Univerzita Karlova

Pedagogická fakulta

Katedra informačních technologií a technické výchovy

BAKALÁŘSKÁ PRÁCE

Systém pro generování statického webu

System for static web generation

Emil Miler

Vedoucí bakalářské práce: PhDr. Josef Procházka, Ph.D.

Studijní program: Specializace v pedagogice

Studijní obor: Informační technologie se zaměřením na vzdělávání

Praha 2020

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Děkuji hlavně svému vedoucímu, doktoru Procházkovi, který mě vždy nasměroval správnou cestou a měl se mnou trpělivost. Velmi děkuji Lukáši Hozdovi za jeho technickou a odbornou asistenci se statickými generátory a se sázením samotné práce, a dále také Honzovi Vaisovi za jeho cenné rady k sázení a k obecnému psaní závěrečnách prací. Také děkuji Albertu Pospíšilovi za jeho pomoc s překlady a korekturou. Dále pak děkuji partnerce, členům spolku *microlab* a ostatním za podporu a pomoc při tvorbě práce.

Název práce: Systém pro generování statického webu

Autor: Emil Miler

Katedra: Katedra informačních technologií a technické výchovy

Vedoucí bakalářské práce: PhDr. Josef Procházka, Ph.D., Katedra informačních technologií a technické výchovy

Abstrakt: Abstrakt.

Klíčová slova: www web generátor

Title: System for static web generation

Author: Emil Miler

Department: Name of the department

Supervisor: PhDr. Josef Procházka, Ph.D.,

Abstract: Abstract.

Keywords: www web generator

Obsah

Úvod

1. Staticky generovaný web

Princip statické webové stránky sahá až ke vzniku WWW, kdy existovaly pouze stránky statické, tedy stejné pro každého uživatele. Jejich obsah může být průběžně aktualizován, ovšem negenerují se zvlášť pro každého uživatele na základě různých proměnných. U statických webů tedy dochází k vytvoření čistého HTML ve chvíli, kdy je změněn zdrojový obsah, nebo kdy autor ručně spustí generátor. (PC Magazine, 2020)

Dynamické stránky jsou generovány speciálně pro každého uživatele na základě jeho nastavení, různých vstupů, proměnných a dalších vlastností. Ke generování dochází ve chvíli, kdy si uživatel stránku vyžádá, nikoliv předem, jako je tomu u staticky generovaných stránek. (PC Magazine, 2017)

1.1 Výhody statických webových stránek

Pro sdílení statického obsahu mezi různé uživatele stačí velmi jednoduchý HTTP server bez jakýchkoliv dalších modulů typu *PHP*, *Python* a dalších systémů, které by obsah dynamicky generovaly například z dat vytažených z databáze, nebo z uživatelského vstupu. Na straně serveru tedy nedochází ke zpracování obsahu těsně před jeho odesláním uživateli, čímž se v komunikaci mezi klientem a serverem se drasticky snižuje „Time To First Byte“[1] a tím dochází ke snížení celkové latence. (Hoffman, 2013)

Snížení samotné latence může pozitivně přispět ke spokojenosti uživatelů, což dokazuje nespočet výzkumů na toto téma, například analýza z webového portálu Financial Times, kde se uvádí, že rychlost webové stránky negativně ovlivňuje hloubku jejího užívání, ať už je odezva sebemenší. Jak je zde rovněž uvedeno, data ukazují, že pohledu uživatelské spokojenosti a finančního dopadu existují jasné a důležité výhody při zrychlení webové stránky. Z tohoto výzkumu se autoři rozhodli v následujících měsících od vydání analýzy zainvestovat více času do úprav všech aspektů jejich nové stránky FT.com s cílem jejího zrychlení.

Nechat opravit překlad

Eliminováním dynamického obsahu se také předchází nevyžádaným vstupům od uživa-

[1] Time To First Byte — čas mezi odesláním požadavku a přijmutím prvního bajtu dat.

tele, které mohou být i cílené na prolomení bezpečnostních nedostatků webové aplikace a v některých případech mohou vést k úniku citlivých dat, převzetí kontroly útočníka nad webovou aplikací nebo celým serverem, podstrčení falešných dat uživateli a mnoho dalším běžně se stávajícím útokům. Statický web eliminuje tento problém, jelikož nemá žádný uživatelský vstup.

Sledování a analýze nejčastějších chyb webových aplikací a serverů se věnuje organizace OWASP[2], která vydává aktualizované seznamy a statistiky. Podle OWASP byly v roce 2017 nejčastější tyto chyby a bezpečnostní nedostatky:

1. Injekce
2. Rozbitá autentizace
3. Odhalení citlivých dat
4. XML External Entities (XXE)
5. Nefunkční řízení přístupu
6. Špatná konfigurace zabezpečení
7. Cross-Site Scripting (XSS)
8. Nezabezpečná deserializace
9. Užívání komponent se známými zranitelnostmi
10. Nedostatečné logování a monitorování

(OWASP, 2017)

Většina těchto chyb se vztahuje právě k dynamickým webovým aplikacím. Bezpečnost tedy závisí nejen na programátorovi který aplikaci vytváří, ale také na tom, že programovací jazyk je bezpečně implementován. To nelze tvrdit o nejpoužívanějším jazyce PHP, který nejen že obsahuje spousty chyb, viz seznam nalezených bezpečnostních děr (CVE Details, 2020a), ale zároveň nevede programátora ke psaní bezpečného kódu a ve

[2]The Open Web Application Security Project — `https://owasp.org/`.

výsledku vzniká nebezpečná aplikace, pokud si autor nedá pozor na správné ošetření vstupů a další bezpečnostní aspekty programu.

