transformácie dát do finálnej dokumentácie. Rôznorodosť štruktúr v XML súboroch vyžaduje flexibilný prístup pri ich spracovaní, aby bolo možné adekvátne zachytiť všetky relevantné informácie a zabezpečiť ich správne formátovanie v dokumentácii. Je dôležité aby mal používateľ k dispozícii prívetivé užívateľské rozhranie, ktoré mu dovolí doplniť XML dáta o ďalší kontext. Cieľom tejto práce je práve spraviť prieskum, akým spôsobom sa dáta v XML formáte ukladajú a následne navrhnúť a implementovať generátor, ktorý bude generovať prehľadnú dokumentáciu.

Práca sa začína teoretickým úvodom o tom, čo sú to vlastne dáta, v kapitole 2. V nasledujúcej kapitole 3 je podrobne rozobraná štruktúra XML dokumentov. Sú tu predstavené nástroje ako UPPAAL a Draw.io, vrátane ich konfiguračných XML súborov. Okrem toho sú v tejto kapitole predstavené aj SVD súbory, ktoré slúžia na popis registrov periférií mikrokontrolérov. V kapitole 4 sú rozoberané kvality, ktoré by mala technická dokumentácia spĺňať, a rôzne typy technickej dokumentácie, pričom súčasťou tejto kapitoly je aj porovnanie aktuálnych generátorov dokumentácie. Návrh generátora dokumentácie je detailne popísaný v kapitole 5. Kľúčové aspekty implementácie tohto navrhnutého generátora sú vysvetlené v kapitole 6. Záverečná kapitola 7 zahŕňa testovanie implementovaného generátora, ktoré som vykonal ja spolu s vybranými respondentmi.

Kapitola 2

Dáta

Práca sa zaoberá generovaním technickej dokumentácie z XML vstupu, respektíve čiastočne štruktúrovaných dát, ktoré sú vysvetlené v sekci 2.1. Slovník Merriam-Webster [3] definuje dáta takto:

- Dáta sú faktické informácie (napríklad merania alebo štatistiky), ktoré slúžia ako základ pre argumentáciu, diskusiu alebo výpočty.
- Dáta sú informácie v digitálnej forme, ktoré je možné prenášať alebo spracovávať.
- Dáta sú výstup informácií zo snímacieho zariadenia alebo orgánu, ktorý obsahuje užitočné aj irelevantné alebo nadbytočné informácie a musí byť spracovaný, aby mal zmysel.

V kontexte práce je uvažované o dátach ako o informáciach uložených v digitálnej forme, ktoré sú organizované.

Potreba ukladať dáta existuje už od počiatku ľudstva. Najskôr to boli zvyky nebezpečných zvierat či liečivé vlastnosti rastlín. Tieto informácie nebolo možné ukladať kvôli neexistujúcemu písmu a spôsobu uloženia. S vývojom ľudstva a potrebou ukladať čoraz viacej rôznych informácií začali vznikať písma aj spôsoby uloženia. Ako spôsoby uloženia sa používali hlinené tabuľky, papyrus, pergamen, či papier [25].

V modernej „počítačovej“ dobe je možné dáta ukladať do súborov. Aby sa s dátami efektívne pracovalo, tak je vhodné v nich mať poriadok. Správnym štruktúrovaním dát je možné docieliť poriadok v dátach. Vstup do môjho generátora technickej dokumentácie je práve XML súbor objasnený v sekci 3, ktorý v sebe obsahuje čiastočne štruktúrované dáta.

2.1 Čiastočne štruktúrované dáta

Čiastočne štruktúrované dáta a štrukturované dáta sú organizované dáta. Štruktúrované dáta majú tabuľkovú štruktúru, ktorú používajú napríklad relačné databázy. Čiastočne štrukturované dáta sú odlišné v tom, že nemajú preddefinovanú schému a sú tým pádom flexibilnejšie pre pridávanie nových atribútov. Tiež sa odlišujú v dátovej štruktúre, pričom štrukturované dáta majú plochú tabuľkovú štruktúru, zatiaľ čo čiastočne štrukturované dáta sú usporiadané do hierarchie s vnorenými informáciami, čo je ideálne pre dáta získané z aplikácií. Čiastočne štrukturované dáta obsahujú metadáta a značky na identifikáciu konkrétnych vlastností dát a na organizáciu dát do záznamov. Oproti neštruktúrovaným dátam je v nich jednoduchšie vyhľadávať, a tým pádom s nimi efektívnejšie pracovať. Medzi najznámejšie príklady čiastočne štruktúrovaných dát patria XML, JSON a CSV [20] [8].

Kapitola 3

XML

XML, celým názvom *Extensible Markup Language*, je štandard podporovaný organizáciou W3C na značkovanie dokumentov. Definuje univerzálnu syntax na označovanie dát jednoduchými, ľudsky čitateľnými značkami. Poskytuje štandardný formát pre počítačové dokumenty, ktorý je dostatočne flexibilný, aby sa prispôsobil rôznym oblastiam, ako sú napríklad webové stránky, elektronické výmeny dát, vektorová grafika, vzdialené volania procedúr a ďalšie. Pri vývoji programu je výhodné interagovať s dátami v XML dokumentoch, vďaka tomu, že existuje široká škála bezplatných knižníc v programovacích jazykoch, ktoré vedia čítať a zapisovať XML [12].

3.1 Formát XML

Na prvý pohľad značkovanie v XML veľmi pripomína značkovanie v HTML, ale existujú kľúčové rozdiely. Dôležité je, že XML je metaznačkovací jazyk. To znamená, že nemá pevne stanovenú sadu značiek a elementov, ktoré majú fungovať pre každého vo všetkých oblastiach záujmu. Na rozdiel od HTML XML umožňuje vývojárom vytvárať elementy podľa potreby. Chemici môžu používať elementy, ktoré popisujú molekuly, atómy, väzby, reakcie a ďalšie pojmy z chémie. Realitní makléri môžu využívať prvky, ktoré popisujú byty, nájmy, provízie, lokality a ďalšie informácie potrebné pre nehnuteľnosti. „X“ v XML znamená rozšíriteľné (*extensible*). Rozšíriteľné znamená, že jazyk môže byť rozšírený a prispôsobený tak, aby vyhovoval mnohým rôznorodým potrebám [12].

Najzákladnejšou stavebnou jednotkou XML dokumentu je element. Element sa skladá z otváracej značky, tzv. *tagu*, ktorý nasleduje obsah elementu, a tiež ukončujúceho tagu [12].

```
<person>
  Alan Turing
</person>
```

Obr. 3.1: *Jednoduchý XML dokument* [12].

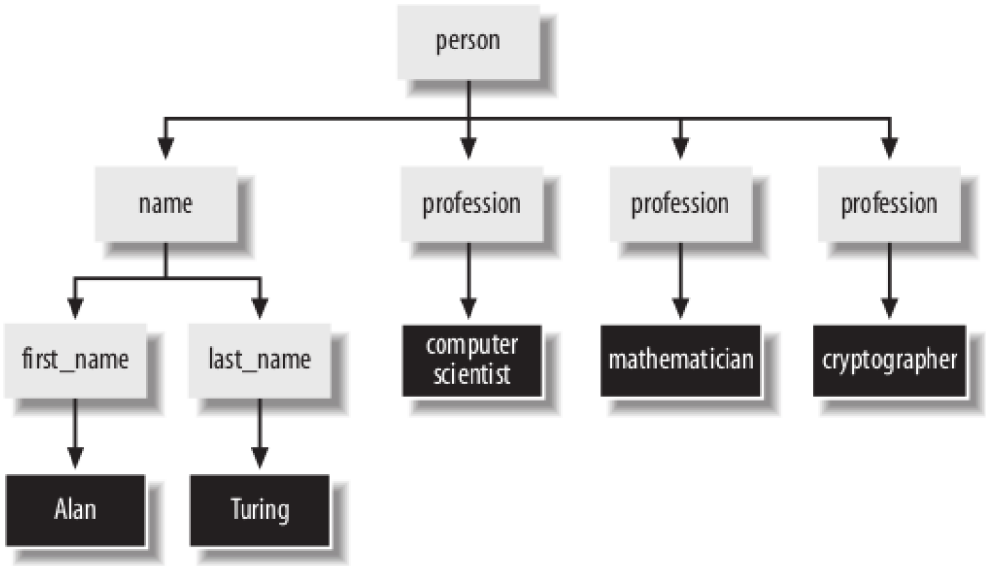
V tomto prípade je `<person>` začiatočný tag elementu, „Alan Turing“ je obsah elementu a `</person>` je ukončovací tag elementu. Na prvý pohľad elementy pripomínajú HTML, začiatočné tagy začínajú s `<` a ukončovacie tagy začínajú s `</`. Obidva sú nasledované názvom elementu a uzavreté znakom `>`. Avšak na rozdiel od HTML elementov, v XML je dovolené vytvárať nové elementy podľa potreby. Na popis osoby sa môže použiť napríklad element s tagmi `<person>` a `</person>`. Na popis kalendára sa zasa môže použiť napríklad

element s tagmi `<calendar>` a `</calendar>`. Existuje aj špeciálny tag pre prázdne elementy, respektíve elementy, ktoré nemajú obsah. Takýto element začína s `<` a končí s `/>`. Prázdny element `person` by sa zapísal `<person/>`. V XML sú názvy elementov citlivé na veľkosť písma, takže element s tagom `<person>` nie je to isté ako element s tagom `<Person>`. Tieto elementy je možné do seba vnárať a tým pádom vytvárať hierarchickú stromovú štruktúru dokumentu, ktorú je možné vidieť na príklade 3.2 [12].

```
<person>
  <name>
    <first_name>Alan</first_name>
    <last_name>Turing</last_name>
  </name>
  <profession>computer scientist</profession>
  <profession>mathematician</profession>
  <profession>cryptographer</profession>
</person>
```

Obr. 3.2: *XML dokument so stromovou štruktúrou* [12].

XML dokument sa skladá iba z jedného elementu `person`, rovnako ako aj ten predošlý. Avšak v tomto prípade obsahuje aj zanorené elementy `name` a `profession`. Pre element `person` sú elementy `name` a `profession` synovské, a pre ne je zasa element `person` otcovský. Element `name` obsahuje synovské elementy `first_name` a `last_name`. Pre elementy `first_name` a `last_name` je otcovským elementom `name`. Každý jeden XML dokument má element, ktorý nemá otca, a to je koreňový element. Koreňový element v sebe obsahuje všetky ostatné elementy. V predošlých príkladoch 3.1 a 3.2 je koreňovým elementom `person`. Ak je začiatočný tag elementu vnorený v inom elemente, potom jeho koncový tag musí byť taktiež vnorený v tomto elemente. Prekrývajúce sa tagy, ako v príklade `<strong><em>bežný príklad z HTML</strong></em>`, sú v XML zakázané [12].



Obr. 3.3: *Zobrazený strom XML dokumentu* 3.2 [12].

XML elementy môžu obsahovať atribúty. Atribúty tvoria páry kľúč a hodnota priradené k otváraciemu tagu elementu. Atribút začína názvom, pokračuje znakom „rovná sa“ a nakoniec sa píše hodnota, ktorá je uzavretá v jednoduchých alebo dvojitých úvodzovkách. Ako je možné zapísať atribút k elementu je možné vidieť v príklade 3.4 [12].

```
<person born="1912-06-23" died="1954-06-07">
  Alan Turing
</person>
```

Obr. 3.4: *Jednoduchý XML dokument s atribútmi* [12].

XML dokument obsahuje element `person`, ktorý má atribúty `born` a `died`. Atribút born má hodnotu `1912-06-23` a atribút `died` má hodnotu `1954-06-07`. Poradie atribútov v XML dokumente nie je dôležité. XML dokument v príklade 3.2 je zapísaný čisto pomocou elementov, ale niektoré jeho elementy by mohli byť zapísané aj ako atribúty, ako je to v príklade 3.5 [12].

```
<person>
  <name first="Alan" last="Turing"/>
  <profession value="computer scientist"/>
  <profession value="mathematician"/>
  <profession value="cryptographer"/>
</person>
```

Obr. 3.5: *XML dokument ako v príklade 3.2 zapísaný pomocou atribútov* [12].

V príklade 3.5 sú zapísané rovnaké informácie ako v príklade 3.2, ale tentokrát sú použité atribúty na uchovanie informácií. Logicky potom prichádza otázka, kedy je vhodné informácie uchovávať pomocou elementov a kedy je vhodné použiť atribúty. Niektorí informatici tvrdia, že atribúty sú vhodné na ukladanie metadát o elemente a samotnú informáciu je lepšie uchovávať v elemente. Každopádne pri navrhovaní spôsobu ukladania informácií je potrebné si uvedomiť, že elementy sú rozšíriteľné o atribúty a ďalšie elementy, zatiaľ čo hodnota atribútu je iba text, ktorý už nie je možné ďalej rozširovať [12].

XML dokumenty by mali začínať (nie je to povinné, ale je to vhodné) deklaráciou. Deklarácia začína postupnosťou znakov `<?xml` a končí `?>`. Medzi začiatočnými a koncovými znakmi obsahuje 3 atribúty a to sú `version`, `encoding` a `standalone` [12].

```
<?xml version="1.0" encoding="ASCII" standalone="yes"?>
<person>
  Alan Turing
</person>
```

Obr. 3.6: *Jednoduchý XML dokument s deklaráciou* [12].

Atribút `version` by vo väčšine prípadov mal byť nastavený na hodnotu `1.0`. Môže nadobúdať ešte hodnotu `1.1`, ale jej nastavením je dokument obmedzený iba na najnovšie verzie syntaktických analyzátorov a pokiaľ sa nejedná o výnimočný prípad, kde by sa verzia `1.1` hodila, tak je lepšie ponechať verziu `1.0`. V atribúte `encoding` je zapísané kódovanie

daného XML dokumentu. Štandardne sú XML dokumenty zakódované vo formáte `UTF-8` s premenlivou dĺžkou znakovej sady Unicode. Ide o striktnú nadmnožinu `ASCII`, takže čisto textové súbory `ASCII` tiež spadajú pod formát `UTF-8`. Atribút `standalone` môže nadobúdať hodnoty `yes` alebo `no`. Ak je hodnota nastavená na `no`, aplikácia, ktorá spracováva XML dokument, môže byť požiadaná čítať externý DTD (*Document Type Definition*) súbor 3.2. Atribút `standalone` je v deklarácii XML nepovinný. Ak je vynechaný, potom sa predpokladá hodnota `no` [12].

Každý dokument XML musí byť správne naformátovaný. To znamená, že musí dodržať niekoľko pravidiel, vrátane nasledujúcich [12]:

- Každý otvárací tag musí mať odpovedajúci koncový tag.
- Elementy môžu byť vnorené, ale nesmú sa prekrývať.
- Musí existovať práve jeden koreňový element.
- Hodnoty atribútov musia byť uvedené v úvodzovkách.
- Element nesmie mať dva atribúty s rovnakým názvom.
- Komentáre a procesové inštrukcie[1] sa nesmú vyskytovať vo vnútri značiek.
- Všetky problémové znaky, ako sú napríklad `<` a `&` (špeciálne znaky v kontexte XML, ktoré sa používajú priamo v syntaxi jazyka), potrebujú byť nahradené.

Tento zoznam nie je úplný. Existuje mnoho spôsobov, ako môže byť dokument nesprávne formátovaný. Kompletný list obsahuje rôzne ďalšie konštrukcie XML a je možné ho nájsť v knihe „XML in a Nutshell“ [12].

3.2 Validácia XML

Hoci je XML extrémne flexibilný, nie všetky programy, ktoré čítajú konkrétne XML dokumenty, umožňujú takúto flexibilitu. Mnoho programov môže pracovať len s niektorými XML aplikáciami, ale nie s inými. Napríklad Adobe Illustrator dokáže čítať a zapisovať súbory vo formáte *Scalable Vector Graphics* (SVG), ale nemožno očakávať, že by porozumel dokumentu *Platform for Privacy Preferences* (P3P). V rámci konkrétnej XML aplikácie je často dôležité zabezpečiť, aby daný dokument dodržiaval pravidlá tejto XML aplikácie. XML 1.0 poskytuje riešenie tejto dilemy, a to zavedením definície typu dokumentu (DTD - *Document Type Definition*). Definícia typu dokumentu je napísaná vo formálnej syntaxi, ktorá presne vysvetľuje, ktoré elementy sa môžu v XML dokumente nachádzať a aký obsah a atribúty majú tieto elementy. DTD môže uvádzať napríklad, že „každý element `zamestnanec` musí mať atribút `social_security_number`“ alebo „element `ul` môže obsahovať len elementy `li`“. Rôzne XML aplikácie môžu používať rôzne DTD na špecifikovanie toho, čo povolujú a nepovolujú. V praxi syntaktický analyzátor porovná dokument s jeho DTD a zistí, či dokument spĺňa alebo nespĺňa predpis podla DTD. Program potom môže rozhodnúť, ako sa má zachovať v prípade nezrovnalostí [12].

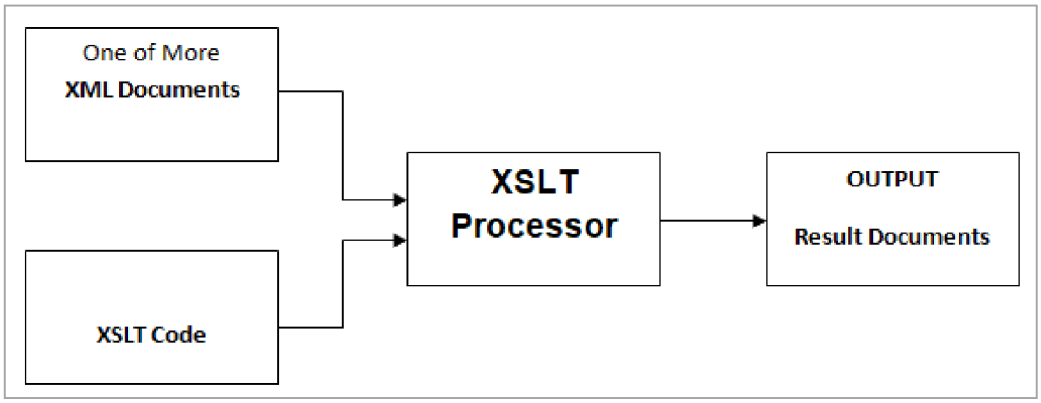
Definície typu dokumentu môžu uplatňovať základné štrukturálne pravidlá na dokumenty, no mnoho aplikácií potrebuje komplexnejšie overenie XML dokumentu. Preto W3C vyvinulo *XML Schema Recommendation*. Schémy môžu popisovať komplexné obmedzenia

[1] Procesové inštrukcie sa používajú pre zápis vo formáte, ktorý potrebuje externá aplikácia, viac v [12].

na prvky a atribúty. Na validáciu XML dokumentu môže byť kombinovaných viacero schém. Ďalšou výhodou je, že XML Schema je napísaná v jazyku XML, čo uľahčuje jej vytvorenie pre tých, ktorí už majú skúsenosti s XML. [12].

