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 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 z pohledu uživatelské spokojenosti a finančního dopadu existují jasné a důležité výhody při zrychlení webové stránky. Z tohoto výzkumu se autoři rozhodli v měsících po vydání analýzy investovat více času do úprav všech aspektů jejich nové stránky FT.com s cílem jejího zrychlení. (Chadburn a Lahav, 2016)

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 mnoha dalším běžným 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. Nezabezpečená 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 mnoho chyb, viz seznam nalezených bezpečnostních děr (CVE Details, 2020a), ale zároveň nevede programátora k psaní bezpečného kódu, což

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

má za následek nebezpečené aplikace, pokud si autor nedá pozor na správné ošetření vstupů a dalších bezpečnostních aspektů programu.

Podstatným 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 rozšíření, která postrádají bezpečnostní prvky. 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 umožňovala 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 často problematická. Kód je nutné udržovat v návaznosti na aktualizace daného jazyka, databázového systému a dalších aspektů. 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.

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, méně technicky zaměřené 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

Ekosystém generátoru statického obsahu je tvořen ze tří hlavních složek. 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 samotný, napsaný v některém ze značkovacích jazyků, nejčastěji v jazyce Markdown. Obsah bývá strukturován do sekcí a souborů, aby bylo snadné rozlišit, do které části výsledné stránky patří. Třetí a poslední složkou 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 globální chování generátoru. Část z nich také integruje jednoduchý webserver, který umožňuje autorovi náhled výstupních stránek zatím co tvoří obsah.

(Cimpanu, 2015)

2. Webová paradigmata

Ve světě webových stránek se setkáváme se spoustou forem a paradigmat, která se hodí pro zpracování různých druhů informací. Neexistuje 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 mnohem rychlejší a bezpečnější, než většina jeho alternativ (Gouy, 2020). Kromě těchto

[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`

výhod si zachovává většinu funkcí a rysů, které lze najít 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 Wikipedii, 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

Definici konceptu značkovacích jazyků, nebo-li „markup 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 můžeme i do binárních formátů 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 nejlepšího z obou světů, tedy o obsah s možností formátování, který je jednoduše čitelný jak pro člověka, tak pro stroj. Toho je dosaženo tím, že v je v běžných textových souborech přiřezen vybraným znakům speciální význam. Uživatel je schopen tyto znaky psát bez potřeby speciálních nástrojů a tím jednoduše vyjádřit 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

V současnosti existuje nespočet značkovacích jazyků. Nejpoužívanějším z nich je jednoznačně HTML, ovšem tato práce se věnuje těm nejpoužívanějším jazykům, které mají uživateli usnadnit psaní a sázení obsahu. Uživatel se tedy nemusí při tvorbě nutně zabývat typografií a formátováním obsahu, což jsou aspekty, o které se později postará generátor pomocí šablon. U HTML je tomu naopak, uživatel řeší samotný obsah i for-

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

mátování v jednu chvíli skrze různé druhy formátovacích tagů. O vyplňování obsahu do HTML 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 kontextu HTML daný tag znamená, kdežto podtržení 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

Jazyka Markdown vznikl 19. března roku 2004, když John Gruber vydal 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 prováděla konverzi Markdownu do HTML. Tento program je také pojmenován jako „Markdown“, ovšem mluvíme-li o „Markdownu“, máme nejčastěji na mysli samotnou syntaxi. Ta má dnes mnoho implementací v různých programovacích jazycích. Gruberova specifikace ovšem není formálním standardem, 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 nevztahuje pouze na jazyk Markdown, Pandoc dokáže operovat mezi všemi 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.

Užitečným rozšířením je, mimo jiné, také integrace matematického prostředí z jazyka TEX, viz sekce 3.2.5.