Skvělým příkladem chybovosti dynamických webů je systém Wordpress, ve kterém jsou každý rok nalezeny desítky bezpečnostních chyb (CVE Details, 2020b), přičemž mnoho dalších přibývá s instalací špatně napsaných rozšíření. Například na začátku roku 2020 byla nalezena bezpečnostní chyba v rozšíření, které bylo využíváno na více než dvě stě tisících webových stránkách a potencionálním útočníkům umožnila smazat obsah databáze (Khandelwal, 2020). Na konci roku 2019 umožnila chyba ve dvou nezabezpečených rozšířeních neautorizované přihlášení k účtu administrátora bez použití hesla (Khandelwal, 2019).

Údržba velkých webových aplikací je také často problematická. Kód je nutné udržovat v návaznosti na aktualizace daného jazyka, databázového systému a dalších věcí. Těmto aktualizacím se z bezpečnostních důvodů nelze vyhýbat. Statický web nemusí udržovat funkční propojení s databázemi a různými frameworky a je tedy mnohem méně náročný na dlouhodobou údržbu. Při zvolení správného generátoru není nutná ani údržba šablon a celý systém při zachování stejného prostředí nepřestane fungovat. Protože statický generátor nepracuje s uživatelským vstupem, vyhýbá se bezpečnostním chybám a tím i nutným aktualizacím.

Lepši slovo?

Jako každý jiný systém, i statické generátory mají své nevýhody. Hlavním z problémů je to, že správa statického generátoru a tvorba obsahu je náročnější, než klasické webové rozhraní s administračním panelem, různými uživateli a jednoduchou správou pro běžné technicky nenadané uživatele. Pro přidání nebo úpravu obsahu je nutné pracovat s lokálními soubory ve stromové struktuře a při generování je často potřebný zásah do shellu[3]. Tvorba systému pro automatizované generování je také náročnější než instalace některého z běžných CMS[4]. (Cimpanu, 2015)

[3]Program pro interpretování příkazů v prostředí příkazové řádky.

[4]Content Management System

1.2 Princip generátorů

Generátor statického obsahu je tvořen ze tří hlavních částí. První částí jsou soubory šablon, které popisují rozložení stránky, vizuální vlastnosti, typografii, ale také vstupní a výstupní kódování a formáty. V podstatě definují jak a kam se bude obsah vkládat. Druhou částí je obsah obecně psaný v některém ze značkovacích jazyků, nejčastěji v jazyce Markdown. Obsah je strukturován do vlastních sekcí a souborů, aby bylo snadné rozlišit, do které části výsledné stránky patří. Třetí a poslední částí je samotné jádro generátoru, které zpracovává obsah, vkládá ho do šablon a renderuje statickou webovou stránku.

Většina generátorů zároveň umí pracovat s konfiguračními soubory, kterými jde nastavit chování generátoru na jednom centralizovaném místě. Část z nich má také integrovaný jednoduchý web server, který umožňuje autorovi náhled výstupních stránek během tvorby obsahu.

(Cimpanu, 2015)

2. Webová paradigmata

Ve světě webových stránek se setkáváme se spoustou forem a paradigmat, která se hodí pro obsažení různých druhů informací. Neexistují žádné formální zařazení druhů webových stránek do skupin, ovšem některé webové portály se pokouší určit základní druhy webů, které se na Internetu objevují. Na základě těchto portálů a jejich rozřazení do skupin[1][2][3], které jsou často mířené na specifický obsah, lze vytvořit tři základní paradigmata, do kterých lze tyto weby zařadit. Jsou jimi:

Přesunout odkazy pod jednu položku.

- Webová prezentace
- Index všeobecných informací
- Technická dokumentace
- Sociální sítě a fóra

V této práci byl ke každému z paradigmat vybrán systém vhodný pro generování a správu daného druhu obsahu. Výjimkou je skupina sociálních sítí a fór, kde staticky generovaný obsah není z důvodu často se měnícího obsahu vhodným řešením.

2.1 Webová prezentace

Nejbližší původním webům z dob vzniku WWW jsou webové prezentace, tedy stránky s jednoduchým obsahem, které slouží k předání informací čtenáři například formou článků. Do této skupiny lze zařadit portfolia, blog, online noviny a časopisy, firemní stránky, foto alba a podobně. Tento druh stránek se skvěle hodí ke statickému generování obsahu, který se odesílá všem uživatelům stejný a nemění se často.

Jako nejvhodnější systém pro generování webových prezentací byl vybrán software Zola. Ten je oproti jiným systémům výhodný tím, že je napsaný v jazyce Rust a je tedy snadno rozšiřitelný a mnohem rychlejší, než jeho alternativy.. I stěmitop výhodami si

Citovat benchmark

[1] http://www.xislegraphix.com/website-types.html

[2] https://www.hostgator.com/blog/popular-types-websites-create

[3] https://www.quora.com/What-are-the-different-types-of-websites

zachovává spousty funkcí a ryců, které lze na jít v ostatních složitých systémech. Také je možné generátor zkompilovat do jednoho staticky linkovaného binárního souboru, se kterým se pracuje mnohem lépe, než se složitým frameworkem.

2.2 Index všeobecných informací

Za obecného zástupce tohoto druhu stránek lze považovat Wikipiedii, která podnítila vznik spousty jiných takzvaných „Wiki systémů“ a stránek.

2.3 Technická dokumentace

Na rozdíl od Wiki stránek se technická dokumentace liší organizováním svého obsahu, který je cílený na přesný popis systému či objektu.

3. Značkovací jazyky pro popis obsahu

3.1 Principy značkovacích jazyků

Vysvětlení principu značkovacích jazyků, nebo také takzvaně „makrup jazyků“, můžeme najít například v RFC 7764[1], tedy že v počítačových systémech jsou kontextuální data ukládána a zpracována několika technikami. Informaci lze kódovat jako čistý text bez speciálních formátovacích znaků. Tento přístup je jednoduchý pro implementaci i použití, ovšem neumožňuje složitější formátování textu.

Kódovat lze můžeme i do binárních dat určených ke zpracování a interpretaci specializovaným programem. Zřejmou nevýhodou je to, že zdroj není čitelný bez programu určeného pro jeho interpretaci.