3.3 Vizualizácia XML

XML dokumenty samy o sebe nesú len dáta a neobsahujú informácie o tom, ako by sa tieto dáta mali vizuálne prezentovať. Na definovanie vzhľadu a formátovania dát v XML existuje špeciálny nástroj, známy ako XSLT (*Extensible Stylesheet Language Transformations*). XSLT je jazyk, ktorý umožňuje transformovať XML dáta do rôznych výstupných formátov, vrátane HTML, PDF alebo dokonca iných XML súborov. Proces transformácie vykonáva tzv. XSLT procesor, ako je napríklad Saxon a Xalan. XSLT procesor načíta jeden alebo viacero XML dokumentov spolu s XSLT štýlovou predlohou, ktorá definuje pravidlá transformácie. Výsledkom tejto transformácie je nový dokument, ktorý je v súlade s definovanými pravidlami v XSLT predlohe. Týmto spôsobom je možné mať jeden XML dokument, ktorý uchováva špecifický typ dát, a zároveň ho zobrazovať rôznymi spôsobmi v závislosti od potrieb. Tento prístup umožňuje flexibilné prezentácie rovnakých dát pomocou rôznych XSLT šablón, ktoré transformujú pôvodné XML do rôznych formátov [34] [31].



Obr. 3.7: *Princíp spracovania XML dokumentov pomocou XSLT štýlov* [31].

3.4 Dostupné nástroje pre prácu s XML

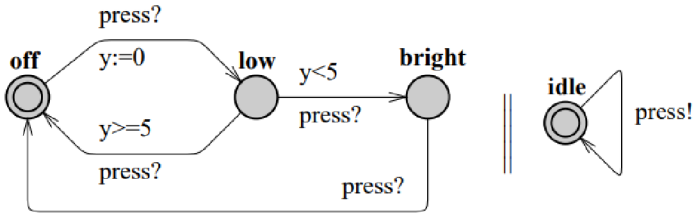
Na výber existuje niekoľko populárnych programovacích jazykov, ktoré poskytujú širokú škálu knižníc pre manipuláciu s XML dokumentmi. Jeden z aktuálne populárnych jazykov je Java, ktorá podporuje technológiu JAXB (*Java Architecture for XML Binding*), ktorá umožňuje mapovanie Java objektov na XML dokumenty a naopak. Hlavným účelom JAXB je jednoduché prevádzanie medzi XML dokumentmi a objektami v Jave a tým uľahčenie práce s XML v aplikáciách napísaných v Jave. JAXB poskytuje nástroje na generovanie Java tried z existujúcich XML schém (XSD) a naopak. Tieto generované triedy potom umožňujú jednoduchú prácu s XML dokumentmi pomocou bežných Java objektov. JAXB rieši konverziu medzi reprezentáciou dát v Jave a ich XML formátom. Navyše, JAXB umožňuje aj anotácie Java tried pre presnejšie riadenie toho, ako majú byť objekty mapované na XML a naopak. Týmto spôsobom je možné ovplyvniť štruktúru generovaných XML dokumentov a programátorovi to dáva určitú flexibilitu. Ďalšou technológiou je *Java API*

for XML Processing (JAXP), ktoré je určené na spracovanie XML dát pomocou aplikácií napísaných v programovacom jazyku Java. JAXP využíva štandardy parserov *Simple API for XML Parsing* (SAX[2]) a *Document Object Model* (DOM), čo umožňuje spracovať dáta buď ako postupnosť udalostí (*stream of events*) alebo vytvárať objektovú reprezentáciu. JAXP tiež podporuje štandard XSLT a tým pádom posyktuje kontrolu nad reprezentáciou dát a umožňuje konvertovať dáta do iných XML dokumentov alebo iných formátov, ako napríklad HTML. V programovacom jazyku Python sa používajú na spracovanie XML dva nástroje, a to ElementTree a lxml. Je prirodzené reprezentovať XML dokument ako strom, a to práve robí knižnica ElementTree. Pre tento účel používa dve triedy – ElementTree, čo je trieda reprezentujúca celý XML dokument ako strom, a triedu Element reprezentujúcu jeden uzol v strome. V praxi sa načíta vstupný XML dokument, z ktorého sa získa koreňový element, a ten obsahuje odkaz na svojich synov, a tí zasa na svojich synov atď. a tým pádom je možné prechádzať celý XML dokument. Pokročilá knižnica lxml pre spracovanie XML a HTML je postavená na C knižniciach libxml2 a libxslt. Je jedinečná tým, že kombinuje rýchlosť a úplnosť XML týchto knižníc s jednoduchosťou natívneho *Python API ElementTree*, s ktorým je väčšinou kompatibilná. Knižnica lxml rozširuje ElementTree o možnosť práce s technológiami XPath, Relax NG, XML Schema, XSLT a c14n. V JavaScripte je možné využiť DOMParser, ktorý umožňuje spracovať reťazec obsahujúci XML alebo HTML a vytvoriť z neho DOM – dokument, ktorý umožňuje manipuláciu s obsahom pomocou stromovej štruktúry. V JavaScripte je prípadne možné využiť knižnicu xml2js, ak je potrebné prevádzať XML dokument na JSON (*JavaScript Object Notation*), keďže táto knižnica poskytuje funkcie na parsovanie XML dát a prevod ich štruktúry na objekty JavaScriptu a naopak. V jazyku C# je najvýznamnejšia knižnica System.Xml, ktorá ponúka sadu tried a funkcií pre prácu s XML v jazyku C#. Táto knižnica umožňuje vytvárať, spracovávať a manipulovať s XML dokumentmi. Medzi najvýznamnejšie triedy tejto knižnice patria XmlReader a XmlWriter, ktoré umožňujú čítanie a zápis XML dát. Trieda XmlDocument reprezentuje celý XML dokument a umožňuje programovo vytvárať, načítať a ukladať XML dokumenty. XmlElement reprezentuje jeden element v XML dokumente a pomocou tejto triedy je možné pristupovať k atribútom a hodnotám elementu [18] [14] [29] [19] [5] [33] [28].

3.5 Testovacie XML dokumenty

Ako už bolo spomenuté, XML má rozsiahle využitie v praxi – dokáže napríklad uchovávať nastavenia projektov vývojových prostredí, medzi ktoré sa radí sada nástrojov UPPAAL. Táto sada slúži na verifikáciu systémov reálneho času reprezentovaných časovanými automatmi, ktoré sú rozšírené o celočíselné premenné, štruktúrované dátové typy a synchronizáciu kanálov. Tento nástroj vyvinuli spoločne Uppsala University a Aalborg University. Časovaný automat je konečný stavový automat rozšírený o hodinové premenné. Používa model hustého času, kde sa hodinová premenná vyhodnocuje ako reálne číslo. Všetky hodiny postupujú synchronne. V aplikácii UPPAAL je systém modelovaný ako sieť niekoľkých paralelných časových automatov [2] [4].

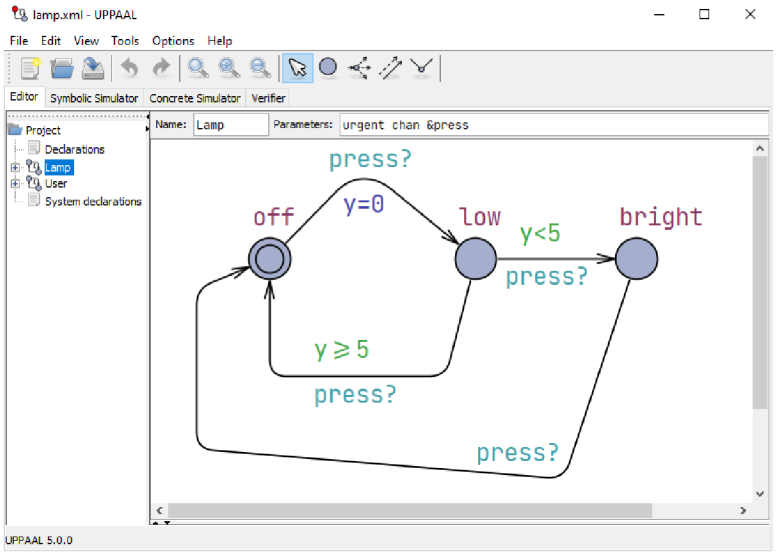
[2]SAX je v súčasnej dobe implementovaný v niekoľkých programovacích jazykov, vrátane Pythonu, JavaScriptu, C++ a C#.



(a) Lamp. (b) User.

Obr. 3.8: *Príklad lampy modelovaný časovanými automatmi* [2].

Na obrázku 3.8 je znázornený časovaný automat, ktorý modeluje systém s lampou a používateľom. Lampa sa môže nachádzať v troch stavoch: `off`, `low` a `bright`. Ak používateľ stlačí tlačidlo, synchronizuje sa s kanálom `press`, ktorý dá lampe správu, aby sa zapla a prešla do stavu `low`. Ak používateľ stlačí tlačidlo znova, lampa sa vypne a prejde do stavu `off`. Ak však používateľ rýchlo dvakrát stlačí tlačidlo, lampa sa rozsvieti veľmi silno a prejde do stavu `bright`. Užívateľ môže tlačidlo stlačiť kedykoľvek alebo ho nemusí stlačiť vôbec. Hodiny lampy sa používajú na zistenie, či bol používateľ rýchly (`y < 5`) alebo pomalý (`y >= 5`) [2].



Obr. 3.9: *Príklad lampy v aplikácii UPPAAL.*

UPPAAL podporuje tri formáty ukladania projektu – XML, XTA a TA. Jedny z testovacích dát budú práve XML súbory exportované aplikáciou UPPAAL, ktoré obsahujú časované konečné automaty reprezentujúce reálne systémy [30]. Pri skúmaní exportovaného XML dokumentu z príkladu 3.9 zisťujem, že celý XML dokument je obalený v koreňovom elemente `<nta>`. Ďalej obsahuje synovské elementy `<declaration>`, `<template>`, `<system>` a `<queries>`. V elemente `<declaration>` sú ukladané globálne premenné. V elemente `<template>` sa ukladajú jednotlivé komponenty systému, respektíve popisy časovaných automatov. Interakcie medzi jednotlivými komponentami sú uložené v elemente `<system>`. Otázky na overovanie vlastností modelu sú uložené v elemente `<queries>`. V elemente

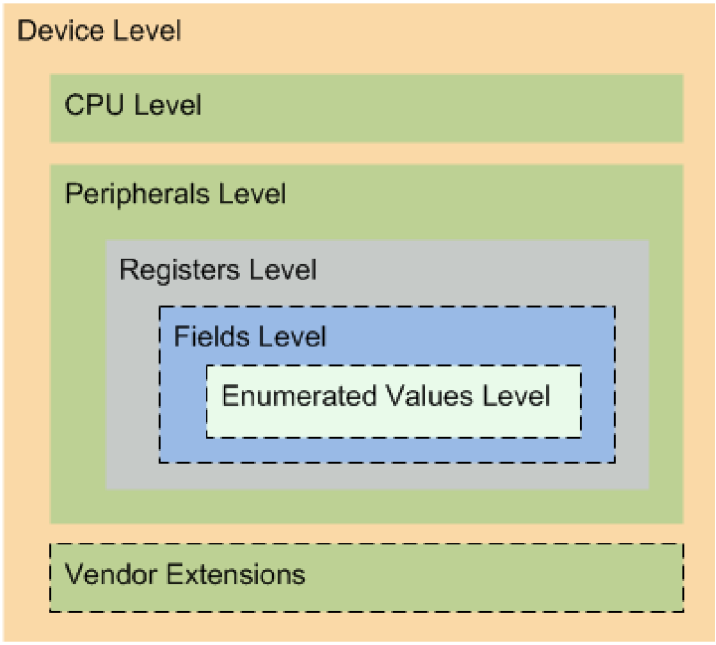
`<template>` sa môže nachádzať element `<declaration>`, ktorý popisuje lokálne premenné danej šablóny, respektíve časovaného konečného automatu. Ďalej sa v ňom môžu nachádzať elementy ako `<location>`, ktorý popisuje stavy automatu, alebo `<transition>`, ktorý popisuje prechody medzi stavmi, s atribútmi `x` a `y` popisujúcimi polohu daných elementov v projekte. Úplný predpis štruktúry XML dokumentu nástroja UPPAAL je možné nájsť v DTD súbore[3].

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE nta PUBLIC ... >
<nta>
  <declaration>
  </declaration>
  <template>
    <name x="5" y="5">Lamp</name>
    <parameter>urgent chan &amp;press</parameter>
    <declaration> // Place local declarations here.
      clock y;
    </declaration>
    <location id="id0" x="0" y="0">
      <name x="-10" y="-34">off</name>
    </location>
    ...
</nta>
```

Obr. 3.10: *Časť XML dokumentu príkladu lampa 3.9.*

Medzi ďalšiu testovaciu sadu patria XML dokumenty SVD. Formát *Cortex Microcontroller Software Interface Standard System View Description* (CMSIS-SVD) formálne popisuje systém obsiahnutý v mikrokontroléroch založených na procesoroch Arm Cortex-M, konkrétne pamäťovo mapované registre periférií. Detaily obsiahnuté v popisoch systému sú porovnateľné s údajmi v manuáloch k zariadeniam. Informácie sa pohybujú od vysokoúrovňových funkčných popisov periférií až po definíciu a účel jednotlivých bitových polí v pamäťovo mapovanom registri. Súbory CMSIS-SVD sú vyvíjané a udržiavané výrobcami čipov. Títo výrobcovia distribuujú svoje popisy ako súčasť balíkov *CMSIS Device Family Packs*. Nástrojoví výrobcovia používajú súbory CMSIS-SVD na poskytovanie zariadeniu špecifických pohľadov na periférie v ich debuggeroch. Hlavičkové súbory zariadení kompatibilné s CMSIS sú generované zo súborov CMSIS-SVD [27].

[3]Odkaz na DTD súbor verzie 1.6: http://www.it.uu.se/research/group/darts/uppaal/flat-1_6.dtd

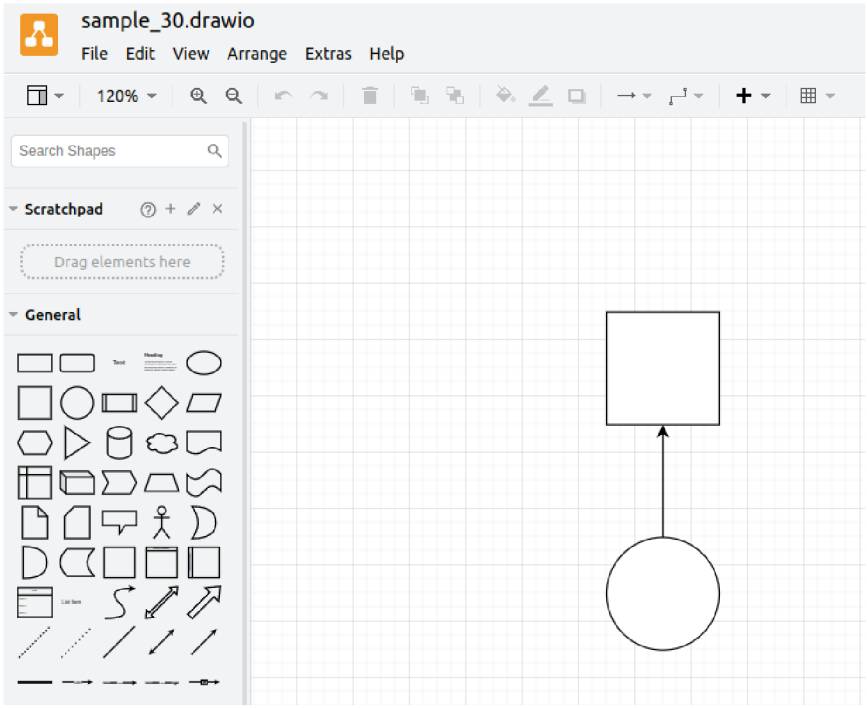


Obr. 3.11: *Hierarchia XML dokumentu CMSIS-SVD* [26].

Jeden súbor CMSIS-SVD obsahuje popis jedného zariadenia. Hierarchia XML súboru je ukázaná na obrázku 3.11. Zariadenie pozostáva z procesora a aspoň jedného periférneho zariadenia. Každé periférne zariadenie obsahuje aspoň jeden register. Register môže pozostávať z jedného alebo viacerých polí (*fields*). Rozsah hodnôt pre pole možno ďalej opísať vymenovanými (*enumerated*) hodnotami. Súbor na konci obsahuje časť *vendor extensions*, ktorá umožňuje rozšíriť zariadenie o dodatočnú špecifikáciu. Kompletný predpis pre SVD súbor je možné vidieť v jeho schéme[4] [26].

Ako poslednú testovaciu sadu som si zvolil uložené projekty online aplikácie Draw.io. Draw.io je softvér pre vytváranie diagramových aplikácií a najviac používaný diagramovací softvér pre koncových používateľov založený na prehliadači. Softvér umožňuje obrovský výber diagramov, ktoré sa dajú v projekte uložiť, a s nimi aj veľké množstvo nastavení. Umožňuje export diagramov do XML súborov [1] [7].

[4] https://www.keil.com/pack/doc/CMSIS/SVD/html/schema_1_2_gr.html



Obr. 3.12: *Online aplikácia Draw.io.*

