Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Bakalářská práce

CSS preprocesory

Jindřich Kytler

© 2016 ČZU v Praze

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Jindřich Kytler

Informatika

Název práce

CSS preprocesory

Název anglicky

CSS preprocessors

Cíle práce

Bakalářská práce je tematicky zaměřena na problematiku CSS preprocesorů. Hlavním cílem práce je komparace CSS preprocesorů a ověření jejich efektivity na vybraných reálných příkladech.
Dílčí cíle práce jsou:
- sumarizace historického vývoje CSS,
- charakteristika principu CSS preprocesorů a aplikačního SW pro tvorbu CSS stylopisu.

Metodika

Metodika řešené problematiky bakalářské práce je založena na studiu a analýze odborných informačních zdrojů. Vlastní práce spočívá v teoretickém rozboru a praktické ukázce realizace a využití CSS preprocesorů na vybraných příkladech. Na základě syntézy teoretických poznatků a výsledků praktické části budou formulovány závěry bakalářské práce.

Doporučený rozsah práce

30 – 40 stran textu.

Klíčová slova

CSS, preprocesor, LESS, SASS, Stylus

Doporučené zdroje informací

CASTRO, E. *HTML, XHTML a CSS : názorný průvodce tvorbou WWW stránek.* Brno: Computer Press, 2007. ISBN 978-80-251-1531-2.

CASTRO, E. – HYSLOP, B. *HTML5 a CSS3 : názorný průvodce tvorbou WWW stránek.* Brno: Computer Press, 2012. ISBN 978-80-251-3733-8.

ECCHER, C. *Profesionální webdesign : techniky a vzorová řešení pro XHTML a CSS.* Brno: Computer Press, 2010. ISBN 978-80-251-2677-6.

HOGAN, B P. *HTML5 a CSS3 : výukový kurz webového vývojáře.* Brno: Computer Press, 2011. ISBN 978-80-251-3576-1.

MEYER, E A. *Eric Meyer o CSS – ovládněte kaskádové styly!.* Brno: Zoner Press, 2004. ISBN 80-86815-03-.

STANÍČEK, P. *CSS : kaskádové styly.* Brno: Computer Press, 2003. ISBN 80-7226-872-4.

Předběžný termín obhajoby

2015/16 LS – PEF

Vedoucí práce

Ing. Pavel Šimek, Ph.D.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 28. 10. 2015

Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 10. 11. 2015

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 23. 12. 2015

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "CSS preprocesory" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 14.3.2016 _____

Poděkování

Rád bych touto cestou poděkoval panu Ing. Pavlu Šimkovi, Ph.D. za jeho cenné a užitečné rady. Dále bych chtěl věnovat velký dík mé rodině, která mě při psaní práce podporovala.

CSS preprocesory

Souhrn

Bakalářská práce se zabývá problematikou tvorby kaskádových stylů (způsob zobrazení elementů na webových stránkách) pomocí třech vybraných preprocesorů (SASS, LESS, Stylus) a jejich porovnání mezi sebou. Dále je také prováděna komparace při zápisu bez CSS preprocesoru.

Teoretická část se zabývá analýzou odborných zdrojů týkajících se problematiky CSS preprocesorů. Poté jsou v této sekci popsány výhody oproti zápisu bez preprocesorové technologie. Dále je rozebrán aplikační software, ve kterém je možná kompilace CSS preprocesorů.

V praktické části je vytvořeno a popsáno několik praktických příkladů pro ověření efektivity CSS preprocesorů.

Klíčová slova: CSS, preprocesor, SASS, LESS, Stylus

CSS preprocessors

Summary

Bachelor thesis deals with the creation of Cascading Style Sheets (way elements are displayed on the website) with three selected preprocessors (SASS, LESS, Stylus), and compare them with each other. It is also carried out a comparison of writing without CSS preprocessor.

The theoretical part deals with the analysis of expert sources concerning CSS preprocessor. Then, in this section is describe the advantages over the listing without preprocessor technology. Furthermore dismantled application software, in which it may compile CSS preprocessors.