3.2.2 Org-mode

Org-mode vznikl jako jeden z módů pro editor Emacs[6]. 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 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 portován na různé druhy systémů 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 hodnotit 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 na-

[2] `https://commonmark.org/`

[3] `https://pandoc.org/`

[4] `https://github.com/Gottox/smu`

[5] Source lines of code

[6] `https://www.gnu.org/software/emacs/`

vrhování, psaní poznámek, hypertextové odkazy, tabulky, seznamy, plánování projektů, GTD, HTML a LaTeX, a to všechno v čistých textových souborech v editoru Emacs. (Dominik, 2008)

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. Je jedním z nejrozšířenějších sázecích jazyků se spoustou možností a funkcionalit, z nichž velmi zajímavým rozšířením je prostředí pro psaní matematických formulí, díky kterému jazyk stal velmi populárním v oblasti technických publikací. Tyto funkcionality se často objevují i v jiných jazycích, které jsou efektivně využívány pro jejich rozšíření.

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 nabude 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,

ľe 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

Tato kapitola se věnuje určení základních požadavků pro modelovou implementaci. Jsou zde shrnuta obecná kritéria, která platí pro většinu webových prezentací, a také kritéria specifická pro modelovou implementaci v rámci této práce. Dle těchto kritérií je poté samotná implementace tvořena v následující kapitole 5.

Jako modelová implementace byl zvolen web pro distribuci výukových materiálů a odkazů užitečných pro výuku. Tvorba těchto webových stránek je zadána Ústavem výzkumu a rozvoje vzdělávání Pedagogické fakulty Univerzity Karlovy za účelem usnadnění práce již aktivních učitelů v době šíření viru COVID-19. Stránky mají učitelům pomoci s přípravou distanční výuky a úkolů v době vyhlášení stavu nouze a celostátní karantény. Modelová implementace je tedy plně využívána v praxi mnoha pedagogy z celé republiky. Tuto implementaci lze ovšem použít na distribuci jakýchkoliv jiných výukových materiálů, či ke psaní a správě dokumentace.

4.1 Obecná kritéria

Jako zdroj obecných kriterií je použit článek ze serveru Calomel (2017), který se mimo jiné věnuje i optimalizacím, jež jsou dále popsány v sekci 5.5.

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.

Stránky by měly být udržovatelné i po předání jinému správci a celý systém by tedy měl být dostatečně zdokumentován. Také je důležité, aby byla zajištěna kompatibilita s nejběžněji používanými prohlížeči. Odkazy by měly být z důvodu přenositelnosti relativní, nikoliv směřující na absolutní cesty.

4.2 Kritéria specifická pro modelový web

Specifická kritéria jsou vytvořena na základě požadavků autorů obsahu, tedy učitelů, ze kterých každý má své specifické požadavky na funkce a vlastnosti, které musí obsah splňovat. Následující kritéria jsou souhrnem a kompromisem mezi všemi požadavky.

Stránky musí být staticky generované a není tedy žádoucí v rámci webu řešit uživatelské účty, přihlašování apod. Hlavním požadavkem pro strukturu stránky 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, tedy uvedení autora, názvu díla apod. Všechny přiložené soubory musí být distribuovatelné přímo z webových stránek, nikoliv z 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. V hlavičce každé stránky musí být možné specifikovat metadata: autora či seznam autorů obsahu, skupinu pro kterou je obsah určen a časovou dotaci.

Obsah stránek musí být možné spravovat předem pověřenými uživateli a jeho změny musí být zaznamenávány v decentralizovaném verzovacím systému. Generování statického webu na základě změn obsahu je nutné řešit automatizovaně bez dalších zásahů správce, či manuálního nahrávání nového obsahu na webserver.

4.3 Kritéria pro šablony a design

Obsah musí být snadno čitelný a zobrazitelný na každém druhu zařízení, tedy jak na monitorech s nadstandardní velikostí, tak na mobilních zařízeních. Zároveň musí být snadno čitelný, v nejlepším případě vysoko kontrastní černý text na bílém pozadí s dostatečnou velikostí. Navigace v obsahu musí být jednoduchá a intuitivní a vzhled celé stránky konzistentní. Na stránce nesmí přesahovat objem vizuálních elementů nad obsahem. Relevantní obsah by měl být na jednom místě, nikoliv rozdělený na několik různých stránek, mezi kterými musí uživatel přecházet.