Markup jazyky se snaží o spojení toho nejlepšího z obou světů, tedy o obsah čitelný v čistém textu s možností formátování. To je dosaženo tím, že běžným znakům jsou přiděleny speciální významy nedefinované původní znakovou sadou. Uživatel je schopen tyto znaky psát jako čistý text a vyjádřit tím speciální význam. Například v rámci jazyka Markdown se znak `#` změní z běžného křížku na definování nadpisu první úrovně, nebo také kombinace znaků `<p>` značí začátek odstavce v HTML. (Leonard, 2016)

3.2 Nejběžnější jazyky

Ke dnešnímu dni vnikl nespočet značkovacích jazyků. Nejpoužívanějším z nich jednoznačně HTML, ovšem tato práce se věnuje těm nejpoužívanějším jazykům, které mají uživateli usnadnit psaní a sázení obsahu. Uživatel tedy nemusí nutně řešit typografii a formátování obsahu při jeho psaní, tedy o věci, o které se později stará generátor pomocí šablon. U HTML je tomu naopak, kdy uživatel řeší samotný obsah i formátování v jednu chvíli skrze různé druhy formátovacích tagů. O vyplňování obsahu do HTML

[1] Jako *RFC* se označují standardy vydané organizací IETF (Internet Engineering Task Force).

se v případě staticky generovaných webů stará právě samotný generátor.

Vybrané jazyky jsou zároveň cílené na čitelnost samotného zdrojového obsahu v čistém textu bez nutnosti jeho interpretace speciálním prostředím či zpracováním do jiného formátu, například do PDF, DjVu, PostScript apod. Například podtržení textu je v nějakém pseudo-jazyce reprezentováno opravdovým podtržením pomocí spojovníků, nikoliv obalením nadpisu ve speciální deklaraci, jako je tomu například u HTML. Podtržení je poté pro čtenáře mnohem jasnější, jelikož nemusí přemýšlet, co v případě HTML daný tag vůbec způsobuje, ale podtržený vyplývá z kontextu.

Seznam nejoblíbenějších jazyků je sestaven podle aktuálních statistik ze serveru Slant, který se věnuje obecnému určení oblíbenosti na základě hodnocení ze strany uživatelů. (Slant, 2020)

3.2.1 Markdown

Vznik jazyka Markdown byl 14. prosince roku 2014, když John Gruber vydal jeho první popis syntaxe a referenční implementaci.

Hlavním z cílů syntaxe jazyka je vytvářet co možná nejčitelnější obsah v syrové podobě. Dokument psaný v Markdownu by měl být publikovatelný sám o sobě jako čistý text bez dalších úprav a zpracování. Jazyk byl ovlivněn několika již existujícími specifikacemi jiných jazyků, ovšem největším zdrojem inspirace pro jeho vznik jsou čisté emailové korespondence. (Gruber, 2004)

První specifikaci Gruber vydal společně s referenční implementací v jazyce Perl, která slouží pro konverzi Markdownu do HTML. Tento program je také pojmenován jako „Markdown“, ovšem mluvíme-li o „Markdownu“, máme nejčastěji na mysli samotnou syntaxi. Ta je dnes již implementována v mnoha různých jazycích a programech. Gruberova specifikace ovšem není formální standard, kvůli čemuž vznikl veliký počet alternativních a více či méně pozměněných implementací, které nemusí být navzájem kompatibilní. Nejčastějšími z nich jsou například Github Markdown, CommonMark, R Markdown a mnoho dalších. (MacFarlane, 2019)

Nevyužívanější formální specifikací je právě CommonMark[2], který slouží jako pevný základ většiny rozšíření. (Martí, 2017).

Podobně jako je tomu u specifikací, existuje velké množství programů, které tyto různé specifikace překládají. Švýcarským nožem mezi nimi je program Pandoc[3], který umí překládat Markdown do enormního výběru jiných formátů, nebo z jiných formátů zpět. Tato funkcionalita se nezvtahuje pouze na jazyk Markdown, ovšem Pandoc dokáže operovat mezi všemy podporovanými formáty, například dokáže konvertovat obsah z HTML do TEXu. Na druhou stranu existují i velmi jednoduché překladače, například program smu[4], který umí překládat Markdown do HTML nebo čistého textu a neobsahuje více než 600 SLOC[5], tedy řádků kódu hlavního programu.

3.2.2 Org-mode

Org-mode vznikl jako jeden z módů pro editor Emacs. Funguje podobně jako ostatní markup jazyky, tedy jako jeden centrální systém pro správu obsahu, ze kterého lze vytvářet jiné formáty, například HTML, LATEX, Open Document, Markdown, PDF a

České slovo?

podobně s možností přidání libovolného nového backendu. Cílem Org-mode je možnost ho používat i s minimální úrovní jeho znalosti, ovšem jeho funkcionalita je vždy přístupná. Vše je realizováno pouze na čistých textových souborech, nejlépe přenositelným typem souboru. Editor emacs je zároveň velmi často protován na různé druhy stsémů a je tedy možné ho využívat v podstatě kdekoliv. (The Org Mode Developers, 2020)

Podporuje také „literate programming“ a „reproducible research“, tedy že Org soubory mohou obsahovat plně funkční bloky s kódem, které lze evaluovat v rámci systému a výstup bloků lze automaticky vkládat přímo do dokumentu. (Schulte a kol., 2012)

Jak popisuje Carsten Dominik ve svém krátkém technickém popisu, Org-mode umí navrhování, psaní poznámek, hypertextové odkazy, tabulky, seznamy, plánování projektů, GTD, HTML a LATEX, a to všechno v čistých textových souborech v editoru Emacs. (Dominik, 2008)

[2] `https://commonmark.org/`
[3] `https://pandoc.org/`
[4] `https://github.com/Gottox/smu`
[5] Source lines of code

3.2.3 AsciiDoc

...

3.2.4 reStructuredText

...