In the practical part is created and described several practical examples to verify the effectiveness of CSS preprocessors.

Keywords: CSS, preprocessor, SASS, LESS, Stylus

Obsah

Seznam obrázků

Seznam tabulek

Seznam příkladů

1 Úvod

Informační technologie se posouvají kupředu nesmírnou rychlostí a u webdesignu tomu není jinak. Webové stránky, prezentace nebo informační weby jsou nedílnou součástí téměř většiny dnešní populace, proto poptávka po těchto produktech neustává.

Webová stránka je dokument napsaný ve značkovacím jazyce HTML, který tvoří kostru celého webu a určuje základní rozložení a organizace všech prvků v dokumentu. Bez HTML jazyka nelze vytvořit žádnou webovou prezentaci.

Druhou základní částí většiny prezentací jsou kaskádové styly, které učiní web pro návštěvníky uživatelsky přívětivým jak po stránce vzhledové, tak i po stránce přístupnosti. Stránky se tedy stanou s připojením stylů pro uživatele přehledné. Díky nim se na internetu nenachází jen pouhé textové elementy s nepřívětivým vzhledem. Oddělují obsah dokumentu od jeho vzhledu. Nebýt stylů, tak weby nejsou populární, jako je tomu dnes.

Velice často se stává, že se po nějaké době rozhodne o značné změně prezentace. Pokud nebyl psán kaskádový styl přehledně, dochází k obtížnému přepisování stylů, což obvykle vede vývojáře k psaní kódu úplně od začátku.

V tomto dokáží usnadnit a zpřehlednit vývoj právě CSS preprocesory, které pomáhají vývojářům k lepším pracovním postupům při psaní kaskádových stylů. CSS preprocesor poskytuje dynamické programování, které je více podobné běžnému programovacímu jazyku v porovnání s editací stylů bez preprocesoru. Principem je kompilace preprocesorového jazyka do běžného stylu. Jeho výhody jsou například snadná modifikace hodnot za pomoci proměnných, matematické operace nebo funkce s parametry.

Bakalářská práce je zaměřena na problematiku CSS preprocesorů a srovnání třech vybraných preprocesorů.

2 Cíl práce a metodika

2.1 Cíl práce

Bakalářská práce je tematicky zaměřena na problematiku tvorby kaskádových stylů pomocí CSS preprocesorů. Hlavním cílem je komparace CSS preprocesorů a ověření jejich efektivity na vybraných reálných příkladech.

2.1.1 Dílčí cíle

- Sumarizace historického vývoje CSS
- Charakteristika principu CSS preprocesorů a aplikačního SW pro tvorbu CSS stylopisu

2.2 Metodika

Metodika řešené problematiky bakalářské práce je založena na studiu a analýze odborných informačních zdrojů.

Vlastní práce spočívá v komparaci CSS preprocesorů. V mnoha případech se preprocesorový zápis komparuje s vygenerovaným kaskádovým stylem. Pro porovnávání se využívají technologie preprocesorů LESS, SASS a Stylus. Na základě syntézy teoretických poznatků a výsledků praktické části budou formulovány závěry bakalářské práce.

Práce se dělí do 2 základních kapitol:

- 3. Teoretická východiska
- 4. Vlastní práce

Kapitola *„Teoretická východiska“* se zaměřuje na stručnou historii vývoje jazyka HTML a CSS. Dále na představení preprocesorů obecně, jejich používání, instalaci či technické vlastnosti. Pro lepší představení jednotlivých vlastností je uveden vždy názorný příklad.

„Vlastní práce“ práce je zaměřena na tvorbu reálných příkladů, se kterými se dá běžně setkat v praxi. Na těchto příkladech se testuje efektivita CSS preprocesorů.

3 Teoretická východiska

3.1 World Wide Web

WWW je celosvětová síť mezi sebou propojených hypertextových dokumentů, které jsou k dispozici prostřednictvím internetové sítě. Chod WWW zajišťuje protokol HTTP.

Zakladatelem je Angličan Timothy Berners-Lee[1], který stojí rovněž za vývojem hypertextových dokumentů. Historie webové prezentace se počítá cca od roku 1990 [1].