```
<?xml version="1.0" encoding="utf-8"?>
<mxfile host="app.diagrams.net" modified="2023-12-11T19:40:33.581Z" ... >
<diagram name="Page-1" id="innbUz3WiTEPd1axAbeo">
  <mxGraphModel dx="1434" dy="725" grid="1" gridSize="10" guides="1" ... >
    <root>
      <mxCell id="0" />
      <mxCell id="1" parent="0" />
      <mxCell id="l-zQ8Usf_r2dOD8fP2Bd-3" style=" ... " ... >
        <mxGeometry relative="1" as="geometry" />
      </mxCell>
      <mxCell id="l-zQ8Usf_r2dOD8fP2Bd-1" value="" style=" ... " ... >
        <mxGeometry x="380" y="340" width="80" height="80" as="geometry" />
      </mxCell>
      <mxCell id="l-zQ8Usf_r2dOD8fP2Bd-2" value="" style=" ... " ... >
        <mxGeometry x="380" y="180" width="80" height="80" as="geometry" />
      </mxCell>
    </root>
  </mxGraphModel>
</diagram>
</mxfile>
```

Obr. 3.13: *Časť XML dokumentu exportovaného z diagramu v obrázku* 3.12.

Kapitola 4

Technická dokumentácia

Technická dokumentácia predstavuje akýkoľvek text, ktorý popisuje aplikáciu, jej účel, vytvorenie alebo architektúru produktu alebo služby. Cieľom technickej dokumentácie je vysvetliť to, čo organizácia ponúka. Existuje niekoľko typov technických dokumentov, zameraných na rôzne cieľové skupiny. Písanie technických dokumentov je obvykle zodpovednosťou technických spisovateľov, projektových manažérov, členov vývojového tímu alebo odborníkov na daný produkt alebo službu. Efektívne napísaná technická dokumentácia môže pomôcť cieľovej skupine porozumieť fungovaniu produktu a jeho potrebným detailom [17].

Veľmi dôležitá je samotná kvalita technickej dokumentácie. Vysoko kvalitná technická dokumentácia je [11]:

- Presná
- Úplná
- Použiteľná
- Jasne napísaná
- Ľahko čitateľná
- Logicky prezentovaná
- Stručná
- Napísaná vhodným jazykom
- Gramaticky správna
- Vhodná obsahom a rozsahom
- Prezentovaná v atraktívnom balení

Presný dokument neobsahuje chyby vo faktách, ktoré by mohli zmiasť čitateľa. Dobrým spôsobom, ako tomu predísť, je napríklad nakresliť diagram toku. Úplný dokument nevynecháva nič, čo je dôležité pre čitateľa. Napríklad pri referenčnom manuále je dôležité, aby dokument obsahoval všetky kroky a nevynechal žiadny. Použiteľný dokument je taký, ktorý váš čitateľ môže ľahko použiť - nie je príliš objemný alebo navrhnutý tak, že váš čitateľ musí vyvinúť extra úsilie na nájdenie informácií. V dokumente je veľmi dôležitá organizácia informácií, v prípade zlej organizácie bude koncový používateľ veľmi frustrovaný. Ak dokument nie je jasne napísaný, používatelia mu nerozumejú, preto je dôležité sa vyhýbať

nejednoznačnostiam v technickej dokumentácii. Aby bol dokument čitateľný, je potrebné poznať, pre koho sa dokument píše, v prípade používateľa začiatočníka je potrebné vysvetliť termíny, zatiaľčo pri skúsenejšom používateľovi je možné ich vynechať. Dokumentácia by mala logicky postupovať od začiatku do konca alebo od jednoduchšieho k zložitejšiemu, prípadne by mal byť využitý iný logický postup, ktorý dáva v danom kontexte zmysel. Kvalitná dokumentácia by nemala obsahovať nadbytočné vety a slová, je dôležité sa vyhnúť zbytočným frázam a udržať dokumentáciu čo najstručnejšiu. Aby bola dokumentácia stručná, je vhodné používať krátke slová namiesto dlhých, ak je to možné. Zaujímavou technikou je jednostranné zobrazenie, kde sa autor dokumentácie snaží na jednu stranu zmestiť všetky informácie, čo môže viesť k vytvoreniu diagramov, ktoré vylepšia dokumentáciu. Vybrať vhodný jazyk pre dokumentáciu môže byť kľúčové. Pri tvorbe reklamného letáku je dôležité uvažovať, že zaujatie čitateľa trvá približne 20 sekúnd a preto je dôležité vybrať vhodné pútavé slová. Väčšina technickej dokumentácie je ale písaná formálnym štýlom a je vhodné sa preto vyhnúť skratkám a slangu. Ak výsledná dokumentácia bude obsahovať príliš veľa gramatických chýb, odradí to čitateľa. Je vhodné sa vyvarovať aj pravopisným a interpunkčným chybám. Vhodný rozsah a obsah pri technickej dokumentácii sa nastavuje podľa čitateľa; začiatočníka netreba zaťažiť zbytočnými podrobnosťami, ktoré sú pre experta potrebné, a experta zasa odradiť čítaním pre neho jasných vecí. Keď už je technická dokumentácia perfektne napísaná, treba ju zabaliť do atraktívneho balenia. V prípade tlačenej knihy je balenie veľmi podstatné, kedže jeho nedotiahnutie môže spôsobiť, že kvalitne napísaná kniha nebude vôbec čítaná. Je náročné dodržať všetky vlastnosti kvalitnej technickej dokumentácie. Preto sú potrební testeri a kritici, ktorí poskytnú potrebnú spätnú väzbu [11].

4.1 Kategórie

Existujú tri typy technickej dokumentácie: marketingové materiály, reporty (materiály, ktoré informujú) a inštruktážne materiály. Marketingové alebo predajné materiály majú za účel presvedčiť. Reporty uvádzajú fakty bez presvedčovacieho alebo inštruktážneho prístupu. Inštruktážne materiály môžu popisovať jednotlivé inštrukcie a tiež aj školiť. Medzi inštruktážne materiály patria tradičné dokumenty, ktoré opisujú produkt pre používateľa. Tabuľka 4.1 ukazuje technické dokumenty, ktoré spadajú do týchto troch kategórií [11].

Marketing	Reporty	Inštruktážne materiály
brožúra	článok v časopise	návod pre zákazníka
prípadová štúdia	článok v novinách	používateľská príručka
predajný leták	žurnálový článok	návod na použitie
tlačová správa	interná publikácia	príručka pre prípravu miesta
príručka produktu	technický článok	inštalačná príručka
katalóg produktov	správa o pokroku	referenčná príručka
marketingový skript	interná správa	údržbový manuál
jazyk marketingu	ročná správa	manuál systémového manažéra
prezentácia pre účely predaja	plán (*blueprint*)	manuál operátora
technický súhrn		technický popis
reklamný text		funkčná špecifikácia
biela kniha		špecifikácia užívateľského rozhrania
„*mock*“ papier		glosár
odporúčanie		vzdelávacia príručka
informačný list		príručka pre rýchly štart
stručný popis produktu		prezentácie
sprievodca použitím		študijné materiály

Tabuľka 4.1: *Typy technických dokumentácií* [11].

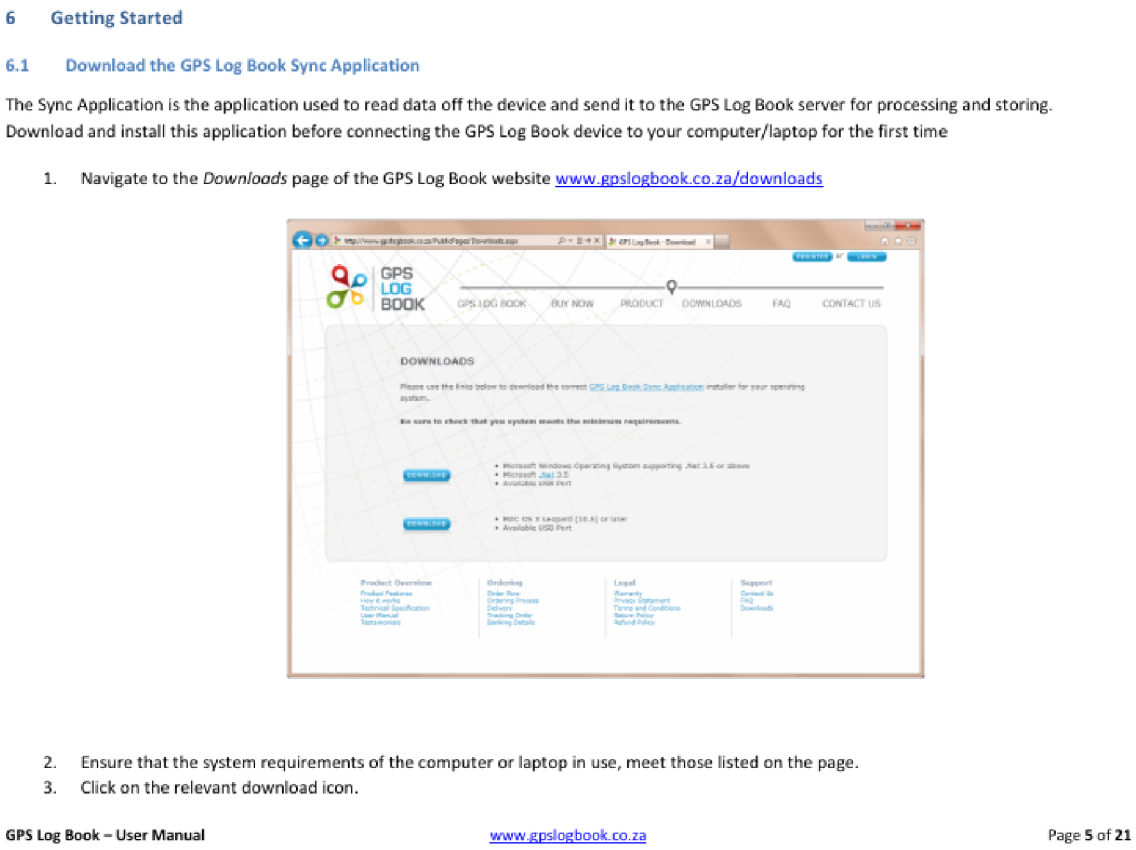
Používateľské príručky sú bežnou formou používateľskej dokumentácie, ktorá vysvetľuje používateľom, ako produkt funguje. Sú užitočné počas procesu zaučovania a sú písane interaktívnym spôsobom, kde sa vysvetluje štýlom „krok za krokom“. Tieto príručky využívajú jednoduchý jazyk a ukážky na vysvetlenie zložitých funkcií a riešenie bežných problémov, čím sú ľahko pochopiteľné pre začiatočníkov. Používateľská príručka je navrhovaná s ohľadom na potreby koncového používateľa, pričom je snaha uvažovať a predvídať jeho potreby a skúsenosti tak, aby bolo predídené frustrácii používateľa [21]. Používateľská príručka môže obsahovať: [22]

- **Úvod do produktu:** Poskytuje používateľom prehľad toho, čo môžu od príručky očakávať, ako napríklad určenie účelu produktu alebo súhrn tém pokrytých v príručke.
- **Inštalácia a nastavenie:** Poskytuje postupné inštrukcie vedúce používateľov k pripraveniu produktu na použitie, jeho integrovaniu s inými produktmi a podobne.
- **Funkcie a prípady použitia produktu:** Vysvetľuje možnosti produktu a spôsoby ich maximalizácie pre rôzne skupiny používateľov.
- **Ako pracovať s produktom:** Popisuje hlavné a pokročilé funkcie produktu. Na rozdiel od inštrukcií na inštaláciu a nastavenie, táto časť si kladie za cieľ zoznámiť používateľov s produktom a vzdelávať ich o každodenných činnostiach.
- **Tipy na odstránenie problémov:** Učí používateľov, ako riešiť každodenné výzvy alebo chyby, ktoré sa môžu objaviť počas inštalácie, nastavenia alebo bežného používania.
- **Často kladené otázky (FAQ):** Zahrňuje všetky bežné otázky, s ktorými sa používateľ môže stretnúť pri zoznamovaní sa s produktom.

- **Glosár:** Zoznamuje používateľov s všetkými odbornými výrazmi alebo technickými pojmami uvedenými v príručke, ktoré by používatelia nemuseli hneď pochopiť.

Biela kniha je detailná správa alebo sprievodca o konkrétnej téme. Slúži na presvedčenie čitateľov o odbornosti a jemne naznačuje, že konkrétny produkt je najlepším riešením problému. Pri písaní bielej knihy je podstatné sa sústrediť na poskytovanie hodnoty vrátane originálnych údajov a odborných analýz, a nie na predaj produktu. Biela kniha je pre marketing dôležitá, aj keď nič nepredáva, lebo buduje dôveru v danú značku [21].

Prípadová štúdia je vynikajúci spôsob, ako potenciálnemu zákazníkovi dokázať hodnotu produktu, pretože ukazuje, ako produkt pomohol konkrétnemu zákazníkovi dosiahnuť požadované výsledky. Na vytvorenie prípadovej štúdie je potrebný rozhovor so súčasným alebo minulým zákazníkom. Rozhovor by mal zahŕňať otázky, ktoré pomôžu získať konkrétne čísla, aby bolo jasné, že spoločnosť dosiahla výsledky [21].



6 Getting Started

6.1 Download the GPS Log Book Sync Application

The Sync Application is the application used to read data off the device and send it to the GPS Log Book server for processing and storing. Download and install this application before connecting the GPS Log Book device to your computer/laptop for the first time

1. Navigate to the *Downloads* page of the GPS Log Book website www.gpslogbook.co.za/downloads

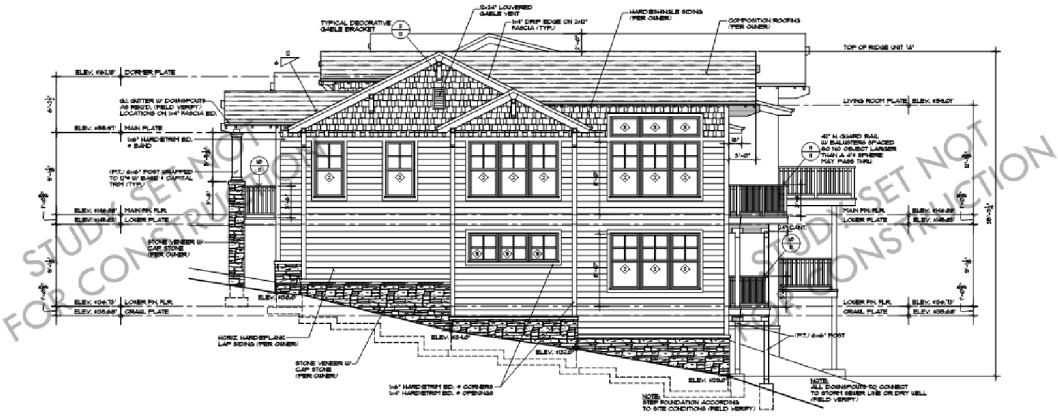
2. Ensure that the system requirements of the computer or laptop in use, meet those listed on the page.
3. Click on the relevant download icon.

GPS Log Book – User Manual www.gpslogbook.co.za Page 5 of 21

Obr. 4.1: *Časť príkladnej používateľskej príručky*[1].

[1]Používateľská príručka prevzatá z http://www.gpslogbook.com/us/Docs/UserManual.pdf

Obr. 4.2: *Návrh produktového katalógu od Freepik*[2].



Obr. 4.3: *Časť prikladného plánu (blueprint)*[3].

4.2 Adresát dokumentácie

Na konečnú podobu dokumentácie má obrovský vplyv cielový adresát dokumentácie. Je logické, že jej vzhľad sa bude líšiť podľa skúsenosti používateľa, pre ktorého je dokumentácia určená.

[2]Katalóg prevzatý z https://www.freepik.com/free-vector/gradient-creative-portfolio-templates_21077116.htm

[3]Plán prevzatý z https://www.houseplans.pro/plans/plan/study

Dokumentácia pre začiatočníka je písaná veľmi jednoducho s jasnými inštrukciami na štýl návodu. Od čitateľa sa neočakáva žiadna skúsenosť a nie su naňho kladené žiadne očakávania. Dokumentácia by nemala byť príliš dlhá, lebo by to používateľa odradilo od čítania. Naopak by mala byť čo najkratšia, aby prípadne ďalšie dokumentácie neboli až také zastrašujúce. Je dôležité v tejto dokumentácii čitateľa povzbudiť k jej čítaniu. Tiež je potrebné uvádzať príklady, ale je veľmi podstatné, aby boli krátke. Celkovo pri písaní dokumentácie pre začiatočníka platí, že menej je viac [11].

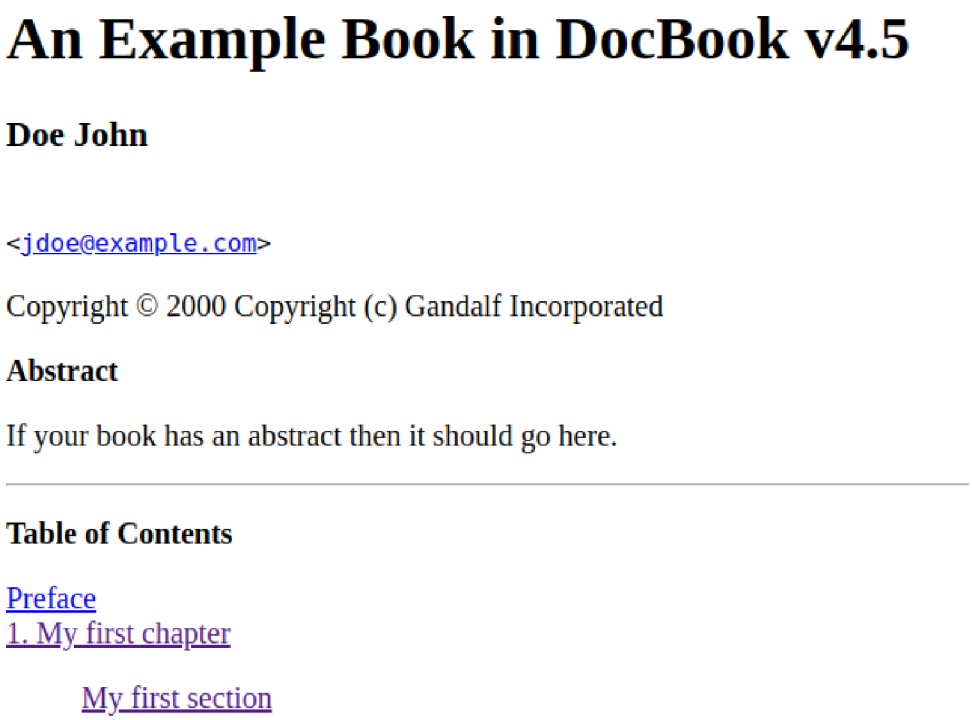
Skúsený používateľ nepotrebuje návod pre zvyčajné úlohy, ale skorej potrebuje podrobnosti o daných úlohách. Skúsený používateľ je pravdepodobne zoznámený s podobným produktom. Preto potrebuje dokument, ktorý ho v priebehu pol dňa efektívne zoznámi s aspoň najvýznamnejšími funkciami. Aj keď sa očakáva skúsenosť s podobným produktom, stále je potrebné popísať daný produkt jasne a v logickom poradí. Taktiež je treba mať krátky úvod, z ktorého sa prejde na detailnejšie spracovanie úloh a prípadne informácie o vlastnom prispôsobení. Do veľkej miery sa spolieha na znalosti, ktoré už čitateľ má [11].

4.3 Aktuálne generátory

V tejto dobe existujú generátory technickej dokumentácie, ktoré sú založené na automatickom generovaní technickej dokumentácie priamo zo súboru XML, prípadne zo zdrojového kódu. V tejto časti predstavím najznámejšie z nich.

DocBook je veľmi populárny súbor tagov na popisovanie kníh, článkov a ďalších prózových dokumentov, najmä technickej dokumentácie. DocBook je definovaný pomocou natívnej syntaxe DTD formátu SGML[4] a XML. DocBook obsahuje prvky, ktoré umožňujú hierarchické štrukturovanie dokumentov do kapitol, podkapitol a podobne. DocBook poskytuje širokú škálu prvkov na označenie rôznych typov obsahu, ako sú obrázky, výstupy programov, súbory, parametre príkazov, klávesové skratky, položky ponúk a iné. Tento systém umožňuje vytvárať dobre štruktúrované dokumenty, ktoré sa efektívne konvertujú do rôznych formátov, vrátane HTML a LaTeX. Zjednodušene povedané, DocBook je DTD pre XML (prípadne SGML), a na základe predpísanej sady tagov vie generovať technickú dokumentáciu. DocBook využíva štýly, napríklad XSL, na spracovanie týchto sád tagov a generovanie výslednej dokumentácie. Je možné využiť rôzne editory na prácu s DocBook, ako sú naprílad Oxygen, Emacs a jEdit. [32].

[4]XML je zjednodušená podmnožina SGML (*Standard Generalized Markup Language*)



An Example Book in DocBook v4.5

Doe John

<jdoe@example.com>

Copyright © 2000 Copyright (c) Gandalf Incorporated

Abstract

If your book has an abstract then it should go here.

Table of Contents

Preface
1. My first chapter

My first section

Obr. 4.4: *Časť vygenerovanej dokumentácie z DocBook XML súboru*[5] *vo formáte HTML.*

DITA (*OASIS Open Darwin Information Typing Architecture*) predstavuje štandardnú architektúru využívajúcu XML na zobrazenie dokumentov primárne určených pre čítanie ľuďmi. Poskytuje architektonické prvky pre modularitu obsahu, opätovné využívanie obsahu a kontrolované rozšírenie slovníkov dokumentov tak, aby sa zabezpečila interoperabilita dokumentov DITA. Architektúra DITA bola pôvodne vyvinutá v spoločnosti IBM pre technickú dokumentáciu IBM, najmä pre obsah na internete, a v roku 2004 bola darovaná organizácii OASIS Open. Architektúra DITA je riadená dvoma súvisiacimi, ale odlišnými požiadavkami [16]:

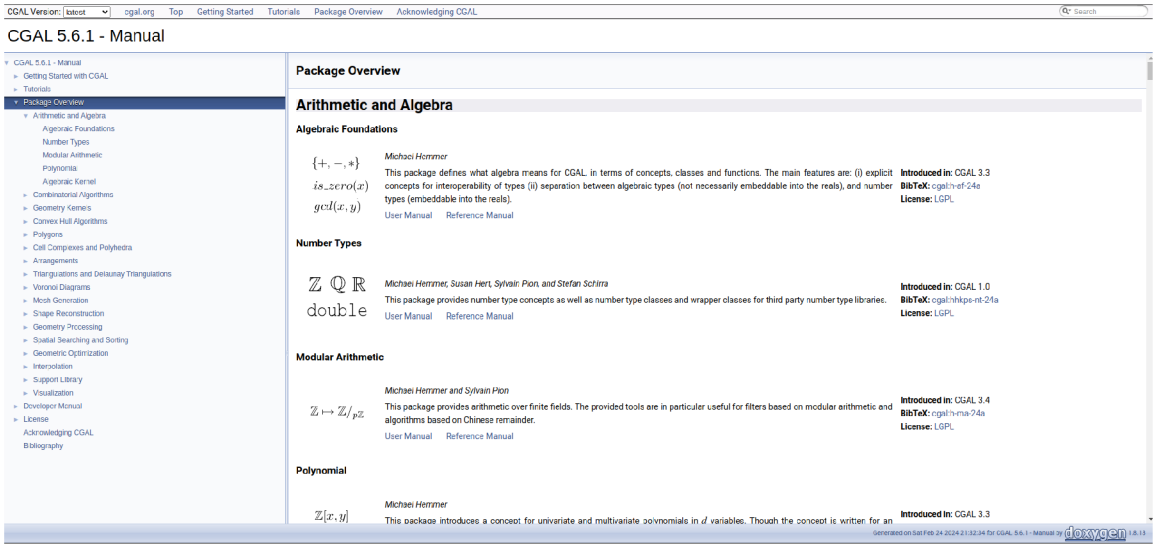
- Umožniť výmenu a interoperabilitu obsahu XML z rôznych zdrojov bez potreby, aby sa všetci zúčastnení zhodli na jedinom nadradenom definovaní typu dokumentu (DTD).
- Umožniť opätovné využívanie obsahu medzi rôznymi publikáciami a tiež v rámci jednej publikácie.