3.2.5 TEX

Tento jazyk se již vzdaluje od původního konceptu čitelnosti zdroje, ovšem ve statických generátorech ho lze stále efektivně využít a jeho části se velmi často objevují jako rozšíření dříve zmíněných jazyků. Jedním z hlavních rozšíření jsou zápisy matematických rovnic, které z TEXu vychází.

Rozšířit o popis TeXu a matiky.

Většina uživatelů se setkala spíše s jazykem LATEX, tedy s nadstavbou původního TEXu, která má uživateli zjednodušit práci svými makry a rozšířeními. Realita je ovšem taková, že LATEX dělá celou práci složitější, jak popisuje doktor Olšák:

> Představte si, že si nějaký uživatel přečte LATEXovou příručku a nabyde dojmu, že mu bude stačit rozumět problematice sazby na úrovni této příručky. Pak se jednou překlepne třeba při sestavování tabulky a na terminálu na něj TEX křičí: `Extra alignment tab has been changed to "\cr".` Uživatel začne znovu listovat ve své příručce a zjistí, že tam o žádném `"\cr"`není jediná zmínka. Má pak tři možnosti: (1) Zmáčkne Enter a podobně se zachová i u dalších chyb. Pomyslí si, že ten LATEX je něco tajemného a mystického. (2) Propadne zoufalství a jde od toho. Dojde k závěru, že je lepší zůstat u Wordu. Vždyť stačí vzít tabulku v Excelu a jednoduše ji přemístit do Wordu a jaképak smolení se s nějakým podezřelým `"\cr"`. (3) Pořídí si TEXbook a po intenzivním studiu nakonec řekne: „aha“. V tuto chvíli ale už nepotřebuje, aby mu LATEX zakrýval složitost TEXu.

(Olšák, 1997)

Ve výsledku je tedy lepší, z různých důvodů popsaných doktorem Olšákem v jeho publikaci, použít samotný plain TEX na úkor vyšší vstupní úrovně pro používání jazyka.

3.2.6 Troff

4. Taxonomie požadavků pro modelový web

Jako modelovoá implementace byl zvolen web pro distribuci výukových materiálů. Webové stránky byly objednány Ústavem výzkumu a rozvoje vzdělávání Pedagogické fakulty Univerzity Karlovy za účelem usnadnění práce již aktivních učitelů a jsou tedy plně využívány v praxi mnoha pedagogy z celé republiky. Materiály jsou určeny pro podporu výuky během vyhlášeného stavu nouze v době šíření viru COVID-19 a mají učitelům pomoci s přípravou distanční výuky a úkolů v době vyhlášení celostátní karantény. Tuto implementaci lze ovšem použít na distribuci jakýchkoliv jiných výukových materiálů, či ke psaní dokumentace.

4.1 Obecná kritéria

Stránky musí být staticky generované a není tedy žádoucí v rámci webu řešit uživatelské účty, přihlašování apod. Zároveň je důležité, aby byl obsah zobrazitelný na každém druhu zařízení, tedy jak na monitorech s nadstandardní velikostí, tak na mobilních zařízeních. Z důvodu potencionálního vytížení sítě je nutné, aby byl celý obsah optimalizován za účelem předejití vysoké latence, a to z důvodů probíraných v předchozí části práce, tedy v sekci 1.1.

Citace obecných kritérií

4.2 Kritéria specifická pro modelový web

Hlavním požadavkem je možnost dělit obsah na sekce dle druhu školy (základní škola, střední škola, vysoká škola atd.) a dále pak na subsekce podle předmětů a oborů. Do samotného obsahu musí být možné vkládat přílohy ke stažení v různých formátech, obrázky a videa s možností jejich ocitování. Všechny přiložené soubory musí být distribuovatelné přímo z webových stránek, nikoliv s externích zdrojů. Všechna videa je

nutné vložit do stránky a musí je být možné přehrát v nativním přehrávači prohlížeče bez nutnosti otevírání externích webových stránek či programů.

Obsah stránek musí být verzován, decentralizován a spravován předem pověřenými uživateli. Generování statického webu na základě změn obsahu je nutné řešit automatizovaně.

4.3 Kritéria pro šablony a design

5. Modelová implementace

Tato část práce se věnuje tvorbě modelové implementace systému pro generování statického webu dle definovaných požadavků v sekci 4. Systém je vytvářen na základě poznatků z předchozích částí práce.

5.1 Výběr vhodného systému

Modelový web se skládá ze dvou sytstémů, a to ze systému pro správu obsahu a systému pro jeho generování do HTML.

Pro správu obsahu i šablon a statických souborů byl zvolen distribuovaný verzovací systém Git. Hlavní výhodou tohoto verzovacího systému je jeho rozšířené využití v praxi a dokáže s ním tedy pracovat spousta uživatelů. Zároveň má v porovnání s jinými verzovacími systémy spousty výhod.

Příklady + citace

Protože forma modelového webu odpovídá paradigmatu webové prezentace ze sekce 2.1, byl pro jeho generování použit program Zola[1], jehož výhody jsou v sekci 2.1 popsány.

5.2 Tvorba šablony

Jak se uvádí v dokumentaci[2], Zola pracuje s několika druhy stránek, primárně s takzvanou „sekcí“ a „stránkou“. Každá sekce může mít vlastní obsah, ovšem může obsahovat i další subsekce, díky čemuž lze dělit obsah do stromové struktury. Stránka slouží pouze k předání obsahu a nikoliv k dalšímu větvení struktury. Dá se tedy říci, že stránka značí konec dané větve. Kořenem celého stromu je speciální sekce s názvem „index“. Každá tato část standardně využívá vlastní HTML šablonu, ovšem nejde o pravidlo a každá část větve může využívat jinou šablonu. To je užitečné například u stránek s různým druhem obsahu. V rámci modelového webu zůstává druh obsahu stejný a není tedy třeba odchylovat se od standardní struktury.