5. Modelová implementace

Tato část práce se věnuje tvorbě modelové implementace systému pro generování statického webu dle definovaných požadavků v kapitole 4. Jsou zde vybrány vhodné součásti, ze kterých je modelová implementace složena. Systém je vytvářen na základě poznatků z předchozích částí práce.

5.1 Výběr vhodného systému

Modelový web se skládá ze dvou částí, a to z verzovacího systému pro správu obsahu a generátoru statického HTML.

5.1.1 Verzovací systém pro správu obsahu

Pro správu obsahu i šablon a statických souborů byl zvolen distribuovaný verzovací systém Git, který má v porovnání s jinými verzovacími systémy, zejména centralizovanými, spousty výhod. Hlavní jeho výhodou je rozšířené využití v praxi a snadné používání. Díky svým decentralizovaným vlastnostem ho lze využívat v mnoha odlišných pracovních postupech. S naklonovaným repozitářem lze pracovat i bez připojení k síti, což lze považovat i za druh zálohy. Git také umožňuje slučování různých změn od mnoha uživatelů a dovoluje jednoduše řešit potenciální konflikty. (Chacon, 2009)

Skvěle využitelnou funkcí pro modelovou implementaci je také to, že po provedení změn v repozitáři lze pomocí Gitu spouštět skripty, které mohou provádět automatické generování obsahu a další užitečné operace. Tato funkcionalita je implementována v rámci modelové implementace v sekci 5.6.1.

5.1.2 Generátor statického webu

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

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

2.1.

Přesunout výběr a výhody generátoru sem.

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 reprezentuje list v rámci stromovité struktury. Kořenem celého stromu je speciální sekce s názvem „index“. Každá tato část standardně využívá vlastní HTML šablonu, to není ovšem pravidlo a každá část větve může využívat jinou šablonu. To je užitečné například u stránek s různými druhy obsahu. V rámci modelového webu zůstává druh obsahu stejný a není tedy třeba odchylovat se od standardní struktury.

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 jako základ, kterou 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.

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

Fig. 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
```

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.

Fig. 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“.

Fig. 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>
</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`.

Fig. 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 %}
```