DITA a DocBook riešia podobné sady všeobecných požiadaviek na zobrazenie dokumentov, ale odzrkadľujú odlišné architektonické prvky a prístupy. Konkrétne DITA umožňuje tvorbu dokumentov z opakovateľných blokov, ktoré môžu byť umiestnené a využité v rôznych častiach dokumentu alebo dokonca v rôznych dokumentoch, zatiaľčo DocBook používa štandardnú hierarchiu a dokument je písaný lineárne ako tradičný text [16].

Doxygen je široko využívaný generátor dokumentácie vo vývoji softvéru. Automatizuje proces generovania dokumentácie z komentárov v zdrojovom kóde, analyzujúc informácie o triedach, funkciách a premenných, aby vytvoril výstup vo formátoch ako HTML a PDF. Doxygen poskytuje robustnú podporu pre dokumentovanie kódu v jazyku C++, rozpoznáva

[5] Odkaz na DocBook XML súbor: https://www.w3.org/International/its/tests/DocBookSample1.xml

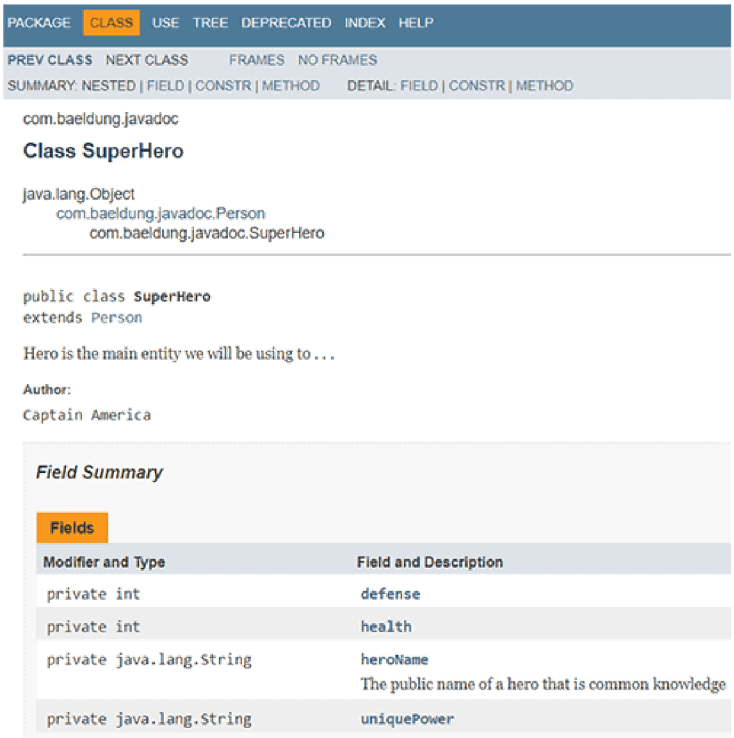
zložitosť tohto jazyka a generuje komplexnú dokumentáciu. Okrem C++ Doxygen podporuje aj jazyky C, Python, PHP, Java, C#, Objective-C, Fortran, VHDL, Splice, IDL a Lex. Jednou z kľúčových vlastností Doxygenu je jeho schopnosť extrahovať dokumentáciu priamo zo zdrojového kódu. To znamená, že vývojári môžu písať dokumentáciu priamo vo svojom kóde, keď ho píšu, namiesto toho, aby udržiavali samostatné súbory dokumentácie. Týmto spôsobom sa môže zabezpečiť, že dokumentácia je vždy aktuálna a presná, a tiež môže uľahčiť vývojárom písanie a údržbu dokumentácie. Doxygen sa široko používa v rôznych oblastiach, vrátane vývoja softvéru, technickej dokumentácie a výskumu. Je obzvlášť užitočný pre veľké projekty s komplexnými kódovými základmi, pretože pomáha organizovať a dokumentovať kód jednoduchým a zrozumiteľným spôsobom. Je tiež veľmi prispôsobiteľný, umožňujúc používateľom ovládať vzhľad a rozloženie generovanej dokumentácie [6] [23].



Obr. 4.5: *Dokumentácia vygenerovaná nástrojom Doxygen*[6].

Javadoc je nástroj v jazyku Java, ktorý slúži na automatickú generáciu dokumentácie z komentárov priamo v zdrojovom kóde. Používa sa na dokumentovanie Java API (*Application Programming Interface*), a tým umožňuje vývojárom generovať podrobnú a štruktúrovanú dokumentáciu priamo zo zdrojového kódu. V Javadoc komentároch sa obvykle umiestňujú špeciálne značky, ktoré obsahujú informácie o triedach, metódach, premenných a iných častiach kódu. Tieto komentáre sa následne spracovávajú nástrojom Javadoc, ktorý vygeneruje HTML formát dokumentácie. Výsledná dokumentácia je užitočná pre vývojárov, ktorí môžu ľahko prehľadávať, rozumieť a používať rôzne časti API [15] [13].

[6]Odkaz na celú dokumentáciu: https://doc.cgal.org/latest/Manual/index.html.



Obr. 4.6: *Dokumentácia vygenerovaná nástrojom Javadoc* [13].

4.4 Formáty technickej dokumentácie

Vyššie spomenuté populárne nástroje na generovanie dokumentácie DocBook, DITA, Doxygen a Javadoc podporujú výstupný formát HTML. Základným princípom výslednej HTML dokumentácie v týchto nástrojoch je, že dokumentácia je interaktívna a umožňuje používateľom preklikávať medzi rôznymi časťami dokumentácie pre jednoduchšie prehľadávanie a pohodlné získavanie informácií. V praxi generátor vygeneruje HTML súbory, ktoré potom môže tvorca zverejniť formou webovej stránky a sprístupniť dokumentáciu online. Ďalší populárny formát, ktorý produkujú generátory DocBook, DITA a Doxygen, je PDF, respektíve Doxygen generuje LaTeX kód, ktorý sa dá preložiť na PDF. Mať výslednú dokumentáciu v PDF je veľmi výhodné – ak by používateľ chcel dokumentáciu vytlačiť a mať ju ako fyzický dokument. Tiež je výhodné zálohovať dokumentáciu v PDF formáte a zároveň má štandardnejší vzhľad ako napríklad HTML, čo môže byť vhodné pre oficiálne publikácie. Taktiež pre offline dokumentáciu môže byť PDF vhodnejší formát ako HTML.

Existujú ešte ďalšie formáty podporované rôznymi generátormi technickej dokumentácie, a to sú napríklad Markdown a RTF (*Rich Text Format*). Markdown je jednoduchý značkovací jazyk, ktorý sa používa na formátovanie obyčajného textu pridávaním rôznych prvkov (nadpisy, odrážky, URL adresy) bez potreby formálneho textového editora alebo použitia HTML tagov. RTF formát je vyvinutý spoločnosťou Microsoft a predstavuje metódu kódovania formátovaného textu a grafiky pre použitie v aplikáciách. Tento formát uľahčuje výmenu dokumentov medzi rôznymi platformami s inými produktami Microsoft, čím slúži účelu interoperabilit. Niektoré generátory, ako napríklad Doxygen, umožňujú výstup priamo v XML, čo dáva zmysel, kedže Doxygen generuje zo zdrojového kódu, ale v prípade mojej aplikácie by to nedávalo zmysel, kedže samotný XML je vstup [24] [9].

Kapitola 5

Návrh

Pri navrhovaní je kľúčové predovšetkým zvážiť, aké potreby a požiadavky by mohol mať koncový používateľ. Je dôležité venovať dostatočnú pozornosť identifikácii funkcií, ktoré by mohli prispieť k efektívnemu využívaniu generátora. Dokumentácia, ktorú vygeneruje tento nástroj, by mala byť prehľadná, jasne štruktúrovaná a jednoduchá na orientáciu, aby čitateľ mohol efektívne lokalizovať potrebné informácie. Vlastnosti kvalitnej technickej dokumentácie sa detailnejšie preberajú v kapitole 4.

Ako prvú vec by som navrhol výber farebnej schémy. Výber vhodnej farebnej schémy pre výslednú dokumentáciu predstavuje dôležitý prvok, ktorý by som ponúkol používateľovi. Farebná schéma môže byť vybraná podľa osobných preferencií používateľa alebo môže byť prispôsobená tak, aby ladila s firemnou identitou alebo charakterom produktu.

Vzhľadom na to, že generátor by mal byť schopný spracovať teoreticky akýkoľvek XML vstup, je zásadné, aby používateľ bol schopný konkrétny XML dokument doplniť o ďalšie dodatočné informácie. Používateľ by mal mať možnosť pridávať komentáre k jednotlivým častiam XML. Tieto komentáre by sa následne mali integrovať do výslednej dokumentácie, čím pridajú ďalší významný obsah a kontext, ktorý zlepší pochopenie generovaného výstupu. Umožnenie pridávania komentárov zaručuje, že používateľ môže ľahšie vysvetliť alebo ozrejmiť rôzne časti XML dokumentu, čo vo výsledku vedie k celkovo kvalitnejšej a užitočnejšej dokumentácii.

Okrem možnosti pridávať komentáre bude tiež existovať možnosť vložiť obrázky, ktoré sa integrujú do výslednej dokumentácie. Tieto obrázky môžu slúžiť na vizuálne ilustrovanie rôznych aspektov XML štruktúry alebo procesov, ktoré sú opísané v dokumentácii. Ich prítomnosť pridáva ďalšiu hĺbku a zrozumiteľnosť pre používateľa, umožňujúc mu lepšie pochopiť a vizualizovať informácie. Integrácia obrázkov do dokumentácie tak zabezpečuje bohatší a pútavejší zážitok pri jej prezeraní. Detailnejšie o integráci obrázkov a komentárov do výslednej dokumentácie sa dá dočítať v sekcii 5.2.

XML dokumenty často obsahujú viacero identických elementov na rovnakej úrovni s rovnakou štruktúrou a obsahom. Tieto opakujúce sa elementy je možné efektívne zoskupiť do tabuľky, čo výrazne zlepšuje prehľadnosť a umožňuje lepšiu organizáciu dát pre čitateľa dokumentácie. Je nevyhnutné, aby bol generátor navrhnutý tak, aby používateľ mal možnosť aktivovať alebo deaktivovať túto funkciu generovania tabuliek. Každý tvorca dokumentácie môže mať iné preferencie zobrazenia a niektoré XML dokumenty môžu byť viacej vhodné, zatiaľ čo iné nie. Konkrétnejšie je táto funkcionalita popísaná v sekcii 5.3.

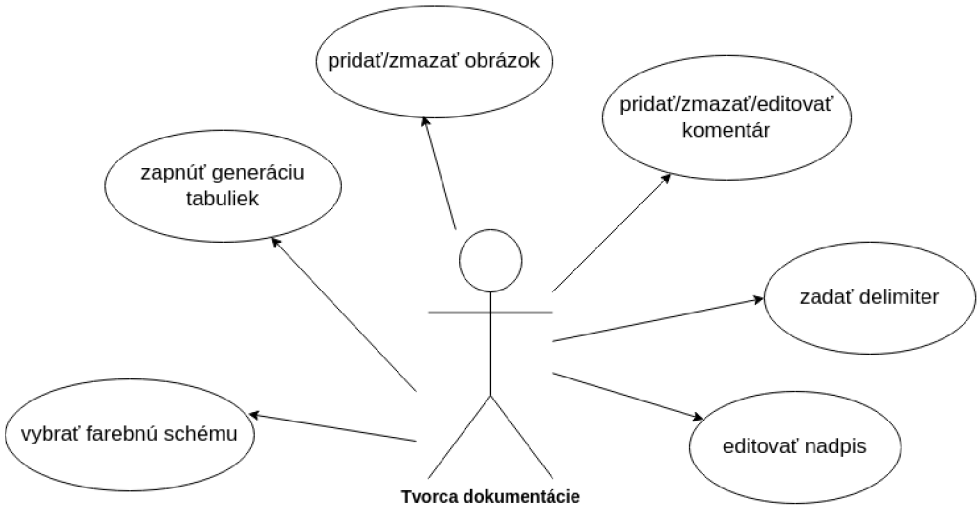
Štandardne by nadpis dokumentácie mal byť odvodený zo samotného koreňového elementu XML s možnosťou používateľa ho zmeniť podľa vlastných preferencií. Ďalším dôležitým architektonickým prvkom v generátori bude schopnosť zadať delimiter (oddelovač),

ktorým bude možné rozdeliť atribúty. Tento delimiter umožní generátoru rozpoznať, ako majú byť atribúty správne rozdelené. Niektoré XML dokumenty môžu mať atribúty, ktoré obsahujú viacero hodnôt oddelených rôznymi delimitrami, ako sú čiarka, bodka, dvojbodka, bodkočiarka a podobne. Pre lepšiu čitateľnosť výslednej dokumentácie je dôležité, aby takéto atribúty boli správne rozdelené a vykreslené. Konkrétnejšie je táto funkcionalita popísaná v sekcii 5.4.

Je dôležité, aby výsledná dokumentácia poskytovala možnosť vyhľadávania. Táto funkcia je obzvlášť užitočná v prípade, keď má dokumentácia veľký obsah, pretože umožňuje používateľovi efektívne nájsť relevantné informácie bez zbytočného prehľadávania celej dokumentácie.

Pre výstupný formát technickej dokumentácie som sa rozhodol zvoliť HTML, čo je štandard medzi modernými generátormi dokumentácie. Tento formát je široko podporovaný a často používaný pre jeho schopnosť generovať vizuálne príťažlivé a prístupné dokumenty. Navyše, HTML umožňuje bohatú interaktivitu a jeho vzhľad sa dá jednoducho a efektívne upravovať pomocou CSS.

Môj hlavný cieľ je vytvoriť nástroj, ktorý bude schopný spracovať ľubovoľný XML dokument nezávisle od toho, či spĺňa nejakú špecifickú definíciu, čo pri DocBook a DITA generátore musí, aby z neho vygenerovali technickú dokumentáciu. Zároveň je mojim cieľom umožniť vytváranie atraktívneho výstupu bez toho, aby používateľ potreboval písať zložité štýly či transformačné pravidlá, ako napríklad je XSLT pre konkrétny súbor. Diagram prípadu použitia tohto generátora je možné vidieť na obrázku 5.1.



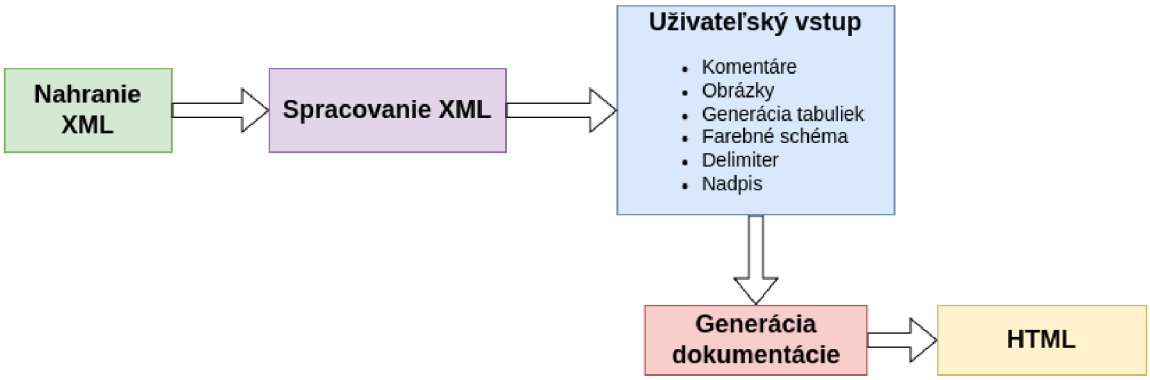
Obr. 5.1: *Diagram prípadu použitia generátora.*

Aby som zaistil, že aplikácia bude funkčná na rôznych platformách, zamýšľam ju vyvinúť ako webovú aplikáciu. Tento prístup umožní používateľom pristupovať k aplikácii prostredníctvom webového prehliadača na rôznych zariadeniach, čím poskytne flexibilitu a prístupnosť bez ohľadu na operačný systém alebo typ zariadenia. Princíp webovej aplikácie je uvedený v následujúcich krokoch aj s ilustračnou schémou 5.2.

1. Používateľ nahrá do webovej aplikácie XML súbor.
2. Generátor spracuje vstupný XML dokument.
3. Používateľ pridá svoj vstup cez webové rozhranie.

4. Používateľ spustí generáciu dokumentácie.

5. Generátor vráti dokumentáciu, ktorá sa používateľovi stiahne.



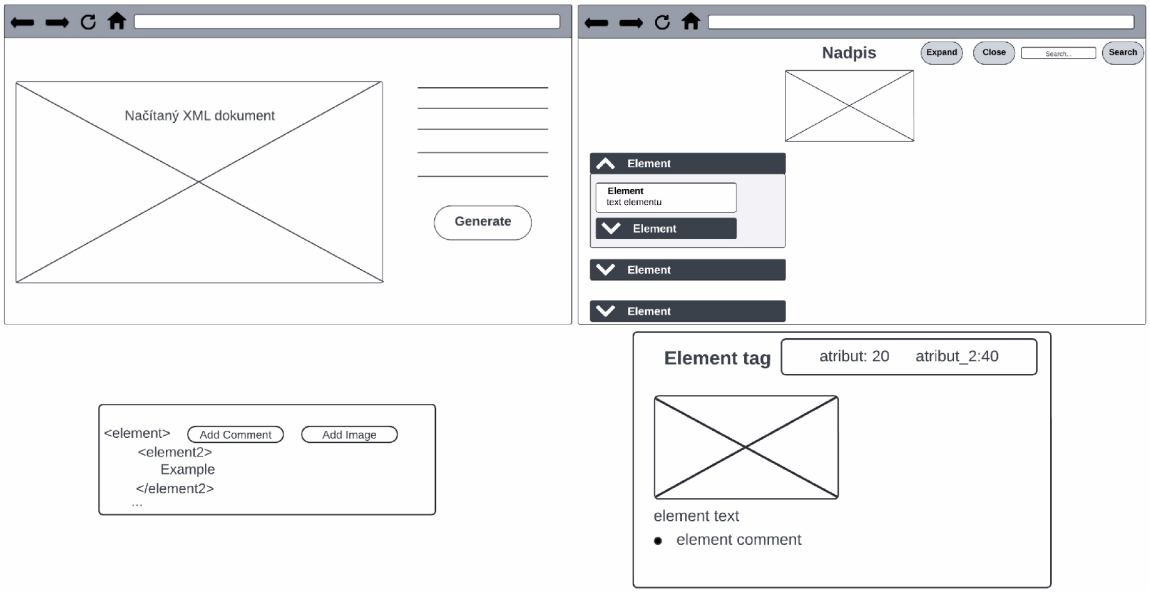
Obr. 5.2: *Schéma princípu generátora.*

5.1 Vzhľad

Vzhľadom na to, že plánujem vyvinúť webovú aplikáciu, je nevyhnutné dôkladne zvážiť jej dizajn a rozloženie prvkov. Na domovskej stránke bude určite možnosť nahrať XML súbor, ktorý bude neskôr spracovaný. Po spracovaní XML súboru bude presmerovanie na ďalšiu stránku, kde bude pekne[1] zobrazený konkrétny XML súbor. Zobrazenie XML súboru bude na ľavej časti stránky, kde bude možnosť pridávania komentárov a obrázkov. Zobrazenie obsahu XML súboru bude umiestnené na ľavej časti webovej stránky, kde budú mať používatelia možnosť nielen prehliadať dáta, ale aj k nim pridávať komentáre a vkladať obrázky. Táto interaktivita zlepšuje používateľskú angažovanosť a umožňuje plynulejšiu a efektívnejšiu prácu s dokumentom. Na pravej časti stránky plánujem mať formulár, ktorý zastrešuje zvyšné možnosti, ktoré môže tvorca dokumentácie mať a ktoré sú zobrazené v diagrame prípadu použitia 5.1.

Vygenerovaná dokumentácia sa bude skladať z viacerých základných častí, ktoré na začiatku zahŕňajú nadpis a atribúty, komentár a obrázok koreňového elementu XML dokumentu. Táto kombinácia prvkov bude fungovať ako titulok dokumentácie, čím poskytne okamžitý vizuálny a textový kontext hlavného obsahu. Následne bude zvyšok dokumentácie obsahovať ďalšie elementy, ktoré budú organizované do rozbalovacích sekcií, čo umožní používateľom zobrazovať alebo skrývať obsah podľa potreby. Tento prístup minimalizuje preťaženie informáciami a umožňuje čitateľom zamerať sa na špecifické časti dokumentu bez nutnosti prechádzať celou dokumentáciou. Dokumentácia bude obsahovať menu, ktoré poskytne používateľom nástroje pre efektívnu navigáciu. V menu bude zahrnutá vyhľadávacia funkcia, umožňujúca používateľom rýchlo lokalizovať konkrétne informácie v rámci dokumentu. Okrem toho, menu bude obsahovať tlačidlá na rozbalenie všetkých sekcií pre rýchly prístup k celkovému obsahu, a na ich spätné zabalenie, čo pomôže udržiavať prehľadnosť a organizáciu dokumentácie.

[1] Takzvaný *pretty print*.



Obr. 5.3: *Wireframy generátora a vygenerovanej dokumentácie. V plnej veľkosti sú dostupné v prílohe C.*

5.2 Komentáre a obrázky

Ako som už v návrhu spomenul, pridanie komentárov a obrázkov do XML bude hlavným bodom, ktorý ponúkne tvorcovi dokumentácie možnosť pridať dodatočné informácie k dátam, ktoré uchováva XML dokument. Konkrétne bude nástroj ponúkať možnosť pridať komentár a obrázok jednotlivo ku každému elementu. Plán je sprístupniť kliknutie na otvárací tag elementu, kde vyskočí možnosť pridať komentár alebo obrázok. Tým, že sa bude dať takto anotovať každý jeden element jednotlivo, bude možnosť rozšíriť dáta, ktoré XML dokument nesie, o potrebné informácie. V načítanom XML dokumente sa označí element, ktorý bol okomentovaný alebo k nemu bol pridaný obrázok, aby bolo jednoznačne poznateľné, ktorý element už bol rozšírený o dodatočný kontext. Po kliknutí na tlačidlo „pridať komentár“ sa zobrazí textové pole, kde sa bude môcť komentár vpísať a kliknutie na „pridať obrázok“ vyzve používateľa nahrať obrázok zo svojho zariadenia. Komentár a obrázok budú integrovaný do elementového bloku.

5.3 Generácia tabuliek

Myšlienka generovania tabuliek spočíva v zjednodušení a zlepšení prehľadnosti dokumentácie tým, že sa predíde opakovaniu rovnakých alebo podobných elementov v dokumente. Namiesto toho sú tieto elementy nahradené jednou tabuľkou, ktorá efektívne a prehľadne zobrazuje všetky relevantné údaje týchto elementov. Na to, aby sa z elementov mohla vytvoriť tabuľka, musia splniť niekoľko podmienok:

- Elementy majú rovnaký názov.
- Elementy majú rovnakého rodiča.
- Každý element obsahuje aspoň 1 synovský element.

- Synovia elementov neobsahujú žiadnych ďalších synov.

Keď skupina elementov splní tieto podmienky, tak z nich vznikne tabuľka. Na príklade 5.4 vidíme opakujúce sa elementy *item* s dvomi synovskými elementami. Ak si používateľ nechá zapnutú generáciu tabuliek, tak je žiadaný výsledok zobrazený v tabuľke 5.1.