[1] `https://www.getzola.org/`

[2] `https://www.getzola.org/documentation/content/overview/`

Soubory se šablonami se nachází ve složce `templates/`, ve které generátor vždy očekává šablonu `index.html`. Ta se využívá jak k vykreslení úvodní kořenové stránky, tak ji mohou ostatní šablony rozšiřovat. Tato kořenová šablona tedy obsahuje základní strukturu celé stránky, přičemž navazující šablony jen mění určité části obsahu a nedefinují celou strukturu znovu.

Generátor v šablonách hledá vlastní řídící sekvence, které se popisují závorkami. Existují tři druhy kombinací, které lze použít:

- `{% %}` – Metoda, funkce, cykly, podmínky, práce s proměnnou atd.
- `{{ }}` – Výpis do HTML
- `{# #}` – Komentář

Generátor také vyžaduje konfigurační soubor `config.toml` v kořenové složce projektu, který obsahuje různé nastavení stránky, globální proměnné a chování generátoru.

Program 5.1: Příklad jednoduché konfigurace v souboru `config.toml`

```
# Adresa ze které se generují odkazy
base_url = "https://ucitelonline.pedf.cuni.cz"
# Název stránky
title = "Učitel online"
# Popis stránky
description = "Web pro ditstribuci užitečných materiálů"
# Zda se bude zpracovávat CSS systémem Sass
compile_sass = true
```

Tohle je pěkná ukázka, vyberte klidně ještě jednu dvě, které jsou něčím zajímavé, typické, nebo naopak výjimečné pro ilustraci toho, co chcete o daném, systému sdělit.

Systém vždy zpracuje úvodní šablonu `index.html`, ze které pak lze odvíjet ostatní šablony. Tato hlavní šablona obsahuje strukturu celé webové stránky a nesmí v ní tedy chybět validní HTML struktura, tedy hlavička, tělo, metadata, kódování a podobně. Do struktury lze vkládat libovolné řídící sekvence pro generátor, které ovlivňují výsledný výstup.

Program 5.2: Základní šablona `index.html`

```
<!DOCTYPE html>
<html lang="cs">
<head>
    <meta charset="UTF-8">
    <title>{{ config.title }}</title>
</head>
<body>
</body>
</html>
```

V příkladu 5.2 je název stránky mezi tagy `<title></title>` vyplněn generátorem. Ten do šablony vloží hodnotu konstanty `config.title`, která je nastavena v konfiguračním souboru `config.toml` z příkladu 5.1. Názvem stránky bude tedy řetězec „Učitel online“. Generátor dokáže převzít kteroukoliv konstantu z kontextu konfiguračního souboru.

Všechny direktivy lze v rámci generátoru navazovat na sebe, podobně jako je tomu v Unixových systémech. Spojování funkcí a filtrů se provádí znakem `|`, stejně jako v POSIX[3] shellu, kde výstup jednoho příkazu se stane vstupem příkazu navazujícího. Například je možné název stránky vypsat ve velkých písmenech i přesto, že v konfiguračním souboru je formátován pouze s velkým písmenem na začátku. K převedení na velká písmena slouží filtr `upper`. Názvem stránky bude po zpracování programem 5.3 řetězec „UČITEL ONLINE“.

Program 5.3: Základní šablona s filtrem pro přepsání názvu na velká písmena

```
<!DOCTYPE html>
<html lang="cs">
<head>
    <meta charset="UTF-8">
    <title>{{ config.title | upper }}</title>
</head>
<body>
```

[3]Portable Operating System Interface – Rodina standardů Unixových systémů

```
</body>
</html>
```

V šabloně je také možnost vytvořit bloky, které lze v navazujících šablonách měnit. K vysvětlení principu fungování bloků je možné název stránky z příkladu 5.3 obalit blokem `title` a těla vložit blok `content`.

Program 5.4: Využití bloků v šabloně z příkladu 5.3

```
<!DOCTYPE html>
<html lang="cs">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}{{ config.title | upper }}{%
       endblock %}</title>
</head>
<body>
{% block content %}
    Ahoj, světe!
{% endblock %}
</body>
</html>
```

Název stránky zůstane stejný a v jejím těle přibude text „Ahoj, světe!“. Vytvoříme-li novou šablonu s názvem `section.html`, generátor nám umožní rozšířit ji o původní šablonu `index.html` a měnit pouze definované bloky. Není tedy nutné znovu definovat celou strukturu stránky. Pro importování, nebo-li rozšíření šablony, slouží direktiva `extends`.

Program 5.5: Definice nové šablony `section.html` rozšiřující šablonu z příkladu 5.4

```
{% extends "index.html" %}
{% block title %}{{ config.title | upper }} &ndash; {{ section.
   title }}{% endblock %}
{% block content %}
    Toto je obsah kategorie.
{% endblock %}
```

Šablona `section.html` se v rámci generátoru Zola implicitně využívá pro všechny existující sekce[4]. Názvem stránky v této šabloně bude, stejně jako u hlavní šablony, název stránky z konstanty `config.title` definované v konfiguračním souboru, ale také spojovník a název dané sekce. Za pseudo-výstup lze považovat například „UČITEL ONLINE – základní a střední škola“, bude-li se uživatel nacházet v sekci pro základní a střední školy.

V bloku s obsahem bude původní obsah „Ahoj, světe!“ nahrazen za řetězec „Toto je obsah kategorie“. Ten ovšem nechceme definovat přímo v šabloně, nýbrž cílem generátoru je vyplňovat obsah ze zdrojových souborů v sázecím jazyce, viz. sekce 1.2. Zola pro vkládání obsahu využívá stejný princip jako v ostatních případech, tedy vypsání obsahu proměnné, v tomto případě proměnné `section.content`, která obsahuje zkompilované HTML z daného Markdown souboru. Zároveň je dobrou praktikou provést vyčištění vstupu filtrem `safe`[5].