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

```
</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`.

Fig. 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, podobně jako u hlavní šablony, název stránky z konstanty `config.title` definované v konfiguračním souboru, ale také spojovník a název dané sekce. Za modelový 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].

Fig. 5.6: Vkládání obsahu ze zdrojového Markdown souboru

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

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

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

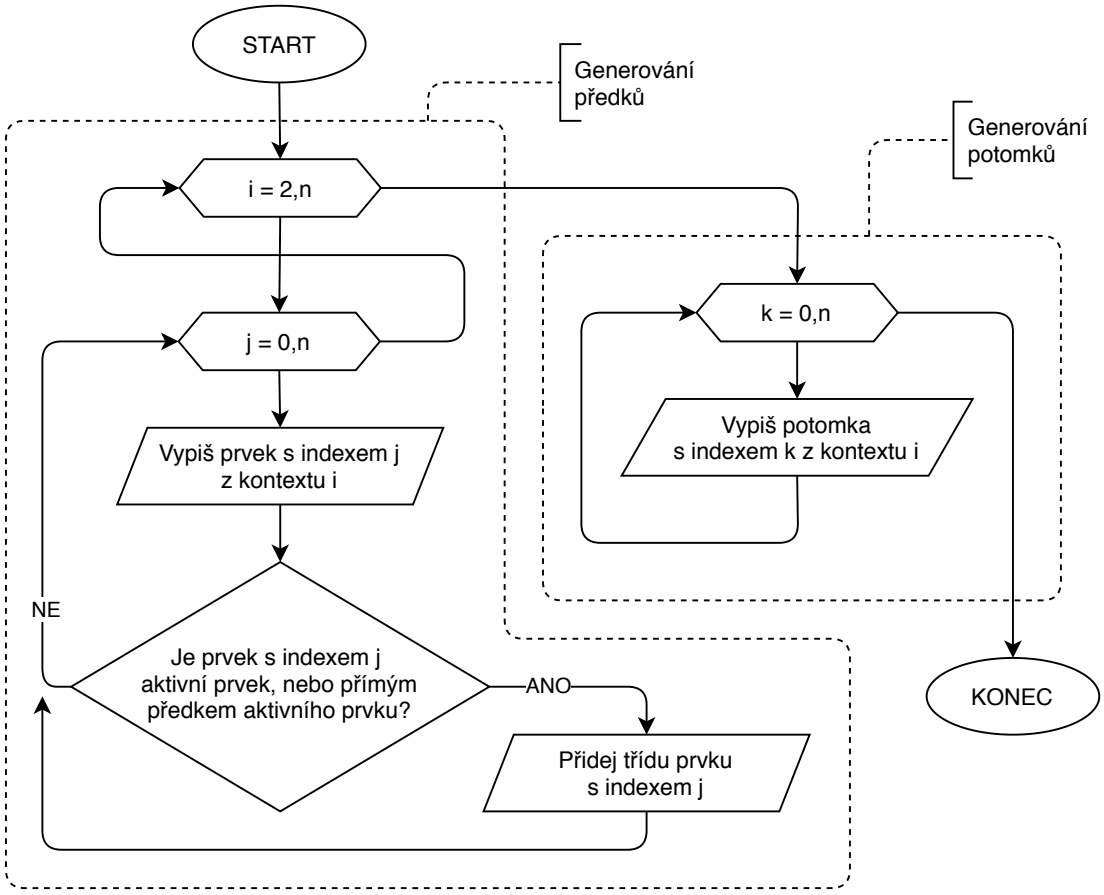
Z principu by žádný obsah neměl být definován přímo v šabloně, nýbrž by měl být do stránky vkládán generátorem z proměnných, nebo ze sázeného obsahu. V rámci modelové implementace je toto nepsané pravidlo dodržováno.

5.3 Automatické generování vícevrstvé navigace

Obsah modelové implementace je dělen do stromové datové struktury o potenciálně 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, hlavní, která je vždy viditelná a obsahuje rozdělení obsahu dle škol a vedlejší, která zobrazuje aktivní větev stromu.

První vrstvou struktury jsou hlavní sekce, v rámci implementace pojmenované jako L_1, které jsou vypsány vždy ve vlastní navigaci. Pod touto navigací je zobrazen 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 všechny podkategorie ve vrstvě L_3. Takto lze stromem procházet potenciálně do nekonečna. Styly modelové šablony ovšem počítají s maximální hloubkou čtyř subkategorií.

Tato funkcionalita je implementována pomocí tří cyklů, z nichž jeden je vložený. První cyklus (příklad 5.7) se provádí pro všechny rodiče aktivní kategorie vrstev L_2,L_3,...,L_n, kde n je aktuální vrstva. V každé iteraci se mění kontext, ve kterém generátor pracuje. Z daného kontextu generátor vypisuje pomocí vnořeného cyklem všechny subkategorie. Ve druhém cyklu (příklad 5.8) se vypisují všichni potomci dané stránky, tedy potomci



Obrázek 5.1: Diagram průběhu generování vícevrstvé navigace

ve vrstvě L_{n+1}.

Fig. 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 %}
```

Fig. 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 zpracovávat nic jiného, než co 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 součástí 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 pomocí speciální řídící sekvence přímo z obsahu. Každý tento shortcode může pracovat s libovolným množstvím proměnných a po zpracování vloží do místa vyvolání zkompilovaný HTML kód. Dá se tedy říci, že shortcode je v své podstatě 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 uvnitř této složky soubor nazvaný `video.html`, budeme v obsahu schopni využívat vlastní filtr s názvem `video`.

Fig. 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 neuvádí, což platí i v případě, kdy se uvádí pouze jeden parametr, jako je tomu v příkladu 5.10.

Fig. 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 dodržen způsob oddělování atributů čárkou, tedy že poslední atribut vždy zůstane bez čárky. Výstupem této direktivy bude následující HTML kód.

Fig. 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`).

Fig. 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 opět možné vyvolat pomocí stejné direktivy kdekoliv v obsahu, ovšem nyní lze libovolně přidávat parametry pro metadata.

Fig. 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, výstupem toho filtru budou i části kódu s metadaty.