```
<item>
  <name>Produkt A</name>
  <price>19.99</price>
</item>
<item>
  <name>Produkt B</name>
  <price>29.99</price>
</item>
<item>
  <name>Produkt C</name>
  <price>39.99</price>
</item>
```

Obr. 5.4: *XML dokument zložený z item elementov.*

Name	**Price**
Produkt A	19.99
Produkt B	29.99
Produkt C	39.99

Tabuľka 5.1: *Tabuľka item elementov.*

5.4 Delimiter v hodnote atribútu

Niektoré nástroje používajú na ukladanie viacerých informácií v jednom atribúte techniku oddelenia týchto údajov pomocou delimitra. Ak by si chcel niekto tento XML dokument zdokumentovať pomocou môjho generátora, tak bude mať sprístupnené pole, kde zadá hodnotu delimitra, podľa ktorého generátor bude vedieť rozdeliť jeden atribút na viacej atribútov, kde každý atribút bude mať už iba jednu informáciu vo výslednej dokumentácii. Napríklad nástroj na kreslenie diagramov Draw.io, ktorý je spomenutý v sekcii 3.5, používa v niektorých prípadoch ukladanie hodnôt v atribúte oddelené bodkočiarkou.

```
style="ellipse;whiteSpace=wrap;html=1;aspect=fixed;"
```

Obr. 5.5: *Atribút style vytiahnutý z Draw.io XML dokumentu.*

Normálne by sa atribút *style* vykreslil ako jeden atribút s hodnotou, ale po zadaní bodkočiarky do pola generátora by mal výstup vyzerať následovne.

```
style(1.)="ellipse" style(2.)="whiteSpace" style(3.)="wrap"
style(4.)="html=1" style(5.)="aspect=fixed"
```

Obr. 5.6: *Rozdelený atribút z príkladu 5.5 podľa bodkočiarky.*

Kapitola 6

Implementácia

Pri implementácii generátora je kľúčové ako prvé zvážiť výber programovacieho jazyka, v ktorom bude daný generátor implementovaný. Hlavným kritériom pri rozhodovaní o jazyku je schopnosť efektívne spracovať vstupný XML súbor. V sekcií 3.4 som detailne prezentoval dostupné nástroje, medzi ktorými som sa rozhodol voľbu uprieť na programovací jazyk Python a knižnicu lxml. Táto voľba je odôvodnená jej širokým záberom, ktorý sa môže počas implementácie ukázať ako nesmierne užitočný. Úlohou knižnice lxml bude spracovávať nahratý XML súbor od používateľa a následne generovať výslednú dokumentáciu vo formáte HTML. Knižnica lxml je už spomenutá v sekcií 3.4. Ako už bolo spomenuté v návrhu, generátor bude implementovaný ako webová aplikácia. Pre tento účel som sa rozhodol použiť webový framework Flask, ktorý je obzvlášť vhodný pre rýchly vývoj jednoduchých webových aplikácií. V ďalšej sekcií si priblížime framework Flask, ktorý mienim využiť.

6.1 Flask

Flask je malý framework, ktorý sa zvyčajne označuje ako *microframework*. Je dostatočne malý na to, aby bolo možné sa s ním veľmi rýchlo zoznámiť. To, že je malý, však neznamená, že toho robí menej ako iné frameworky. Flask bol od základu navrhnutý ako rozšíriteľný framework. Poskytuje pevné jadro so základnými službami, zatiaľ čo rozšírenia poskytujú zvyšok. Vďaka tomu je môžné vybrať a používať rozšírenia, ktoré sú skutočne potrebné. Flask má dve hlavné závislosti. Smerovanie, ladenie a rozhranie brány webového servera (WSGI) pochádzajú z Werkzeug, zatiaľ čo podporu šablón poskytuje Jinja2. Werkzeug aj Jinja2 vytvoril hlavný vývojár Flasku. V aplikácii Flask nie je natívna podpora pre prístup k databáze, overovanie webových formulárov, autentifikáciu používateľov ani iné úlohy na vysokej úrovni. Tieto a mnohé ďalšie kľúčové služby, ktoré väčšina webových aplikácií potrebuje, sú k dispozícii prostredníctvom rozšírení, ktoré sa integrujú so základnými balíkmi. Vývojár môže používať rozšírenia, ktoré najlepšie vyhovujú konkrétnemu projektu alebo si dokonca môže napísať vlastné. To je rozdiel od väčších frameworkov, kde sú väčšinou všetky voľby urobené a je zložité alebo nemožné ich zmeniť[1] [10].

[1] Odkaz na online dokumentáciu Flasku: https://flask.palletsprojects.com/en/3.0.x/

6.2 Načítanie XML dokumentu

Flask využíva takzvané templaty HTML súborov, ktoré sa majú vykresliť pri navštívení konkrétnej URL. Aplikácia sa skladá z dvoch templatov, a to sú `index.html` a `generator.html`. Template `index.html` sa vráti pri zadaní koreňovej URL, na ktorej je možné nahrať XML súbor. Nahratý XML súbor spracuje knižnica lxml, ktorá vráti koreňový element obsahujúci odkaz na svojich synov, ktorí sa zasa odkazujú na svojich synov; toto odkazovanie pokračuje až po listové uzly. Na to, aby sa mi neskôr pohodlnejšie pracovalo s XML dokumentom, si potrebujem načítať jednotlivé elementy do nejakej štruktúry. Preto som sa rozhodol si vytvoriť list elementov, s ktorým budem pracovať. Vytvorenie listu elementov mi zabezpečí funkcia **parse_xml** 6.1.