Program 5.6: Vkládání obsahu ze zdrojového Markdown souboru

```
{% extends "index.html" %}
{% block title %}{{ config.title | upper }} &ndash; {{ section.
   title }}{% endblock %}
{% block content %}
    {{ section.content | safe }}
{% endblock %}
```

Z principu by žádný obsah neměl být definován přímo v šabloně, nýbrž by měl být do stránky vkládán generátorem z proměnných, nebo ze sázeného obsahu. V rámci modelové implementace je toto nepsané pravidlo dodržováno.

5.3 Automatické generování vícevrstvé navigace

Je následující část srozumitelná?

[4] `https://www.getzola.org/documentation/content/section/`

[5] `https://tera.netlify.com/docs/#safe`

Obsah modelové implementace je dělen do stromové datové struktury o potencionálně nekonečné hloubce, kdy každá část větve je v rámci generátoru vlastní kategorií, nikoliv stránkou. Pro modelovou implementaci bylo zvoleno, aby navigace byla generována v návaznosti na aktivní cestu ve stromě. Ve stránce jsou dvě různé navigace, jedna hlavní a vždy vidiětelná, která obsahuje rozdělení obsahu dle škol a druhá navigace, která zobrazuje aktivní větev stromu.

První vrstvou struktury jsou hlavní sekce, v rámci implementace pojemnované jako L_1, které jsou vypsány vždy ve vlastní navigaci. Pod touto navigací se zobrazuje seznam všech kategorií, které vybraná položka v L_1 obsahuje. Pokud uživatel zvolí kteroukoliv položku v L_2, v navigaci se objeví další sloupec, který obsahuje všechny podkategorie vybrané položky, tedy položky ve vrstvě L_3. Takto lze stromem procházet potencionálně do nekonečna. Styly modelové šablony ovšem počítají s maximální hloubkou čtyř subkategorií.

Tato funkcionalita je implementována pomocí dvou cyklů. První cyklus (příklad 5.7) se provádí pro všechny rodiče aktivní ktegorie vrstev L_2,L_3,...,L_n, kde n je aktuílní vrstva. V každé iteraci se mění kontext, ve kterém generátor pracuje. Z daného kontextu generátor vypisuje všechny subkategorie každého svého rodiče. V druhém cyklu (příklad 5.8) se vypisují všichni potomci dané stránky, tedy potomci ve vrstvě L_{n+1}.

Program 5.7: Cyklus pro vypisování všech rodičů v dané větvi navigace

```
{% if section.ancestors %}
    {% for s in section.ancestors %}
        {% if loop.index < 2 %}{% continue %}{% endif %}
        <ul>
        {% set s = get_section(path=s) %}
        {% for s in s.subsections %}
            {% set s = get_section(path=s) %}
            <li><a href="{{ s.permalink }}"
                {% if current_path == s.path %}
                    class="active"
                {% elif current_path is containing(s.path) %}
                    class="ancestor"
                {% endif %}
```

```
            >{{ s.title }}</a></li>
        {% endfor %}
        </ul>
    {% endfor %}
{% endif %}
```

Program 5.8: Cyklus pro vypisování všech potomků dané stránky do navigace

```
{% if section.subsections %}
    <ul>
    {% for s in section.subsections %}
        {% set s = get_section(path=s) %}
        <li><a href="{{ s.permalink }}">{{ s.title }}</a></li>
    {% endfor %}
    </ul>
{% endif %}
```

5.4 Rozšíření šablony

Ve výchozím stavu generátor neumí vkládat nic jiného, než je uvedeno ve specifikaci CommonMark, viz. sekce 3.2.1. Dle požadavků modelového webu je nutné, aby generátor uměl vkládat videa přímo do stránky. Taková funkcionalita není soušástí specifikace CommonMark a je tedy potřeba rozšířit generátor. Nejvhodnějším způsobem přidání vlastních funkcionalit je využití filtrů, které se v rámci generátoru nazývají „shortcode“.

Principem vlastních filtrů je to, že si uživatel vytvoří vlastní šablonu, kterou lze vyvolat speciální řídící sekvencí přímo z obsahu. Každý tento shortcode může pracovat s libovolným množstvím proměnných a po zpracování vloží do místa vyvolání zkompilovaný HTML kód. Lze tedy tvrdit, že shortcode je v své podstatě imperativní funkce, která umí pracovat s parametry.

Pro tvorbu těchto filtrů je v generátoru Zola určena složka `templates/shortcodes`, která obsahuje jejich HTML šablony a kód pro zpracování generátorem. Název HTML

souboru definuje název vlastního filtru. Vytvoříme-li uvitř této složky soubor nazvaný `video.html`, budeme v obsahu schopni využívat vlastní filtr s názvem `video`.

Program 5.9: Příklad jednoduchého filtru s jedním atributem

```
<video controls><source src="{{ src }}"></video>
```

V příkladu 5.9 bude filtr očekávat atribut `src` a bude vracet jednoduchý HTML kód pro vložení videa do stránky. Tento filtr lze vyvolat kdekoliv v obsahu, tedy v kterémkoliv souboru s koncovkou `.md`. Za názvem filtru se do závorky uvádí parametry oddělené čárkou. U posledního parametru se čárky nevuvádí, což platí i v případě, kdy se uvádí pouze jeden parametr, jako je tomu v příkladu 5.10.

Program 5.10: Vyvolání vlastního filtru s jedním parametrem

```
{{ video(src="video.webm") }}
```

V rámci vybraného generátoru není nutné specifikovat atributy na jeden řádek a lze je pro přehlednost vypisovat na více řádků, jako tomu je například u programu 5.13, zůstane-li dodržena koncepce oddělení atributů čárkou, tedy že poslední atribut vždy zůstane bez čárky. Výstupem této direktivy bude následující HTML kód.