Fig. 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 projde 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, pro které se v rámci Zoly používá termín „assety“. Tuto funkcionalitu lze tedy implementovat jednoduchým cyklem a filtrem, které zpracují

všechny případné soubory ve složce dané stránky. Soubory lze filtrovat mnoha způsoby, z nichž je nejuniverzálnější funkce `matching()`, která dovoluje filtrovat vstup regulárními výrazy dle jejich implementace v jazyce Rust[6]. V následujícím příkladu je pro ilustraci této funkcionality implementován program vypisující obrázky s předem definovanými koncovkami.

Fig. 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 implementováno, protože jedním z požadavků je možnost vkládání souborů na libovolné místo v obsahu. Na stejném principu je vytvořen filtr pro vkládání 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, podle 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 uživateli volně pracovat s názvy souborů v souborové struktuře bez ovlivnění obsahu stránky.

Fig. 5.16: Filtr pro výpis souborů s automatickým hledáním

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

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

```
            {{ filename }}
        {% endif %}
    </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 souboru 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, tak se iterativně zkontrolují všechny soubory, zda splňují podmínku názvu. Kontrola této podmínky je tvořena kombinací proměnných generátoru a regulárního výrazu. Každý soubor, který splňuje podmínku je poté vypsán do obsahu 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()`.

Fig. 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 umožňuje 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`, bude výstupem filtru následující HTML.

Fig. 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 (2017), 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.

Jak se na webu Colomel píše, 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 připojení, klienti navštěvující 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 zrychlení odesílání stránek a k zastavení nechtěného a škodlivého provozu.

Práce se věnuje pouze technickým optimalizacím spojených s tvorbou samotné webové stránky, nikoliv však optimalizacím sítě, web serveru a vizuálního návrhu. Nenačítá-li

se stránka během několika vteřin, většina uživatelů jednoduše odejde. Cílem této sekce je provést optimalizace, které urychlí načítání modelové implementace.

5.5.1 Typy a kvalita obrázků

Fotografie a grafika využívají mnohem více dat pro přenos než běžný HTML text a je tedy nutné provést optimalizaci (kompresi) obrázků na co nejmenší možnou velikost souborů. Obrázky není třeba renderovat na více než 72 dpi a pro každý druh grafiky je třeba zvolit vhodný formát, tj. formát JPEG pro fotografie a formáty PNG či SVG pro jednoduchou grafiku. Rastrové obrázky mají pouze potřebné rozlišení, tedy maximálně hodnotu největšího rozlišení, které se ve stránce bude zobrazovat. Klíčové je také nevyužívat obrázky v případě, kde je lze nahradit čistým HTML a CSS.

Obrázky ve formátu JPEG mají velice efektivní ztrátovou kompresi, pomocí které lze zredukovat velikost obrázku o značnou část. Autor článku tvrdí, že většinu obrázků lze komprimovat až o 50% bez viditelné ztráty na kvalitě. Své obrázky dokonce zkomprimoval ze 27 kilobajtů na pouhých 8 kilobajtů s JPEG kompresí 60%.

5.5.2 Ikona *favicon.ico*

Původně je *favicon.ico* výtvorem firmy Microsoft, kdy Internet Explorer automaticky odesílal požadavek na pevnou URL `/favicon.ico` od kořene webového serveru. Jde o malou ikonku, která se dnes zobrazuje u každé záložky s webovou stránkou. Problémem je, že se požadavkům o ní nelze vyhnout a vždy se počítá s tím, že ikona na web serveru existuje. Odesílá se vždy s každou stránkou a některé prohlížeče se po ní dotazují z neznámých důvodu dvakrát. Autor článku uvádí, že u některých serverů bylo až 30% přenesených dat využito jen na odesílání ikony.

Principem optimalizace je udržet ikonu co nejmenší, v nejlepším případě tak malou, že se vejde do jednoho TCP paketu, tedy do velikosti 1460 bajtů na většině systémů. Toho lze docílit tím, že ikona nebude větší než 16x16 pixelů s nízkou barevnou hloubkou, nejlépe s pouze čtyřmi barvami. Také je možné poslat pouze 1x1 pixelů veliký prázdný

obrázek, nebo vracet stavový kód 204[7] a neodesílat ikonu žádnou.

5.5.3 Obecné HTML optimalizace

Redukcí nepotřebných znaků v HTML lze také ušetřit značnou část přenosu dat. Dobrými praktikami mohou být:

- nepoužívání HTML komentářů,
- využití CSS pro formátování stránek,
- využití oddělovače nebo elementu `span` namísto tabulek,
- odstranění přebytečných tagů a prázdných mezer a řádků,