3.2 Hyper Text Transfer Protokol

Protokol HTTP je nedílnou součástí pro komunikaci prostřednictvím WWW a obsahuje veškeré standardy pro správnou komunikaci mezi jednotlivými dokumenty.

Protokol pracuje na principu dotaz - odpověď. Uživatel zadává serveru dotaz prostřednictvím klienta na nějaký dokument, k čemuž využívá jeho URL adresu. Server následně dotazu buď vyhoví, nebo vrátí chybovou zprávu. Protokol určuje komunikaci mezi klientem a serverem [2].

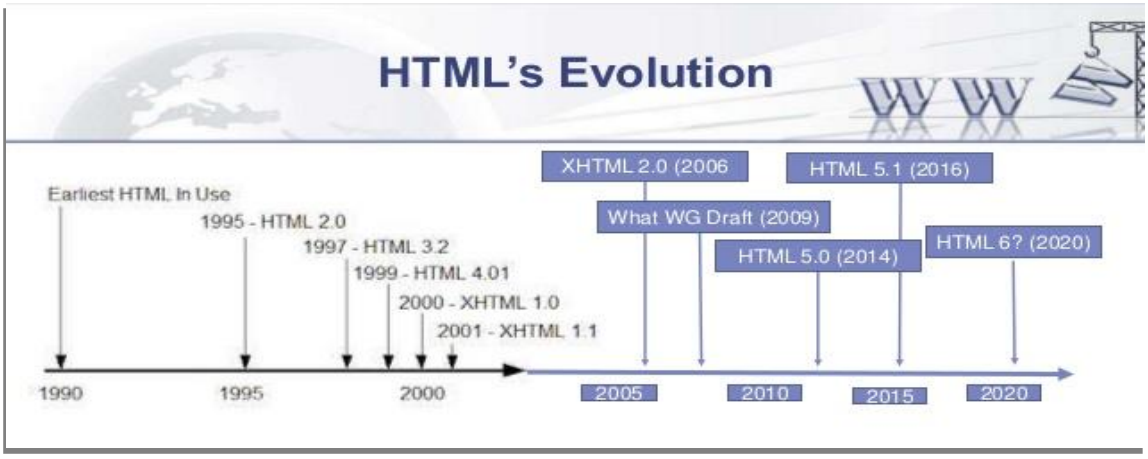
3.3 HTML

Základy HTML byli vynalezeny při spolupráci Tima Berners-Lee a Roberta Caillau v roce 1990 jako součást projektu WWW, který měl umožnit vědcům pracujícím v CERN zpřístupnění komunikace a sdílení vědeckých textů v počítačové síti. Umožňovala text rozčlenit do určitých logických úrovní, použít několik druhů zvýraznění textu a zařadit do textu odkazy a obrázky. Při navrhování autor nepředpokládal znalost daného jazyka. Prvotní verze byla napsaná přímo pro operační systém NextStep a obsahovala jak integrovaný editor, tak i prohlížeč [3].

V roce 1991 CERN spustil svůj web. Současně organizace NCSA svolala Marca Andreessena a Erica Binu k vytvoření prohlížeče Mosaic. Ten vznikl v roce 1993 ve verzích pro počítače IBM PC a Macintosh a měl obrovský ohlas. Byl to první webový prohlížeč s grafickým uživatelským rozhraním.

[1] TimothyBerners-Lee (* 8. června 1955, Londýn), zdroj www.w3.org

Následoval velmi rychlý vývoj webu, takže bylo nutné pro HTML definovat standardy.