```
def parse_xml(element, depth=0, id=0):
  def recursive_process(element, depth, id=0,
                        parent_id="root", parent_tag="root"):

      parent_position = str(parent_id).rsplit('-', 1)[-1]
      if parent_id == 0:
          before_path = ''
      else:
          before_path = f"{parent_tag}({parent_position}.) > "

      element_data = {
          'tag': element.tag,
          'attributes': element.attrib.items,
          'text_value': element.text.strip() if element.text else '',
          'depth': depth,
          'id': f"{parent_id}-{id}" if id != 0 else str(id),
          'path': f"{before_path}{element.tag}({id}.)" if id != 0 else '',
          'comment' : '',
          'image' : None,
          'image_name' : '',
          'flag' : []
      }

      element_list.append(element_data)

      for child_id, child in enumerate(element, 1):
          recursive_process(child, depth + 1, child_id,
          f"{parent_id}-{id}" if id != 0 else id,
          f"{before_path}{element.tag}" if id != 0 else '')

      element_list.append({'tag': element.tag, 'depth': depth, 'id': "x"})

  element_list = []
  recursive_process(element, depth, id)
  return element_list
```

Obr. 6.1: *Čiastočne skrátená a upravená funkcia* ***parse_xml****, ktorá vracia list elementov.*

Funkcia **parse_xml** je trochu skrátená oproti skutočnej implementácii (je skrátená iba o vyriešenie problému, ktorý zvykol nastať, ak mal XML dokument odkaz na *namespace*) kvôli lepšej demonštrácii. Funkcia obsahuje ešte svoju vlastnú vnútornú funkciu, ktorá sa

rekurzívne volá pre synov pre kompletné prehľadanie XML stromu. Výsledok tejto funkcie je list elementov, kde jednotlivý element je vytvorený pomocou slovníku. Každý element obsahuje tag, atribúty tiež uložené v slovníku, textovú hodnotu, hĺbku zanorenia, unikátne id, úplnú cestu s tagmi rátanú od elementu prvej úrovne zanorenia, komentár, obrázok, meno obrázka a list príznakov. Keď sa spracujú všetci synovia elementu, tak znova vkladám jeho tag do listu s id hodnotou „x“ kvôli správnemu zobrazeniu načítaneho XML dokumentu na frontende.

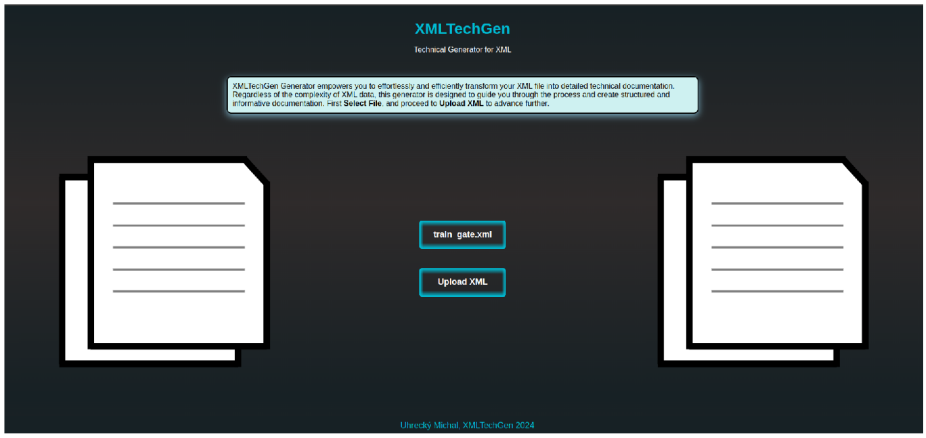
Názov, atribúty a textovú hodnotu ukladám priamo z elementu. Element je rozšírený o hĺbku, ktorú využívam pre správne odsadenie elementov na frontende a má vplyv aj na generovanie dokumentácie. Unikátne identifikačné číslo (id) každého elementu je vytvorené z jeho cesty v dokumente, začínajúc od koreňového elementu až po daný element. Toto id je zostavené z čísel oddelených pomlčkami. Koreňový element má pridelené id s hodnotou 0. Čísla pre podradené elementy sa začínajú od 1 a s každým ďalším elementom na rovnakej úrovni sa hodnota zvyšuje o 1. Týmto spôsobom každý element v štruktúre dokumentu dostane svoje jedinečné id, ktoré jasne určuje jeho pozíciu. Možné to je vidieť na príklade 6.2, kde je id pre demonštračné účely napísané ako atribút elementu.

```
<root id="0">
  <child id="0-1">
      <subchild id="0-1-1"/>
      <subchild id="0-1-2"/>
  </child>
  <child id="0-2">
      <subchild id="0-2-1">
          <subsubchild id="0-2-1-1"/>
      </subchild>
  </child>
</root>
```

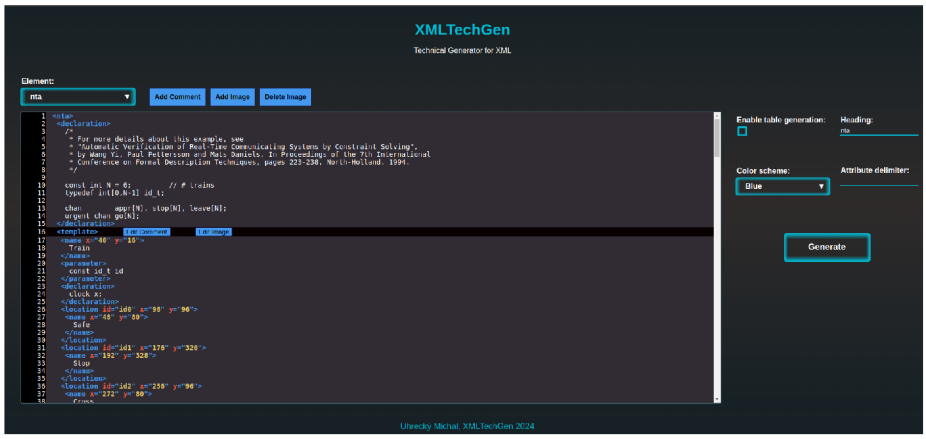
Obr. 6.2: *XML dokument, na ktorom majú elementy vpísané id ako jeho atribút.*

Úplná cesta (*path*) je veľmi podobná ako id, avšak neobsahuje koreňový element, ale za to obsahuje názov tagu. Cesta pre *subchild* s id „0-1-2“ z príkladu 6.2 by bola `child(1.) > subchild(2.)`. Cesta je použitá ako tooltip pre element vo výslednej dokumentácii, ktorá slúži na to, aby čitateľ dokumentácie v prípade veľkého zanorenia bol schopný rýchlo zistiť, kde sa nachádza. Komentár, obrázok a meno obrázka slúži ako „úložny priestor“, kde sa jednotlivé veci nahrajú po používateľskom vstupe. Príznak (*flag*) slúži na označenie elementu v prípade, že sa má pridať do tabuľkového zobrazenia.

Tento list elementov sa ukladá do relácie (*session*). Je použitý prístup takzvaných *server-side sessions*. To znamená, že list je uložený na strane servera a klient obsahuje iba relačné id, pomocou ktorého sa na list odkazuje. Ako východzie nastavenie používa Flask *client-side sessions*, kde sa zasa všetky údaje ukladajú na strane klienta. *Client-side sessions* neboli úplne vhodné pre tento účel najmä kvôli ich obmedzenej kapacite na 4 KB. Aby bolo možné nahrať aj väčšie XML dokumenty, bolo treba stiahnuť rozšírenie Flask-Session, ktoré sprístupnilo *server-side sessions*. Po uložení listu do relácie je používateľ presmerovaný na druhý template `generator.html`, kde, ako je spomínané v návrhu, používateľ môže manipulovať s XML dokumentom a pridávať mu komentáre a obrázky.



Obr. 6.3: *Domovská stránka generátora.*



Obr. 6.4: *Generátor s načítaným XML dokumentom.*

Keď používateľ klikne na element v načítanom XML dokumente, objavia sa tlačidlá, ktoré mu umožňujú pridať komentár alebo obrázok. Po kliknutí na jedno z týchto tlačidiel sa otvorí dialógové okno, kde môže používateľ vpísať text komentára alebo nahrať obrázok. Všetky tieto údaje sa následne odosielajú na server prostredníctvom webového API XMLHttpRequest, ktoré umožňuje vytvárať HTTP požiadavky z JavaScriptu bez toho, aby bolo potrebné znovu načítať stránku. Tieto požiadavky smerujú na koncové body (*endpoints*), ktoré sú definované v Python aplikácii s použitím frameworku Flask a aktualizujú list elementov uložený v relácii. V porovnaní s pôvodným návrhom bola pridaná funkcia hromadnej aktualizácie komentárov a obrázkov prostredníctvom tagu elementu. Toto rozšírenie umožňuje kategorickú anotáciu elementov, čo je výhodné, keďže tieto elementy často predstavujú rovnaký kontext. Komentáre sú ukladané do relácie ako reťazce, zatiaľ čo obrázky sú prevedené na reťazce pomocou Base64 kódovania. Podporované formáty obrázkov sú PNG A JPG.

6.3 Generácia dokumentácie

Pred samotným spustením generácie si používateľ môže zapnúť generáciu tabuliek, upraviť nadpis, vybrať farebnú schému z modrej, zelenej a čiernej farby a vyplniť delimiter

v atribúte. Generácia sa uskutočňuje prechádzaním listu elementov pomocou cyklu, ktorý postupne spracováva každý element. Hlavná podstata algoritmu je zobrazená nižšie v algoritme 1.

Algoritmus 1: Základný cyklus, ktorý tvorí elementy v HTML.