- vytvoření obrázkových náhledů namísto odesílání obrázků v plném rozlišení,
- recyklování již použitých obrázků a tlačítek.

Přesunout do návrhu pro rozšíření?

K odstranění přebytečných mezer, zalomení řádků, HTML komentářů a prázdných řádků lze použít automatický filtr, který provede kompresi výstupu. Generátor Zola provádí kompresi CSS, ovšem nemá zabudovanou funkcionalitu pro minifikaci výsledného HTML, která je v době psaní této práce vyvíjena[8].

Touto redukcí lze ušetřit 2% přenosu dat oproti ručně psanému neoptimalizovanému kódu. Je-li průměrná velikost stránky sto kilobajtů, lze touto optimalizací ušetřit dva kilobajty při každém odeslání stránky. Při odeslání sta tisíce stránek za měsíc je ve výsledku ušetřeno dvě stě megabajtů dat, které jsou jinak zbytečně odesílány uživatelům, kteří je stejně nezobrazí.

Další obecné rady pro optimalizaci HTML jsou uvedeny na serveru Yahoo! (2020), kde se uvádí spousta dalších způsobů ke zrychlení načítání stránky a k nižšímu vytížení sítě.

Připojením externích CSS a JavaScript souborů je umožněno jejich ukládání do paměti cache, což snižuje HTTP požadavky vůči serveru. Je-li obsah těchto souborů přímo ve

[7] 204 No Content – Server úspěšně zpracoval požadavek, ale nevrací žádný obsah.

[8] `https://github.com/getzola/zola/issues/542`

stránce, je odesílán pokaždé s novou stránkou a to vede ke zbytečnému vytěžování sítě. S tím souvisí i velikost stránek, kdy soubory větší než je daná maximální velikost se do mezipaměti neukládají a je proto dobré tuto velikost nepřekračovat.

U starých zařízení jsou pevně dané velikosti, například v roce 2011 byly limity 25.6K u iOS nebo 5M u Firefoxu. Mám je zde uvádět, i když se to rychle mění, nebo to stačí takhle obecně?

Připojením externího CSS přímo do hlavičky je umožněno progresivní vykreslování webové stránky, které urychluje „Time To First Byte“, viz sekce 1.1. Naopak umístěním případných JavaScript souborů až na konec celé stránky se prioritizuje načítání viditelného obsahu před méně důležitými skripty.

5.5.4 Videa a jejich vložení do stránky

Výhody CDN a problematika sledování uživatelů.

5.6 Správa obsahu a verzování

Statické stránky neumožňují správu uživatelů v rámci webové aplikace, tedy, že se případný editor nebo administrátor přihlásí a upravuje obsah klikáním, či psaním ve WYSIWYG[9] 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 5.1.1.

V tomto systému jsou soubory uloženy v repozitářích, kde každý projekt je vlastní repozitář. V rámci jednotlivých repozitářů se ukládají všechny změny obsahu prostřednictvím takzvaných „commitech“, nebo-li záznamů o provedených změnách včetně jejich krátkého popisu a autora. Tyto revize lze provádět v různých větvích repozitáře a větve

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

je možné mezi sebou spojovat a kombinovat. Je také možné vracet se do kteréhokoliv bodu v historii v rámci každé větvě.

Nastane-li konflikt při nahrávání změn, umožňuje Git jejich snadné vyřešení. Konflikt je stav, kdy například dva různí uživatelé provedli úpravy na stejném místě stejného souboru a snaží se je nahrát do repozitáře. Git v tuto chvíli druhého uživatele upozorní, že původní soubor byl změněn a je třeba tento konflikt vyřešit. Zamezuje se tedy přepsání změn prvního uživatele.

K systému Git existují různé služby, které tento systém rozšiřují o webové grafické rozhraní s množstvím dalších rozšíření. Nejčastěji používanými službami jsou GitHub[10], GitLab[11], nebo Bitbucket[12], z nichž některé lze provozovat na vlastním serveru. Snadným systémem pro vlastní provozování je také program Gitea[13], který je oproti předem zmíněným systémům zcela svobodným softwarem a je velmi jednoduchý na instalaci a správu. Tyto systémy mají navíc integrovaný jednoduchý WYSIWYG editor pro úpravu souborů přímo z webového rozhraní a také umí renderovat soubory s obsahem napsaným v jazyce Markdown, který je popsán v sekci 3.2.1.

5.6.1 Automatizace generování obsahu

Tato část práce se věnuje samotné implementaci automatického generování obsahu na základě změn v repozitáři.

Jak bylo zmíněno v sekci 5.1.1, git umožňuje nastavení takzvaných „hooks“, které se v určité chvíli spustí. Jak uvádí dokumentace[14], existuje spousta druhů hooků, které jsou vyvolány v různé části zpracování požadavku. V případě této implementace je nejvhodnější hook *post-receive*, který je spouštěn až po nahrání a zpracování všech změn v repozitáři.

Následující skript po vyvolání Gitem provede veškeré potřebné operace ke zpracování nového obsahu na web serveru.

[10] `https://github.com/`
[11] `https://gitlab.com/`
[12] `https://bitbucket.org/`
[13] `https://gitea.com/`
[14] `https://git-scm.com/docs/githooks`

Fig. 5.19: Skript pro automatizované generování obsahu