Obrázek 1 – vývoj HTML (Zdroj: Vlad Alexander, 2013)

3.3.1 XHTML

Před příchodem HTML5 se HTML 4.01 nijak nevyvíjelo a přišel jazyk XHTML. XHTML je téměř totožný se značkovacím jazykem HTML a je založený na jazyku XML. Nic nového oproti HTML nepřináší. Žádné nové možnosti nebo specifikace, ale jen omezení. Pro správné zapisování XHTML jen nutné dodržovat pravidla [4].

```
Zápis v HTML:  <input type="text" placeholder="jmeno">
Zápis v XHMTL: <input type="text" placeholder="jmeno" />
```

Existují tři druhy XHTML:

- **XHTML 1.0 Strict** – Strukturovaný dokument, který neobsahuje žádné značky s formátováním vzhledu (např. b, i, u).
- **XHTML 1.0 Transitional** – Umožňuje požívat atributy pro formátování textu a odkazů v elementu body a některé další atributy.
- **XHTML 1.0 Frameset** – Určen pro rámce na webu pro rozdělení okna prohlížeče na dvě nebo více částí.

3.3.2 DHTML

Jako DHTML, čili Dynamické HTML, se označuje spojení HTML, JavaScriptu, kaskádových stylů a někdy DOM. Spojení zavedla společnost Microsoft ve svém prohlížeči

Internet Explorer 4. Nejde o žádný nový jazyk, ale o soubor nástrojů a postupů kombinující ostatní technologie. Předností DHTML je jeho schopnost efektně stránky rozpohybovat a zvýšit jejich dynamiku (odtud název DHTML). Obsah stránky je možné měnit i po jejím načtení (což u statického HTML nelze), změny jsou navíc vázány na konkrétní akce návštěvníka (např. kliknutí na určité místo, přejetí obrázku myší…). K tomu DHTML využívá rozhraní DOM, objektový model dokumentu [5].

3.3.3 HTML 5

Aktuální verze HTML5, která byla schválená v posledním čtvrtletí roku 2014, přináší podstatné změny v technologiích webových stránek proti předchozí verzi HTML4 z roku 1997, umožňuje kromě jiného přehrávat multimediální obsah přímo ve webovém prohlížeči a vytvářet v něm vlastní aplikace, které fungují i bez připojení k internetové síti. Přináší také nové HTML značky sémanticky definující strukturu stránky. [6]

3.4 CSS – kaskádové styly

Kaskádové styly (CSS) slouží k formátování HTML stránek. Lze jimi nastavit styl nadpisů, odstavců, odkazů, obrázků nebo i celých bloků (částí) stránky. Lze tedy jakékoliv elementy, které se nachází v dokumentu nastylovat.

Pomocí stylů lze například určit typ, velikost a barvu písma nadpisů, možno určit, zda budou tučné, či psané kurzívou, zarovnané vlevo či na střed apod. [7]

CSS umožňují oddělit vzhled a obsah stránky. Vzhled jednotlivých elementů je úsporně definován odděleně od HTML kódu. Jeden styl může být sdílen více stránkami [8]. Tím se dosáhne jednotného vzhledu. Lze pak také poměrně rychle změnit design, pokud byl styl zapisován systematicky.

3.4.1 Historie CSS

Rok 1996 dal vzniknout prvním webovým prohlížečům, které podporovali v určité míře grafické prvky. Tím došlo k myšlence tvorby galerie stylů, která by mohla být aplikována na HTML elementy. Zde se již dalo hovořit o snaze oddělení obsahové části webu od vzhledu.

Historie CSS je vlastně historií jejich podpory (a nepodpory) ve webových prohlížečích (protože ta se na rozdíl od standardů mění, jak přicházejí nové prohlížeče). Každý prohlížeč totiž styly interpretuje trochu jinak, některé prohlížeče neznají některé vlastnosti atd. [8]

Proto o rok později vychází první verze kaskádových stylů – CSS 1.0.

3.4.1.1 CSS 1

První z verzí kaskádových stylů měla majoritní podporu v prohlížeči Internet Explorer, který se ve své době objevoval ve verzi 3. Podpora CSS se v tomto směru omezovala na rodiny písma a obarvování elementů.