```
for element in element_list do
    if element.depth == 0 then
        create_root_element(element);
    else
        create_basic_element(element);
    end
end
```

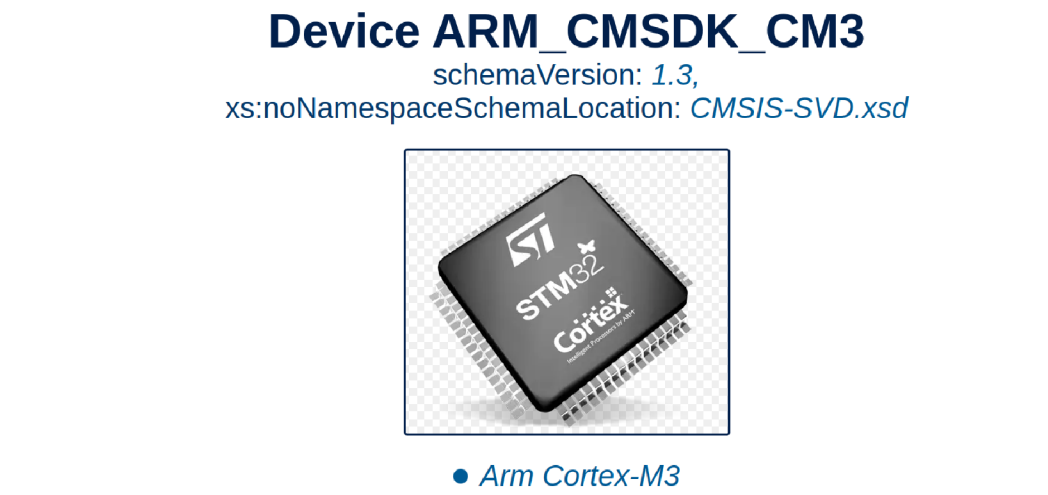
Predtým, než sa začne iterácia, vytvorím pomocou lxml knižnice html, head a body tag. Funkcia **create_root_element** vytvorí koreňový element ako priamy podelement tagu body. Algoritmus funkcie **create_basic_element** spočíva v nájdení rodiča elementu a následnom vytvorení podelementu tohto rodiča. V obidvoch funkciách sa najprv pre každý element vytvorí nadpis, ktorý vychádza z jeho tagu. Následne sa overí, či element obsahuje atribúty, obrázok, hodnotu alebo komentár. Ak obsahuje niektorú z týchto súčastí, príslušné časti elementu sú postupne vytvárané a pridávané k jeho celkovému zobrazeniu. Pri vytváraní HTML elementov sa postupne pridávajú aj príslušné štýly. Pre tento účel sa používa zoznam, do ktorého sa pridávajú názvy štýlových tried podľa potreby. Keď sa objaví element, ktorý má byť vygenerovaný, skontroluje sa, či názov jeho štýlovej triedy už existuje v zozname. Ak v zozname ešte nie je, pridá sa. Obsah koreňového elementu sa centruje na stred, aby pripomínal nadpis, prípadne titulnú stranu. Zvyšné nekoreňové elementy sú umiestnené na ľavej strane. Hĺbka každého elementu ovplyvňuje štýl, konkrétne veľkosť písma a farbu pozadia. Farebný odtieň pozadia sa strieda medzi bledším a tmavším tónom na základe hĺbky, čo zlepšuje vizuálne rozlíšenie medzi vnorenými elementami. K vygenerovanej dokumentácii sa prikladá CSS súbor.

Elementy, ktoré majú synov, sú zobrazené ako zabalovacie a rozbalovacie sekcie. Na zistenie, ktoré elementy budú slúžiť ako sekcie, je najskôr potrebné si zistiť z listu elementov, ktoré elementy majú synov. Funkciu, ktorá mi zistí počet synov, je možné vidieť v kóde nižšie 6.5.

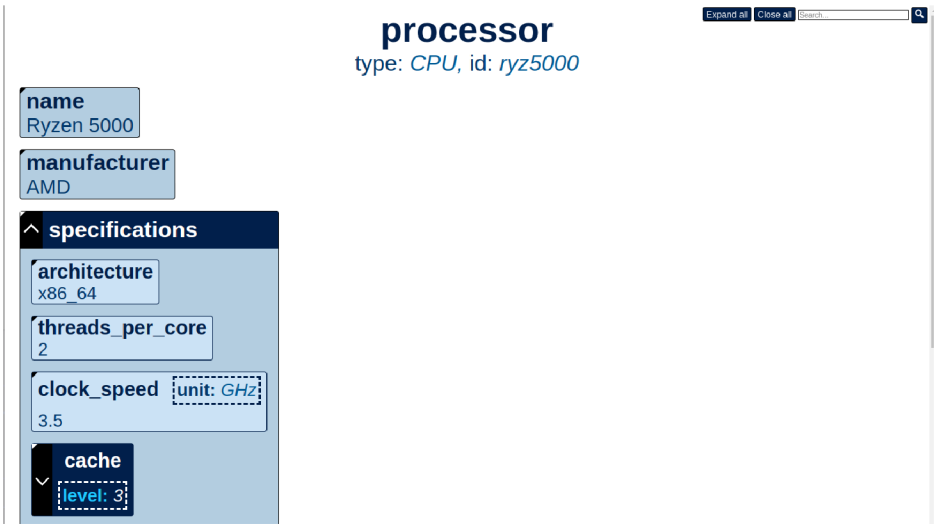
```python
def get_children_count(element_list):
    children_count = {}
    for element in element_list:
        if element['depth'] == 0:
            continue
        parent_id = element['id'].rsplit('-', 1)[0]
        if parent_id in children_count:
            children_count[parent_id] += 1
        else:
            children_count[parent_id] = 1
    return children_count
```

Obr. 6.5: *Funkcia **get_children_count**, ktorá vracia slovník elementov s počtom synov.*

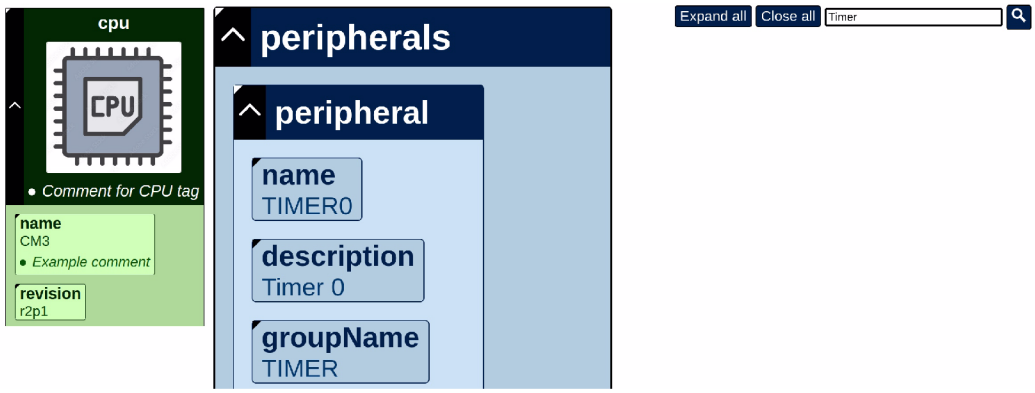
Pri tvorbe nekoreňového elementu, sa kontroluje, či element obsahuje synovské elementy. Ak obsahuje synovské elementy, element sa vytvorí ako rozkladacia a zbalovacia sekcia s použitím HTML tagov *details* a *summary*. Vygenerovaná dokumentácia obsahuje panel vpravo hore, kde je možnosť tieto rozbalovacie sekcie hromadne zabaliť a rozbaliť. V paneli sa ešte nachádza možnosť filtrovať obsah na základe zadaného podreťazca. Pri filtrovaní sa v dokumentácii zobrazia len tie elementy, ktoré obsahujú zadaný podreťazec, pričom toto vyhľadávanie je nezávislé na veľkosti písmen (*case-insensitive*). Funkcie sú umožnené vďaka dodanému JavaScriptovému súboru spolu s dokumentáciou. K spomínanému CSS súboru a JS súboru sa prikladajú ešte používateľom pridané obrázky v adresári *images*. Celá odpoveď zo servera bude odoslaná v ZIP formáte.



Obr. 6.6: *Koreňový element s atribútmi, komentárom a obrázkom vo výslednej dokumentácii ako titulok v modrej schéme.*



Obr. 6.7: *Náhlad na celkový výzor dokumentácie v modrej schéme. Hodnoty sú napísané pod tagom elementu a atribúty su vypísané v čiarkovanom ráme.*

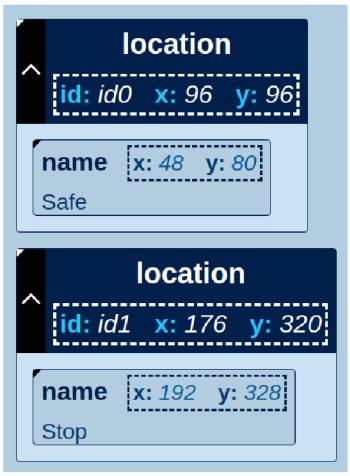


Obr. 6.8: *Na lavej strane obrázok v elemente v zelenej schéme a na pravej strane vyhľadávanie v dokumentácií pomocou podreťazca Timer.*

Ako ďalšiu vec bolo potrebné implementovať generáciu tabuliek. Generácia tabuliek je implementovaná následujúcim algoritmom:

1. Nájdu sa elementy, ktoré majú rovnakého rodiča a názov; musia byť aspoň dva.
2. Overí sa, že každý element obsahuje aspoň jedného syna a žiadny syn elementa už nemá ďalších synov.
3. Overeným elementom sa pridá označenie do ich položky flag.
4. Pri hlavnom cykle sa tieto elementy preskočia a uložia sa bokom aj s ich rodičmi.
5. Po skončení hlavnej generácie sa dogenerujú tabuľky pre tieto elementy pod uložených rodičov.

Do tabuľky sa zobrazujú aj atribúty elementov a ich hodnoty. Každý atribút má pred sebou ešte tag elementu v zložených zátvorkách, aby bolo jasné, z akého elementu je daný atribút. K tabuľke je možné pridať aj komentár a obrázok, tie sa berú z prvého elementu tabuľky. Problém môže nastať problém, ak majú synovské elementy rovnaký tag. Aktuálne je tento problém riešený zoskupením do jedného stĺpca, kde sú hodnoty oddelené zvislicou.



Obr. 6.9: *Elementy location bez generovania tabuliek.*

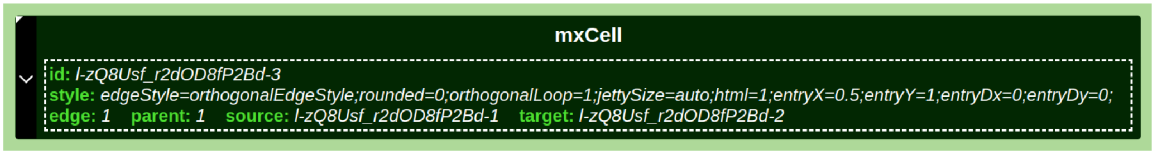
location table:

{location} id	{location} x	{location} y	name	{name} x	{name} y	label	{label} kind	{label} x	{label} y
id0	96	96	Safe	48	80				
id1	176	320	Stop	192	328				
id2	256	96	Cross	272	80	x<=5	invariant	272	96
id3	96	232	Appr	32	216	x<=20	invariant	32	232
id4	256	232	Start	272	216	x<= 15	invariant	272	232

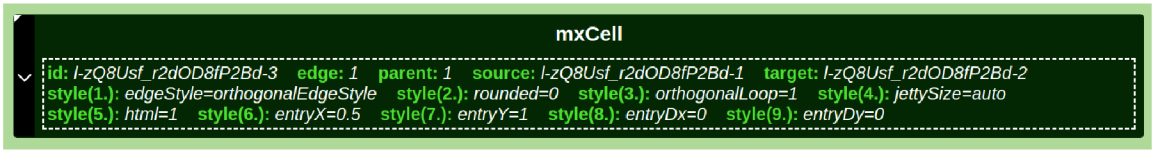
Obr. 6.10: *Elementy location s generovaním tabuliek.*

Na rozdelenie atribútu podľa delimitra sa používa funkcia, ktorá upravuje atribúty elementov v zozname elementov tak, že pre každý atribút, ktorý obsahuje špecifikovaný delimiter:

1. Rozdelí hodnotu atribútu podľa delimitra na viacero častí.
2. Vytvorí nové atribúty z každej časti, kde každý nový atribút dostane príponu s jeho poradovým číslom.
3. Pôvodné atribúty, ktoré boli rozdelené, sú odstránené.
4. Pridá nové transformované atribúty do elementu.

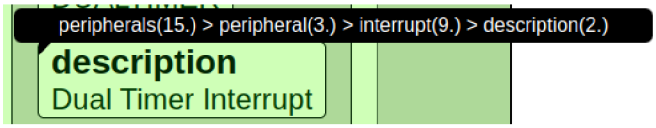


Obr. 6.11: *Atribút style bez použitia delimitra.*



Obr. 6.12: *Atribút style s použitím delimitra „bodkočiarka“.*

Ďalším z prvkov, ktorý môže zlepšiť čitateľnosť dokumentácie, je tooltip schovaný v ľavom hornom rohu elementu. Tento tooltip vychádza z položky *path* každého elementu a ukazuje cestu od prvého nekoreňového elementu. V tejto ceste je každý element identifikovaný poradovým číslom v zátvorkách, ktoré označuje jeho pozíciu pod príslušným rodičom. Tento nástroj pomáha čitateľom lepšie navigovať a rozumieť štruktúre dokumentácie tým, že poskytuje jasnú a intuitívnu vizuálnu stopu k hierarchii elementov.



Obr. 6.13: *Tooltip nad elementom description.*

Kapitola 7

Testovanie

Na testovanie aplikácie som použil 90 pripravených testovacích súborov. Sú to konfiguračné súbory nástroja UPPAAL a Draw.io a tiež SVD súbory. Detailnejšie sú súbory popísané v sekcií 3.5. Draw.io súbory boli prevzaté z príkladov ponúknutých v menu aplikácie. Súbory UPPAAL použité v tejto práci boli získané kombináciou príkladov priamo z nástroja UPPAAL a súborov, ktoré mi poskytol vedúci práce. SVD súbory boli prevzaté z GitHub repozitára[1] a stránky popisujúcej SVD formát[2]. Aplikáciu som spustil lokálne na bežnom notebooku vybavenom procesorom Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz a 16 GB operačnej pamäte RAM. Na súboroch aplikácie Draw.io sa dal dobre otestovať zadaný delimiter v generátore, kedže ich súbory využívajú ako delimiter bodkočiarku. Súbory majú malú kapacitu od 1 KB až po 110 KB a ich spracovanie prebiehalo bez problémov. UPPAAL súbory už majú väčšie rozpätie a ich kapacita sa nachádza medzi 1 KB až 2.5 MB. Na UPPAAL súboroch som sa najmä zameral na generovanie tabuliek. Najčastejšie boli tabuľky zložené z tagov *location* a *transition*. Tabuľky častokrát obsahovali stĺpce, ktoré obsahovali hodnoty viacerých elementov oddelených zvislicami a tiež aj veľa atribútov. Podľa mna sa v nich už čiastočne strácala prehľadnosť a niekde neboli úplne vhodné. Naopak pri SVD súboroch zapnutie generácie tabuliek bolo veľké sprehladnenie dokumentácie. SVD súbory charakteristicky obsahujú menší počet atribútov a elementy použité na generovanie tabuliek nezahŕňajú potomkov s rovnakým názvom. Vďaka tomu vznikali prehľadné a jasne štruktúrované tabuľky. SVD súbory patria medzi väčšie a tie testovacie majú kapacitu od 33 KB až po 9.4 MB. Pri UPPAAL súboroch, ktoré presiahli veľkosť 1 MB, sa začali objavovať výkonnostné problémy, a aplikácia už nebežala úplne plynule. Pri súboroch s veľkosťou 2 MB sa niekedy dokonca vyskytli problémy s načítaním súboru. Tieto isté problémy boli aj pri SVD súboroch a veľa z ních sa nepodarilo v aplikácii načítať. Ak sa súbor podarilo načítať, tak samotná generácia pri väčších súboroch trvala maximálne v desiatkách sekúnd. Niektoré výsledné HTML súbory prekročili hranicu 10 MB pri väčších vstupných XML súboroch.

[1] https://github.com/cmsis-svd/cmsis-svd/tree/4decfa8e40a0ba2d1f65f551886af74dbf67a46e/data

[2] https://www.keil.com/pack/doc/CMSIS/SVD/html/svd_Example_pg.html

7.1 Používateľské testovanie

Na ďalšie hodnotenie generátora som pripravil deväť jednoduchých úloh a jednu bonusovú. K tomu som vytvoril dotazník, ktorý spolu s riešením úloh vyplnilo 12 respondentov. K dotazníku som respondentom poslal krátky XML súbor a obrázky. Respondenti si v úlohách vyskúšali pridať komentáre a obrázky k elementom a tiež aj ich mazanie. Tiež si vyskúšali pridať aj hromadný komentár a obrázok a vygenerovať dokumentáciu. V dokumentácií mali za úlohu nájsť ich pridané komentáre a obrázky a vyskúšať si nájsť informáciu pomocou vyhľadávania v pravom hornom paneli. Ako poslednú úlohu si mali vyskúšať aj generovanie tabuliek a porovnať elementy s vypnutou a zapnutou generáciou tabuliek. Ako bonusovú úlohu mali respondenti možnosť samostatne vyskúšať generátor, konkrétne tým, že si mali nahrať vlastný XML súbor a k nemu pridať obrázky a komentáre. Následne mali za úlohu sledovať, ako generátor spracuje tieto vstupy a vytvorí z nich dokumentáciu. Po týchto úlohách následoval krátky dotazník s otázkami, ktoré mali zhodnotiť prácu s generátorom a výslednou dokumentáciou.

Okrem jedného respondenta všetci ostatní mali predchádzajúce znalosti o tom, čo obsahuje XML dokument. Taktiež, s výnimkou jedného účastníka, všetci úspešne dokončili zadané úlohy. Používatelia prevažne hodnotili prácu s generátorom ako pohodlnú, rýchlu a jednoduchú. Niektorým používateľom UI neprišlo úplne intuitívne a navrhujú pridať *help*/popis prvkov generátora priamo na stránke. Respondenti tiež odporúčajú niekoľko úprav vo frontende, ako napríklad možnosť nahrať XML súbor prostredníctvom jediného tlačidla, zbalenie načítaného XML súboru po úrovniach podobne ako v konečnej dokumentácii, a automatické rozbalenie elementov v prípade použitia vyhľadávania v dokumentácii. Ako veľké plus hodnotia možnosť generáciu tabuliek, keďže veľa XML súborov obsahuje opakujúce sa elementy. Jeden respondent vyskúšal nahrať pom.xml, čo je konfiguračný súbor pre projekt, ktorý používa nástroj Maven[3] a vyskúšal si na ňom generáciu tabuliek. Odporúčal zahrnúť aj obrázky a komentáre stĺpcových elementov vo forme legendy, kedže toto zatiaľ nie je podporované a podporuje sa iba nadpisový element. Používateľom sa páčila prehľadnosť výslednej dokumentácie, ale chýbala im možnosť iných výstupných formátov a viac možností vzhľadu výstupného HTML súboru. Odporúčali generátor rozšíriť o generovanie do LaTeXu/PDF a Markdown formátu. Tiež odporúčali pridať možnosť generovať výslednú dokumentáciu ako viacej HTML súborov, ktoré sú rozdelené do sekcií. Odporučili aj pridať možnosť nahrania viacerých XML súborov, ktoré by boli použité vo výslednej dokumentácii. Grafy z výberových odpovedí je možné nájsť v prílohe B.

[3]Systém na správu projektov a automatizáciu zostavovania, široko používaný v Jave.

7.2 Celkové zhodnotenie generátora

Skombinoval som moje poznatky nadobudnuté počas testovania s pripomienkami používateľov. Hlavné pozitívne či negatívne vlastnosti generátora sú zhrnuté v tabuľke 7.1.

Plusy	**Mínusy**
+ Jednoduchá a rýchla manipulácia s generátorom	- Performančný problém s väčšími XML súbormi (už pri 2 MB býva problém)
+ Prehladná výsledná dokumentácia	- Iba jeden výstupný formát
+ Hromadné pridávanie komentárov a obrázkov	- Možnosť naraz spracovať iba jeden vstupný XML súbor
+ Generácia tabuliek z opakujúcich sa elementov	- Chýbajú popisy ovládacích prvkov generátora
+ Možnosť zadať delimiter	- Chýbajúca legenda z komentárov zo stĺpcových elementov tabuľky

Tabuľka 7.1: *Hodnotenie plusov a mínusov generátora.*

Medzi najviac sľubné vlastnosti generátora patrí generácia tabuliek, ktorá predstavuje významnú oblasť pre ďalšie vylepšenia. Algoritmus zodpovedný za ich tvorbu by bolo možné zdokonaliť, najmä v oblasti výberu prvkov vhodných na zaradenie do tabuliek a efektívnejšieho zoskupovania zložitejších XML štruktúr. Anotácia elementov pomocou komentárov a obrázkov je tiež veľmi perspektívnym nápadom. Tento prístup by sa dal ďalej rozvíjať tak, aby umožňoval dopĺňanie elementov o ďalšie rôznorodé formáty. Pridanie možnosti hromadne pridávať komentáre a obrázky podľa tagu elementu predstavuje významnú výhodu. V budúcnosti by bolo prínosné túto funkciu ďalej rozšíriť tak, aby používateľ mohol ešte detailnejšie špecifikovať výber elementov, na ktoré sa anotácia aplikuje. Napríklad by bolo možné umožniť anotovanie všetkých podelementov určitého elementu, alebo presne definovať cestu, pre ktoré konkrétne elementy sa majú komentáre a obrázky pridať, vrátane možnosti výberu podľa atribútov elementov. Okrem toho, v súčasnosti by bolo vhodné navrhnúť implementáciu tak, aby efektívne zvládala spracovanie aj väčších vstupných XML súborov. To by zvýšilo robustnosť a škálovateľnosť generátora a umožnilo by plynulé spracovanie aj rozsiahlejších dátových súborov. Rovnako by bolo prospešné, aby generátor podporoval generáciu dokumentácie do viacerých formátov, ako sú LaTeX a Markdown, a prípadne ponúkol aj štruktúrovanejší vzhľad výslednej dokumentácie vo formáte HTML, podobne ako to robí nástroj Doxygen. Toto by rozšírilo možnosti použitia generátora a zvýšilo jeho atraktivitu pre širšie spektrum používateľov.

Kapitola 8

Záver

V rámci mojej práce sa mi podarilo úspešne navrhnúť a implementovať multiplatformný generátor technickej dokumentácie. Tento nástroj načíta vstupný XML súbor a poskytuje používateľovi webové rozhranie, ktoré ponúka jednoduchú a rýchlu manipuláciu s XML dátami. Generátor je schopný vytvoriť štruktúrovanú prehľadnú dokumentáciu vo viacerých farebných schémach, ponúka možnosť zoskupovania elementov do tabuliek, umožňuje rozdeliť atribúty podľa delimitra, a navyše umožňuje pridávať k elementom komentáre a obrázky. Generátor bol následne testovaný mnou a skupinou vybraných respondentov, ktorí vykonali sériu jednoduchých úloh. Tento postup mi umožnil zhromaždiť cenné spätné väzby na to, ako sa im s nástrojom pracuje a aká je ich spokojnosť s kvalitou a čitateľnosťou vygenerovanej technickej dokumentácie.

V budúcnosti vidím hlavný prínos v rozšírení mojej bakalárskej práce o schopnosť spracovávať väčšie XML súbory, čo je kľúčové pri spracovaní väčšieho objemu dát. S väčšími XML súbormi by bolo vhodné aj naviazať na rozdelenie výslednej dokumentácie na viacej HTML súborov, kde by napríklad element mohol predstavovať jeden súbor. Toto rozdelenie zvýši prehľadnosť a uľahčí navigáciu v dokumentácii, čo je obzvlášť dôležité pri komplexných alebo rozsiahlych dátových štruktúrach. Tieto vylepšenia by mali značne zlepšiť používateľskú prívetivosť a škálovateľnosť generátora, čím sa rozšíri jeho aplikovateľnosť na rôzne typy projektov a potreby používateľov. Ďalším zaujímavým rozšírením mojej práce by bolo pridanie viacerých výstupných formátov, ako sú LaTeX a Markdown. Tieto formáty pridajú veľkú flexibilitu a užitočnosť generátora, pretože umožnia používateľom vybrať si formát, ktorý najlepšie vyhovuje ich konkrétnym potrebám a kontextu použitia.

Prácu vnímam ako prínos v oblasti preskúmavania možností rýchleho a prehľadného zobrazovania dát uložených v XML formáte s možnosťou používateľského vstupu. Pre tvorcov dokumentácie je kľúčové, aby proces tvorby nebol časovo náročný a aby technická dokumentácia bola pre čitateľov ľahko zrozumiteľná a prehľadná.

Literatúra

[1] *About draw.io* [online]. Draw.io [cit. 2023-12-19]. Dostupné z: https://www.drawio.com/about.

[2] Behrmann, G., David, A. a Larsen, K. G. *A Tutorial on Uppaal 4.0*. November 2006 [cit. 2023-12-15]. Dostupné z: https://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf.

[3] *Data* [online]. Merriam-Webster, 2023 [cit. 2023-11-04]. Dostupné z: https://www.merriam-webster.com/dictionary/data.

[4] David, A., Larsen, K. G., Legay, A., Mikučionis, M. a Poulsen, D. B. *Uppaal SMC Tutorial*. Január 2018 [cit. 2023-12-15]. Dostupné z: https://uppaal.org/texts/uppaal-smc-tutorial.pdf.

[5] *DOMParser* [online]. [cit. 2023-12-10]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/DOMParser.

[6] *Doxygen homepage* [online]. Doxygen [cit. 2024-01-03]. Dostupné z: https://www.doxygen.nl/.

[7] *Draw.io Documentation*. [cit. 2023-12-19]. Dostupné z: https://www.drawio.com/doc/.

[8] Education, I. C. *Structured vs. Unstructured Data: What's the Difference?* [online]. IBM, jún 2021 [cit. 2023-12-7]. Dostupné z: https://www.ibm.com/blog/structured-vs-unstructured-data/.

[9] *Getting Started* [online]. Markdown Guide [cit. 2024-01-08]. Dostupné z: https://www.markdownguide.org/getting-started/.

[10] Grinberg, M. *Flask Web Development*. 1. vyd. O'Reilly Media, 2014. ISBN 978-1-449-37262-0.

[11] Haramundanis, K. *The Art of Technical Documentation*. 1. vyd. Digital Press, 1991. ISBN 978-1555580803.

[12] Harold, E. R. a Means, W. S. *XML in a Nutshell*. 3. vyd. O'Reilly Media, 2004. ISBN 978-0-596-00764-5.

[13] *Introduction to Javadoc* [online]. Baeldung, január 2024 [cit. 2024-01-04]. Dostupné z: https://www.baeldung.com/javadoc.

[14] *Java API for XML Processing (JAXP) Tutorial* [online]. Oracle [cit. 2023-12-10]. Dostupné z: https://www.oracle.com/java/technologies/jaxp-introduction.html.

[15] *Javadoc* [online]. Oracle [cit. 2024-01-04]. Dostupné z: https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html.

[16] Kimber, E. *What is DITA?* [online]. XML.COM, január 2017 [cit. 2024-01-02]. Dostupné z: https://www.xml.com/articles/2017/01/19/what-dita/.

[17] Krosel, A. *What Is Technical Documentation? (And How To Create It)* [online]. Indeed, august 2023 [cit. 2023-12-20]. Dostupné z: https://www.indeed.com/career-advice/career-development/technical-documentation.

[18] *Lesson: Introduction to JAXB* [online]. Oracle [cit. 2023-12-10]. Dostupné z: https://docs.oracle.com/javase/tutorial/jaxb/intro/index.html.

[19] *Lxml - XML and HTML with Python* [online]. [cit. 2023-12-10]. Dostupné z: https://lxml.de/index.html.

[20] Nixon, M. *SEMI-STRUCTURED DATA 101* [online]. Snowflake, 2019 [cit. 2023-12-7]. Dostupné z: https://www.snowflake.com/guides/semi-structured-data-101.

[21] Olmstead, L. *12 Types of Technical Documentation* [online]. Whatfix, september 2022 [cit. 2024-01-12]. Dostupné z: https://whatfix.com/blog/types-of-technical-documentation/.

[22] Olmstead, L. *How to Create a User Guide* [online]. Whatfix, august 2023 [cit. 2024-01-12]. Dostupné z: https://whatfix.com/blog/user-guides/.

[23] Parwana, D. *A brief introduction to Doxygen* [online]. Medium, december 2022 [cit. 2024-01-03]. Dostupné z: https://medium.com/@diyar.parwana/a-brief-introduction-to-doxygen-434ce18f48a7.

[24] *RTF* [online]. FILEFORMAT [cit. 2024-01-08]. Dostupné z: https://docs.fileformat.com/word-processing/rtf/.

[25] Stockwell, F. *A History of Information Storage and Retrieval*. McFarland, 2001. ISBN 978-0-7864-3772-6.

[26] *SVD Description (*.svd) Format* [online]. Keil [cit. 2023-12-18]. Dostupné z: https://www.keil.com/pack/doc/CMSIS/SVD/html/svd_Format_pg.html.

[27] *System View Description* [online]. Keil [cit. 2023-12-18]. Dostupné z: https://www.keil.com/pack/doc/CMSIS/SVD/html/index.html.

[28] *System.Xml Namespace* [online]. Microsoft [cit. 2023-12-10]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/api/system.xml?view=net-8.0.

[29] *The ElementTree XML API* [online]. Python [cit. 2023-12-10]. Dostupné z: https://docs.python.org/3/library/xml.etree.elementtree.html.

[30] *Uppaal Documentation*. [cit. 2023-12-17]. Dostupné z: https://docs.uppaal.org/.

[31] Vijay. *XSLT Tutorial – XSLT Transformations & Elements With Examples* [online]. Marec 2024 [cit. 2024-03-15]. Dostupné z: https://www.softwaretestinghelp.com/xslt-tutorial/.

[32] Walsh, N. a Muellner, L. *DocBook The Definitive Guide*. 1. vyd. O'Reilly Media, 1999. ISBN 978-1565925809.

[33] *Xml2js* [online]. [cit. 2023-12-10]. Dostupné z: https://www.npmjs.com/package/xml2js.

[34] *XSL Transformations (XSLT) Version 2.0 (Second Edition)* [online]. W3, marec 2021 [cit. 2024-03-15]. Dostupné z: https://www.w3.org/TR/xslt20/.

Príloha A

Obsah priloženého pamäťového média

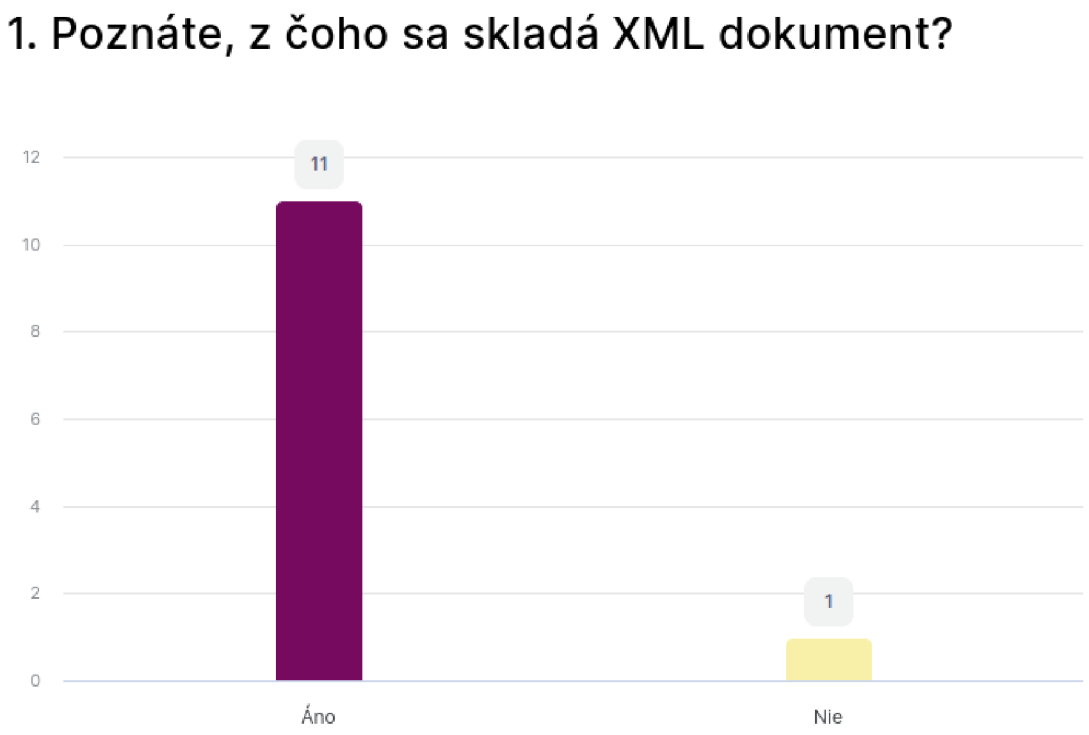
Nižšie prikladám časť adresárovej štruktúry pamäťového média. Priečinok **src** obsahuje zdrojový kód generátora a súbor *README.md*, ktorý popisuje, ako generátor nainštalovať, spustiť a ovládať. Zdrojové súbory textovej časti práce je možné nájsť v priečinku **src_latex**. Úlohy k testovaniu a výsledok dotazníka sú v priečinku **survey**. Testovacie XML súbory a príkladné vygenerované dokumentácie sa nachádzajú v priečinku **testing_files**.