Program 5.11: Výstup direktivy z příkladu 5.9

```
<video controls><source src="video.webm"></video>
```

Součástí požadavků pro modelový web jsou i citace přiložených souborů a videí. Existující filtr je tedy třeba rozšířit o možnost přiložení různých metadat. Tato metadata ovšem nejsou pro vložení videa povinná. Ve specifikaci vlastních filtrů lze využívat všechny operátory, které generátor nabízí. Nejlepším přístupem k tomuto problému je tedy využití jednoduchých podmínek, které kontrolují, zda je každá z hodnot zadána jako parametr a v případě že ano, vepíše se do obsahu. Atributy ošetřené podmínkami tedy nejsou povinné, zatímco nevyplněný atribut `src` by při generování vyvolal chybu. V následujícím příkladu jsou přidány podmínky pro kontrolu a případné vložení, jimiž jsou název videa (`title`), jméno autora (`author`) a rok vytvoření (`year`).

Program 5.12: Filtr pro vkládání videa s využitím podmínek

```
<video controls><source src="{{ src }}"></video>
```

```
{% if title or year and author %}
<div class="metadata">
    {% if title %}{{ title }}{% endif %}
    {% if author and year %}
        ({{ year }}, {{ author }})
    {% endif %}
</div>
{% endif %}
```

Filtr je možné opět vyvolat pomocí stejné direktivy kdekoliv v obsahu, ovšem nyní lze libovolně přidat parametry pro metadata.

Program 5.13: Vyvolání filtru 5.12 s formátováním na řádky

```
{{ video(
    src="video.webm",
    title="Název videa",
    author="Jméno autora",
    year="2020"
) }}
```

Protože byly zadány všechny povinné i nepovinné atributy, výtupem toho filtru budou i části kódu s metadaty.

Program 5.14: Výstup direktivy z příkladu 5.13

```
    <video controls><source src="video.webm"></video>
<div class="metadata">
    Název videa (2020, Jméno autora)
</div>
```

Pro modelový web byla zvážena možnost vypisování obsahu automaticky, tedy že program zkontroluje složku s obsahem a pokud narazí na soubor se specifikovanou koncovkou, vypíše jej do obsahu podle daných pravidel. Generátor Zola umožňuje prohledávání složek a práci se soubory, v rámci Zoly takzvanými „assety“. Tuto funkcionalitu lze tedy implementovat jednoduchým cyklem a filtem, které zpracují všechny případné soubory ve složce dané stránky. Zoubory lze filtrovat mnoha způsoby, z nichž je nejuniverzálnější

funkce `matching()`, která dovoluje filtrovat vstup regulárními výrazy dle implementace regex v jazyce Rust[6]. V následujícím příkladu je pro ilustraci této funkcionality implementován program vypisující obrázky s předem definovanými koncovkami.

Program 5.15: Automatický výpis obrázků s pevně definovanými koncovkami

```
{% if section.assets %}
    {% for asset in section.assets %}
        {% if asset is matching("\.(?i:jpg|gif|png)$") %}
            <img src="{{ get_url(path=asset) }}" alt="{{ asset
               }}">
        {% endif %}
    {% endfor %}
{% endif %}
```

Toto řešení ovšem není ve výsledném modelu implemntováno, protože jedním z požadavků je možnost vkládání souborů na libovolné místo v obsahu. Na stejném principu je ovšem vytvořen filtr pro vládání souborů, který tento požadavek splňuje. Výhodou filtru je, že ho lze vyvolat kdekoliv v obsahu a není vázán na pevně dané místo v šabloně. Ten očekává alespoň jeden parametr uvádějící název souboru bez koncovky, pro dle kterého pak filtr vyhledá všechny různé formáty s tímto názvem a ty vloží do stránky. Druhým libovolným parametrem je název souboru, který se do stránky vloží místo názvu souboru. to umožňuje uivateli volně pracovat s názvy souborů v souborvé struktuře bez ovlivnění obsahu stránky.

Program 5.16: Filtr pro výpis souborů s automatickým hledáním

```
{% if section.assets and filename %}
<div class="file">
    <div class="title">
        {% if title %}
            {{ title }}
        {% else %}
            {{ filename }}
        {% endif %}
```

[6] https://docs.rs/regex/1.3.6/regex/

```
    </div>
    {% for asset in section.assets %}
        {% if asset is matching(section.path ~ filename ~ "\..*
           $") %}
            <a href="{{ get_url(path=asset) }}" class="format">
               {{ asset | split(pat=".") | last }}</a>
        {% endif %}
    {% endfor %}
</div>
{% endif %}
```

V první části filtr zkontroluje, zda byl vyplněn parametr `title` a v případě že ano, nastaví ho jako název souoru v obsahu. V opačném případě využije název souboru samotného. Ve druhém kroku nastává kontrola, zda se ve složce nacházejí soubory (mimo hlavní soubor `_index.md`) a pokud ano, přes všechny soubory se iteruje kontrola, zda soubor splňuje podmínku názvu. Kontrola této podmínky je tvořena kombinací proměnných generátoru a regulárního výrazu. Každý soubor který splňuje podmínku je poté vypsán do obahu jako přímý odkaz k jeho stažení.

Jako text v odkazu se použije koncovka souboru, která se získává spojením několika filtrů, tedy filtru `split(pat=".")`, který rozdělí řetězec podle znaku tečka do pole a navazující filtr `last` vrátí poslední položku v poli. Tím filtr získá samotnou koncovku souboru.

Filtr lze vyvolat stejně, jako je tomu u filtru pro vkládání videa. Název filtru je opět definován názvem souboru `tmeplates/shortcodes/document.html` a bude jím tedy název `document()`.

Program 5.17: Vyvolání filtru 5.16

```
{{ document(
    filename="pracovni-list",
    title="Pracovní list"
) }}
```

V příkladu 5.17 je definován i nepovinný atribut `title`, který kvůli přehlednosti umož-

ňuje nastavit název. Atribut `filename` definuje název souboru ve složce bez koncovky. Všechny soubory, které chce uživatel vypsat, musí tedy mít stejný název a musí se lišit pouze koncovkou. Jsou li ve složce soubory s názvem `pracovni-list` a koncovkami `pdf`, `odt`, `djvu` a `ps`, bute výstupem filtru následující HTML.