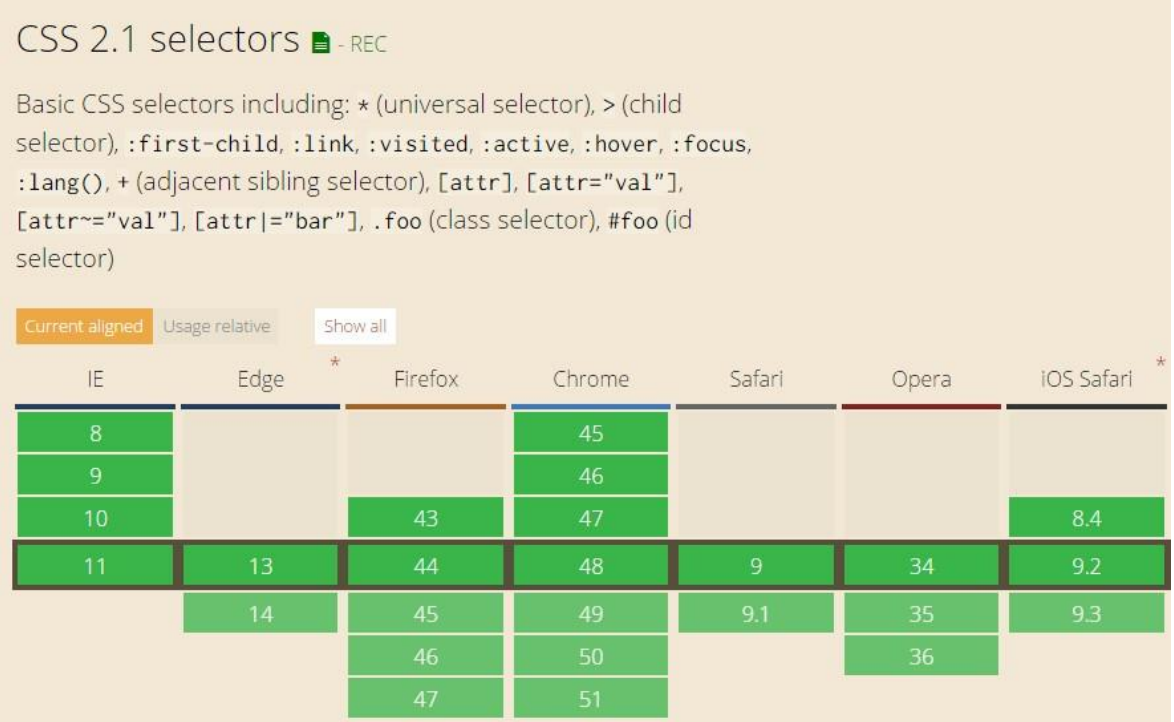
Větší podpory oproti předchozí verzi se dostává s příchodem následující verze IE, který se snaží podporovat zejména značnou část pozicování. Avšak přesto podpora nebyla nijak chvályhodná a tak se ji vývojáři často distancovali. Největším objevem se stala podpora pseudotřídy :hover, která umožňuje akci po najetí myší.

Prohlížeče 4 verzí neumožňovali speciálně formátovat tzv. pseudoelementy: první písmeno a první řádek elementu, ačkoliv CSS1 ono formátování ve specifikaci mělo. [9]

3.4.1.2 CSS 2

Druhá verze kaskádových stylů byla představena v roce 2000. S jejím příchodem však nastává velký problém s její podporou a jednotností v prohlížečích, což se v průběhu času změnilo.