```
/
├── src/
│   ├── Sessions/
│   ├── static/
│   │   ├── css/
│   │   ├── images/
│   │   └── js/
│   ├── templates/
│   │   ├── index.html
│   │   └── generator.html
│   ├── utils/
│   │   ├── common.py
│   │   └── html/
│   ├── views/
│   │   ├── generator.py
│   │   └── index.py/
│   ├── main.py
│   └── README.md
├── src_latex/
├── survey/
└── test_files/
    ├── Draw.io/
    ├── Generated_docs_examples/
    ├── SVD/
    ├── UPPAAL/
    └── user_testing/
```

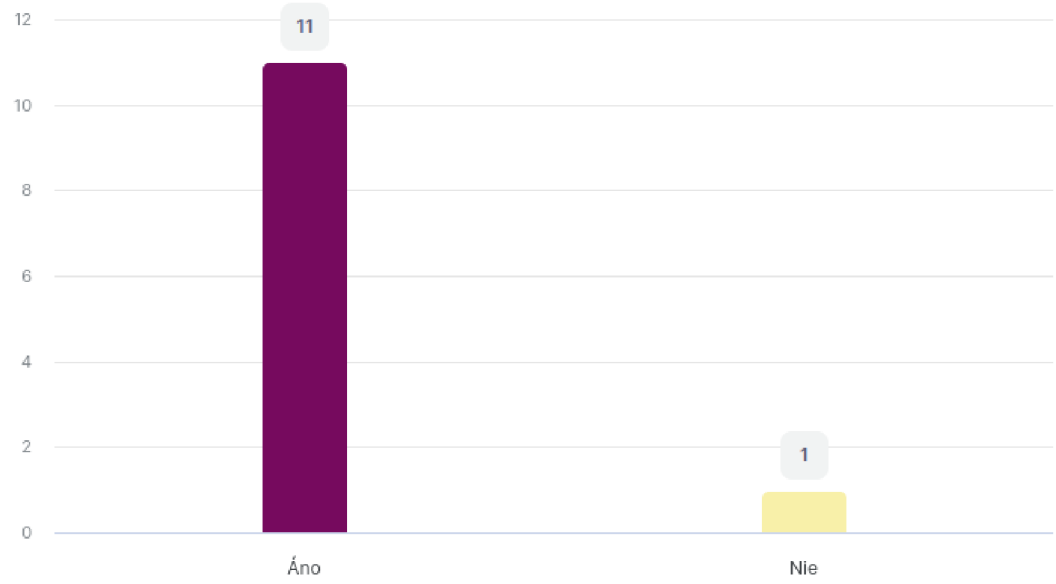
Príloha B

Dotazník

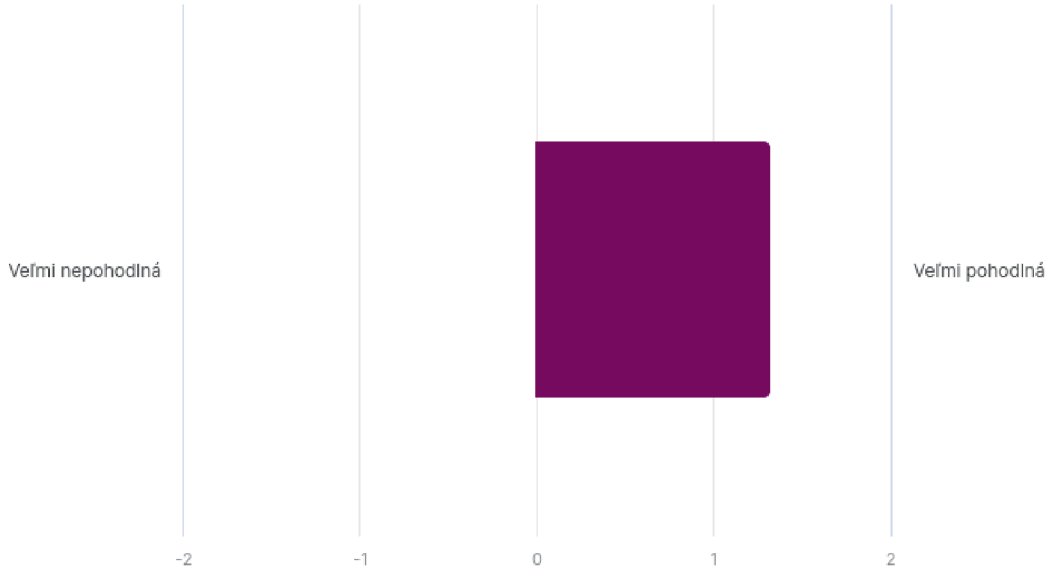
Výsledné grafy z odpovedí dotazníka, ktorý používatelia vyplnili spolu s otestovaním generátora. Konkrétne testovacie úlohy a slovné odpovede na plusy a mínusy generátora je možné nájsť v pamäťovom médiu v priečinku **survey**.



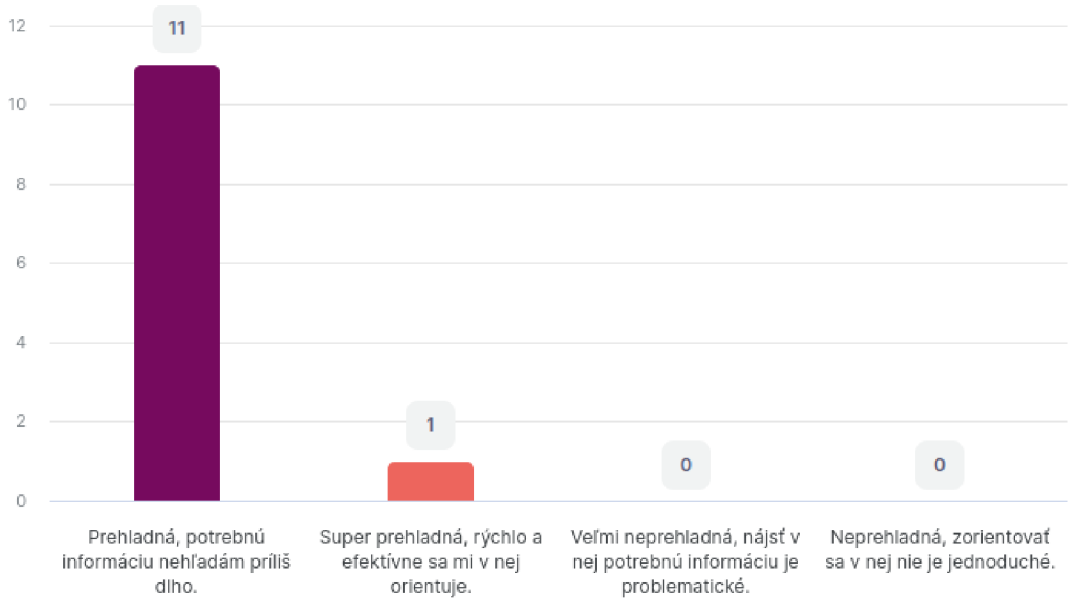
2. Podarilo sa vám splniť všetky úlohy?



3. Ako hodnotíte prácu s nástrojom?*



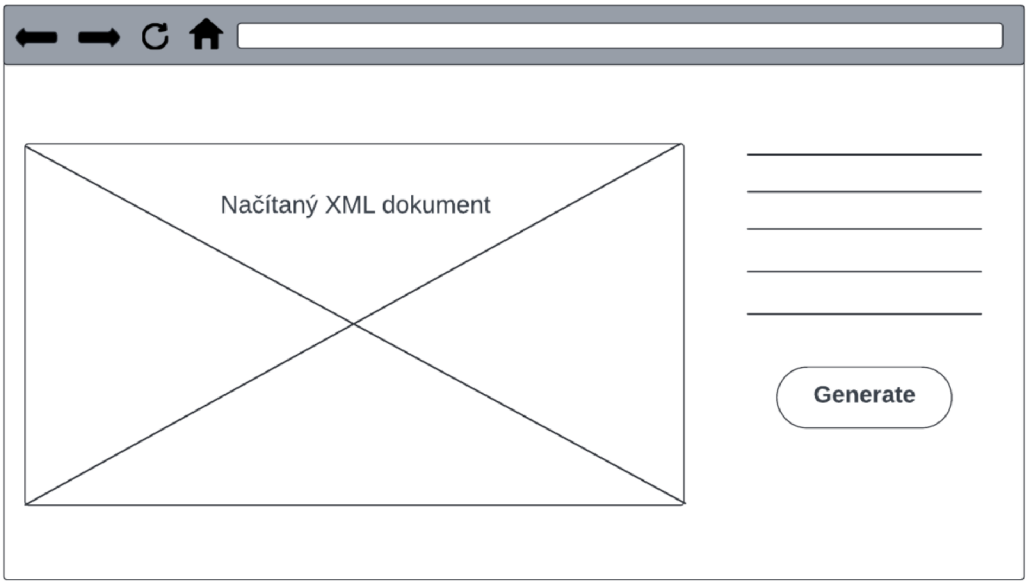
4. Výsledná dokumentácia je



Príloha C

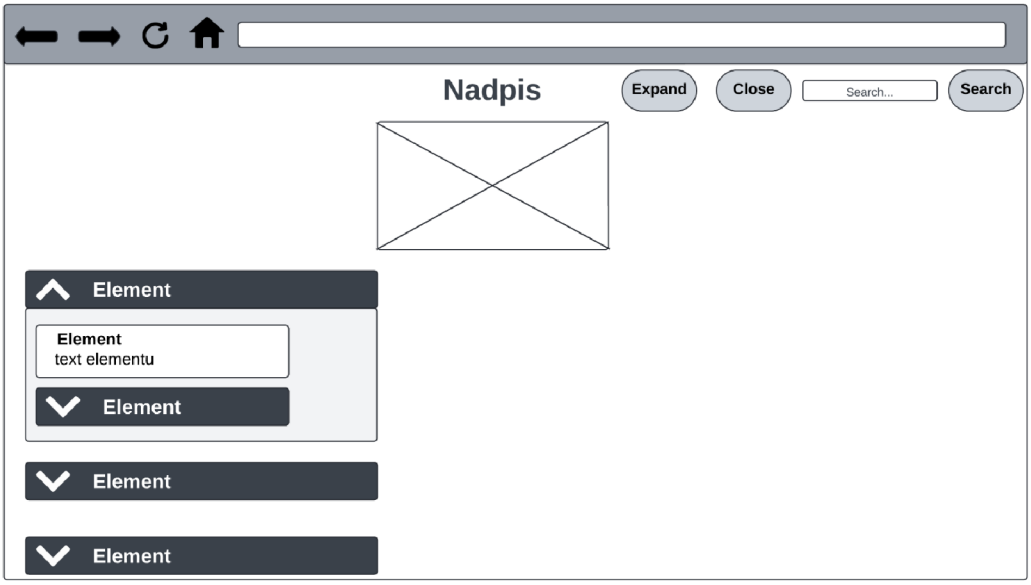
Wireframy

Zväčšené wireframy návrhu aplikácie z obrázku 5.3.

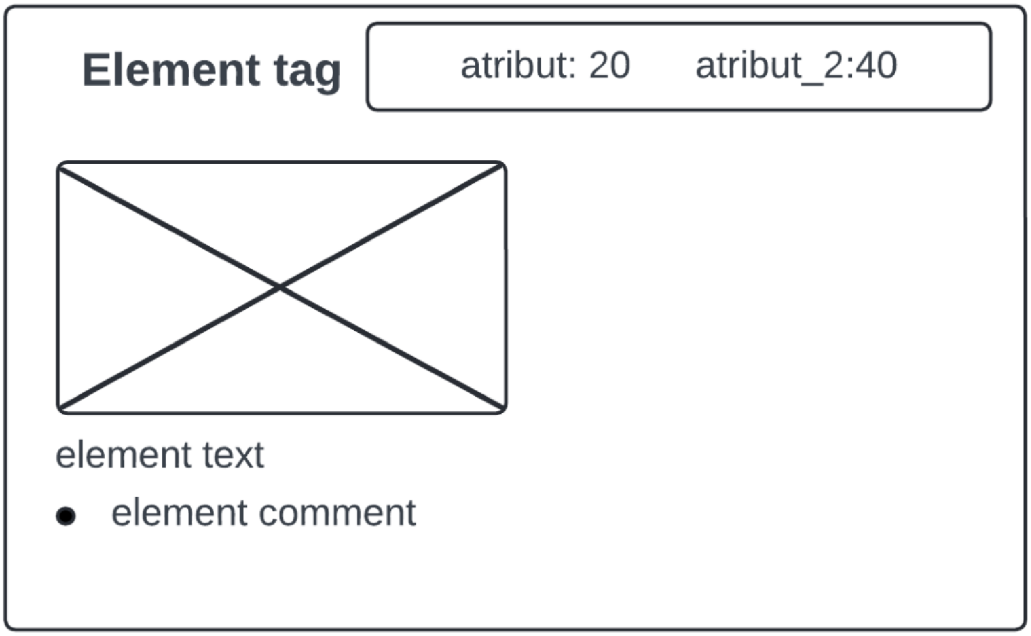


Obr. C.1: *Wireframe generátora dokumentácie.*

Obr. C.2: *Wireframe pridávania komentára a obrázku k XML elementu cez otváraciu značku.*



Obr. C.3: *Wireframe vygenerovanej dokumentácie.*



Obr. C.4: *Wireframe elementu obohateného o komentár a obrázok.*