```
0  #!/bin/sh -e
1  GREPO="https://git.microlab.space/pedf/ucitelonline"
2  GDIR="ucitelonline"
3  WEBROOT="/srv/www/ucitelonline"
4
5  cd $(dirname $0)
6
7  [ ! -d "$GDIR" ] && git clone --recursive "$GREPO" "$GDIR"
8
9  cd "$GDIR"
10 git pull
11 zola build
12 rsync --recursive --delete --checksum \
13       --group --groupmap=*:www-data --chmod=D750,F640 \
14       public/ "$WEBROOT"
```

Skript 5.19 je složen z několika částí. Jako první probíhá na řádcích 1–3 nastavení proměnných, ve kterých se ukládá odkaz na vzdálený Git repozitář, název složky, do které se obsah má klonovat a název složky, do které se má kopírovat výstup, nebo-li vygenerované HTML. Dále se skript na řádku 5 přepíná do složky, ve které se sám nachází, proto aby skript fungoval vždy, ať je spuštěný ze kteréhokoliv místa v souborovém systému.

V další části skriptu probíhá na řádku 7 kontrola, zda již existuje složka s naklonovaným Git repozitářem. Pokud složka neexistuje, provede se naklonování vzdáleného repozitáře a tím i k vytvoření složky.

Třetí část provádí generování statického obsahu. Nejprve se skript přepne do repozitáře, v němž provede příkaz `git pull`, který do složky stáhne poslední změny ze vzdáleného repozitáře, tedy synchronizuje obsah na poslední verzi. Po synchronizaci repozitáře proběhne samotné spuštění generátoru, který z obsahu vygeneruje statické HTML, jenž vloží do složky `./public`. Poté na řádcích 12–14 probíhá kopírování nově vygenerovaného obsahu do složky `/srv/www/ucitelonline`, včetně nastavení Unixových práv

souborů na bezpečné hodnoty, které se liší pro složky a pro soubory.

Skript spoléhá na to, že systém má již předem správně nakonfigurované uživatele, uživatelské skupiny, web server, a že jsou nainstalované potřebné programy Git, Rsync, generátor Zola. Systémový uživatel, pod kterým je vyvolán Git hook, musí být ve skupině *www-data*, nebo v jiné skupině společně s uživatelem, pod kterým je spuštěn web server. Zároveň musí mít uživatel práva pro zápis do cílové složky `/srv/www/ucitelonline`.

Ve většině případů by bylo vhodné klonovat a generovat obsah v dočasné složce, například v `/tmp`, a po zkopírování souborů do složky web serveru opět zdrojové soubory smazat. To se ovšem v této implementaci nehodí, a to z důvodu, že repozitář se zdrojovými soubory může být velký a jeho klonování může potenciálně zabrat zbytečné množství času, na rozdíl o příkazu `git pull`, který pouze stáhne změny. Generátor zároveň při generování zpracuje pouze nutné změny, zatímco po čistém naklonování musí zpracovat celý obsah znovu, což může také trvat dlouho, obzvlášť při zpracování mnoha obrázků. V této implementaci se tedy zachováním naklonovaného repozitáře výrazně zkracuje čas celého skriptu.

6. Vyhodnocení modelové implementace

V této části práce je shrnuta a zhodnocena modelová implementace z kapitoly 5, a to jak implementace a využití samotného systému, tak jeho rozšíření implementovaných v sekci 5.6.1. Součástí této kapitoly jsou také návrhy pro další rozšíření systému.