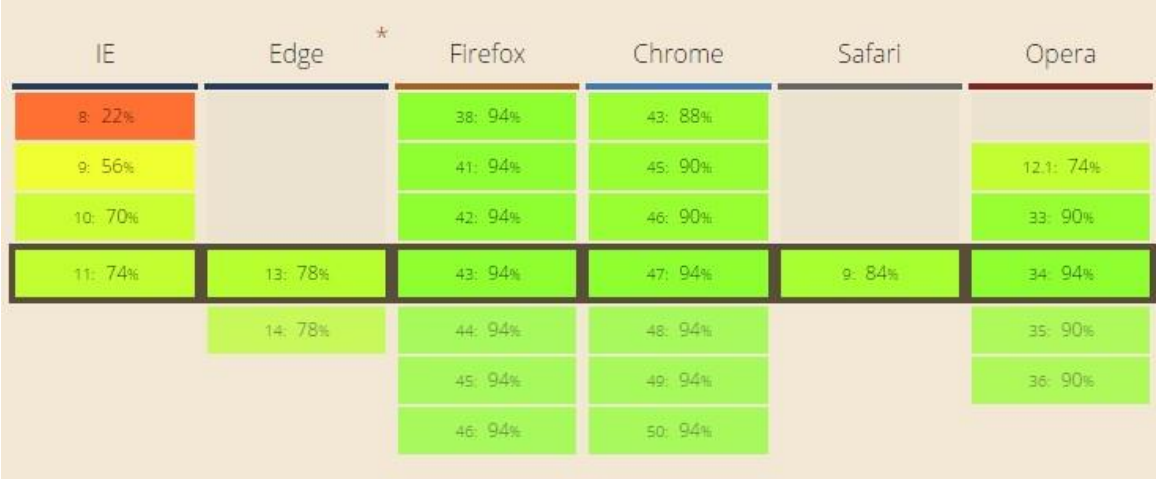
Cílem kaskádových stylů druhé generace je snaha stát se hlavním formátovacím jazykem napříč různými platformami. V dnešní době je jeho aktualizace (verze 2.1) nejrozšířenější a nejvíce podporovanou verzí jazyka vůbec. [9] Byla vytvořena, aby odstranila problémy, které vznikaly v prohlížečích s verzí 2.0.



Obrázek 2 - Podpora CSS 2.1 (Zdroj: caniuse.com, 2015)

3.4.1.3 CSS 3

S příchodem HTML5 vyšla i nová verze stylů. Nynější verze jazyka CSS, která se vyvíjí již od roku 2005. Jeho podpora již dnes není problémem téměř pro žádný z prohlížečů. Jednou z výjimek je například IE8, kde se využívají jiných metod, aby se dosáhlo jednotného nebo téměř identického vzhledu. Mezi jeho vlastnosti patří např. animace, stín u blokového prvku, zaoblené rohy, stín u textu, transformace objektů a další.



Obrázek 3 - Podpora CSS 3 (Zdroj: caniuse, 2015)

3.4.2 **Připojení CSS stylů k HTML**

3.4.2.1 Přímý inline zápis stylu pomocí atributu style

Do určitého prvku, na který se má aplikovat určité CSS vlastnosti, se vloží atribut style, vlastnost a její hodnota. Nevýhodou je ztráta „káskádovitosti“. Pro aplikaci se musí uvést u každého elementu a špatně se pak v budoucnu upravuje. V praxi by se měla používat většinou jen v nejnutnějších případech.

Příklad: obarvení elementu modrou barvou

```
<div style="color:aqua"></div>
```

3.4.2.2 Zápis pomocí stylopisu

Stylopis je seznam stylů pro daný dokument zapsaný mezi tag <style> v hlavičce dokumentu. Vhodné pro použití tam, kde je třeba použít určitý styl jen pro jeden dokument. Dnes už se nemusí uvádět atribut type, pokud nebude použitá starší HTML syntaxe.

Příklad: všechny nadpisy úrovně h1 budou obarveny stejnou barvou

```
<style>h1{ color: #eaeaea; }</style>
```

3.4.2.3 Připojením externího souboru

Samotné připojení je řešeno pomocí odkazu, opět v hlavičce stránky. Tento způsob je nejvíce rozšířený a doporučovaný. Změna jednoho CSS souboru se poté projeví na všech stránkách.

Příklad:

```
<link rel="stylesheet" type="text/css" href="./app.css"/>
```

nebo nově v HTML5 se nemusí uvádět atribut type

```
<link rel="stylesheet" href="./app.css"/>
```

3.4.3 **Slabiny a nedostatky CSS**

Při rozsáhlejší webových projektech jako jsou například interní webové aplikace, dochází k nepřehlednosti kódu a velice obtížné správě. Dělením na menší css soubory je sice dosaženo určité logiky, ale soubory se musí jednotlivě nalinkovat, a to znamená s každým novým css souborem bude vznikat další http dotaz.