Program 5.18: Výstup direktivy z příkladu 5.17

```
<div class="file">
    <div class="title">Pracovní list</div>
    <a href="pracovni-list.pdf">pdf</a>
    <a href="pracovni-list.odt">odt</a>
    <a href="pracovni-list.djvu">djvu</a>
    <a href="pracovni-list.ps">ps</a>
</div>
```

5.5 Optimalizace

Optimalizace modelové implementace je provedena na základě článku ze serveru Calomel, který se věnuje sestavením užitečných rad pro optimalizaci webových stránek na serverech s omezeným připojením do sítě a pro zvýšení spokojenosti uživatelů z užívání optimalizovaného webu, jak je rozebráno v sekci 1.1.

> Provozování webserveru může být hodnotná zkušenost, ale zároveň může být i zkouškou trpělivosti. Chcete svým uživatelům předávat všechny vaše stránky a obrázky, ovšem máte jen omezenou šířku pásma, pomocí které můžete data přenášet. Pokud přetížíte své pipojení, klienti nevštěvující váš web server si budou myslet, že je pomalý a neresponzivní. Je tedy třeba webový server nastavit tím nejlepším možným způsobem s cílem získat co nejvíce návštěv a zlepšit zážitek vašim návštěvníkům. Následující rady slouží ke snížení zátěže serveru, ke zrcyhlení odesílání stránek a k zastavení nechtěného a škodlivého provozu.

(Calomel, 2017)

Lze cistovat vlastní překlad takhle v bloku? Mám někde specifikovat, že jde o můj vlastní překlad?

5.6 Správa obsahu a verzování

Statické stránky neumožňují správu uživatelů v prámci webové aplikace, tedy že se případný editor nebo administrátor přihlásí a upravuje obsah klikáním, či psaním ve WYSIWYG[7] editoru. Správu uživatelů lze jednoduše řešit omezením přístupu na web server, kde jen oprávnění uživatelé mohou do obsahu zasahovat. To je ovšem velmi těžkopádné řešení, protože neumožňuje práci více uživatelům najednou a neudržuje předešlé verze obsahu a historii úprav. Lepší alternativou je využití některého verzovacího systému. Pro účely modelové implementace byl vybrán distribuovaný verzovací systém Git, jak je vysvětleno v sekci 5.1.

[7]What You See Is What You Get – Princip editoru který během psaní formátuje text tak, jak bude ve výsledku vypadat, například LibreOffice Writer atd.

6. Vyhodnocení modelové implementace

6.1 Návrhy pro rozříšení systému

6.2 Implementace rozšíření

Závěr

Seznam použité literatury

Calomel (2017). Webserver optimization and bandwidth saving tips. https://calomel.org/save_web_bandwidth.html. Cit. 2020-03-23.

Cimpanu, C. (2015). How static site generators work. https://web.archive.org/web/20200316165614/https://news.softpedia.com/news/How-Static-Site-Generators-Work-482007.shtml. Cit. 2020-03-16.

CVE Details (2020a). Php : Vulnerability statistics. https://www.cvedetails.com/product/4096/Wordpress-Wordpress.html?vendor_id=2337.

CVE Details (2020b). Wordpress : Vulnerability statistics. https://www.cvedetails.com/product/4096/Wordpress-Wordpress.html?vendor_id=2337.

Dominik, C. (2008). Technical description in 24 words. https://orgmode.org/worg/org-quotes.html. Cit. 2020-04-15.

Gruber, J. (2004). Markdown. https://web.archive.org/web/20200227143926/https://daringfireball.net/projects/markdown/. Cit. 2020-02-27.

Hoffman, B. (2013). Improving search rank by optimizing your time to first byte. https://web.archive.org/web/20190416124447/https://moz.com/blog/improving-search-rank-by-optimizing-your-time-to-first-byte. Cit. 2020-02-12.

Khandelwal, S. (2019). Flaw in elementor and beaver addons let anyone hack wordpress sites. *The Hacker News*.

Khandelwal, S. (2020). Critical bug in wordpress theme plugin opens 200,000 sites to hackers. *The Hacker News*.

Leonard, S. (2016). Guidance on markdown: Design philosophies, stability strategies, and select registrations. RFC 7764, Internet Engineering Task Force. URL https://tools.ietf.org/html/rfc7764.

MacFarlane, J. (2019). Commonmark spec. https://spec.commonmark.org/. Cit. 2020-03-22.

Martí, V. (2017). A formal spec for github flavored markdown. https://github.blog/2017-03-14-a-formal-spec-for-github-markdown/. Cit. 2020-03-23.

Olšák, P. (1997). Proč nerad používám latex. http://petr.olsak.net/ftp/olsak/bulletin/nolatex.pdf.

OWASP (2017). Owasp top ten 2017. Technical report, OWASP.

PC Magazine (2017). Definition of: dynamic web page. https://web.archive.org/web/20170117040526/https://www.pcmag.com/encyclopedia/term/42199/dynamic-web-page. Cit. 2020-02-12.

PC Magazine (2020). Definition of: static web page. https://web.archive.org/web/20200223095514/https://www.pcmag.com/encyclopedia/term/static-web-page. Cit. 2020-02-12.

Schulte, E., Davison, D., Dye, T. a Dominik, C. (2012). A multi-language computing environment for literate programming and reproducible research. *Journal of Statistical Software*, **46**(3), 1–24. ISSN 1548-7660. URL http://www.jstatsoft.org/v46/i03.

Slant (2020). What are the best markup languages? https://web.archive.org/web/20200210061112/https://www.slant.co/topics/589/~best-markup-languages. Cit. 2020-02-10.

The Org Mode Developers (2020). *The Org Manual*.

Seznam programů