6.1 Návrhy pro rozšíření systému

V praxi bylo zjištěno, že uživatelé, kteří neznají verzovací systém Git, mají problémy se jej naučit, obzvlášť v prostředí, které vyžaduje rychlé zpracování změn. Systém by bylo dobré rozšířit o jednoduchou webovou administraci, která umožňuje nezkušeným uživatelům jednoduchou práci s obsahem bez nutnosti hledání souborů ve stromové struktuře a znalosti jazyka Markdown. Částečně je tato funkcionalita poskytována systémem Gitea, který umožňuje jednodušší úpravy provádět přímo v prohlížeči, ovšem uživatel musí stále znát a pracovat s unikátnostmi jazyka Markdown a generátoru Zola.

Skript 5.19 pro automatické generování obsahu ze sekce 5.6.1 je možné rozšířit tak, aby byl schopen pracovat se vstupem z Git hooku, či se standardním vstupem *stdin*, který by umožňoval využití skriptu univerzálně pro různé webové stránky, nikoliv jen specificky pro tuto implementaci. Skript by také bylo možné rozšířit o jednoduché příkazy `echo`, které by oznamovaly stav, ve kterém se skript nachází. Standardní výstup skriptu vyvolaný přes Git hook je přesměrován uživateli, který spustil příkaz `git push` a tím i samotný hook. Skript by poté informoval uživatele o tom, zda právě stahuje změny na server, generuje statický obsah, či kopíruje soubory do kořenové složky web serveru. Z důvodu zachování jednoduchosti skriptu nebyly tato funkcionality implementovány.

6.2 Vyhodnocení implementace vlastních rozšíření

Do systému v modelové implementaci byla přidána vlastní rozšíření, tedy filtry pro vkládání souborů a videí do obsahu stránky, viz 5.4. Tyto filtry splňují původní požadavky, avšak jejich použití v obsahu se vymyká původnímu principu jazyka Markdown, tedy že obsah je čitelný i v čistém textu. Pro vyvolání filtrů je třeba vyplňovat různé jejich atributy, což se může zdát nepřehledné někomu, kdo si fungování filtrů neprostudoval. Zároveň je pak obsah nepřenositelný do jiných systémů, které neumí tyto filtry zpracovat a ve kterých by se kód pro vyvolání filtrů v takovém případě mohl interpretovat jako čistý text.

Závěr

Seznam použité literatury

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

Chacon, S. (2009). Why git is better than x. http://z.github.io/whygitisbetter/. Cit. 2020-03-26.

Chadburn, M. a Lahav, G. (2016). How slow websites damage publishers revenue. https://web.archive.org/web/20180929125709/http://engineroom.ft.com/2016/04/04/a-faster-ft-com/. Cit. 2020-02-15.

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.

Gouy, I. (2020). The computer language benchmarks game. https://benchmarksgame-team.pages.debian.net/benchmarksgame/which-programs-are-fastest.html.

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*.

Yahoo! (2020). Best practices for speeding up your web site. `https://developer.yahoo.com/performance/rules.html`. Cit. 2020-04-21.

Seznam ukázek zdrojového kódu