S příchodem CSS3 sice došlo k mnoha novým možnostem tvorby vzhledu, které v předešlé verzi nebyly možné a musely se řešit složitými javascripty, ale docházelo k nekonzistentnostem mezi jednotlivými prohlížeči. Prohlížeče tedy zavedly prefixy, které však při psaní bez preprocesoru značně zpomalují psaní CSS.

Příklad: ukázka použití CSS3 vlastnosti transform pro rotaci objektu

```
.box {
    /* Chrome, Safari 3.1+: */
    -webkit-transform: rotate(7.5deg);
    /* Firefox 3.5-15: */
    -moz-transform: rotate(7.5deg);
    /* IE 9: */
    -ms-transform: rotate(7.5deg);
    /* Opera 10.50-12.00: */
    -o-transform: rotate(7.5deg);
    /* Firefox 16+, IE 10+, Opera 12.10+: */
    transform: rotate(7.5deg);
}
```

Velice často se také stává, že nedochází k takzvanému DRY (don´t repeat yourself), což znamená, aby se určité bloky kódu neopakovaly, když jsou stejné pro více elementů.

CSS neobsahuje skoro žádné základní funkce, které jsou v běžných programovacích jazycích základem. Jako například proměnné či základní matematické operace.

3.4.4 BEM

BEM (Block, Element, Modifier) je postup, kterým zapisovat CSS pravidla a pojmenovávat CSS třídy.

```
.blok {}
.blok__element {}
.blok--modifikator {}
```

Hlavní princip BEM boří jeden z hlavních pilířů kaskádových stylů – kaskádovitost. Vůbec se totiž nevyužívá zanořování. Každý element má svou třídu, podle které je jednoznačně identifikován. Postup Block, Element, Modifier může být užitečný jako společná konvence při práci hodně lidí nad stejném CSS, kdy je jasně patrné, co která třída znamená. Jde použít i nějaký kompromis, kdy se alespoň v rámci bloku používají třídy s názvem bloku na začátku [10].

CSS:

```
.produkt{ }
.produkt__popis{ }
.produkt--slevneny{ }
```

HTML:

```
<div class="produkt">
    <div class="produkt__popis">Popis produktu</div>
</div>
<div class="produkt produkt--slevneny">
    <div class="produkt__popis">Popis produktu, který je ve slevě a tak
bude výrazněji, pomocí CSS, nastylovaný modifikátorem ".produkt--slevneny"
    </div>
</div>
```

3.5 CSS preprocesory

CSS preprocesor je nástroj, který ze zdrojového kódu zapsaného ve vlastní syntaxi vygeneruje výsledný validní CSS pro prohlížeč [11]. Ve své podstatě se jedná o jazyk postavený nad CSS [12]. Preprocesor je určen pro vývojáře webových stránek. Umožňuje zjednodušit a zrychlit psaní CSS kódu, učinit jej přehledným a snadněji modifikovatelným. Pokud na webovém projektu spolupracuje více vývojářů, může kód stylů narůst obvykle i přes tisíce

řádků, ve kterých se velmi těžko orientuje a provádí jakékoliv zásadní změny. Pomocí technologie preprocesorů se zvyšuje přehlednost a udržovatelnost kódu, což je při vývoji velmi důležité. Při psaní kaskádových stylů bez preprocesoru nemůžeme využívat vlastnosti, jako jsou proměnné, vnořování kódu, mixiny (znovu použitelné bloky kódu), podmínky, cykly a další. Díky CSS preprocesorům jsou všechny tyto prvky dostupné. Každý preprocesor má vlastní syntaxi, vlastní kompilaci. Preprocesorů byla dříve celá řada [13]. Dnes se však okruh vývojářů přesunul na tři nejvíce rozšířené preprocesory a těmi jsou LESS[2], SASS[3] a Stylus[4].

Preprocesory sice ve skutečnosti neobohacují CSS o žádné nové vlastnosti, ale přináší efektivní vývojový pracovní postup, který dokáže při správném používání ušetřit plno času a tím pádem i financí při tvorbě projektů.

Plno z dříve dostupných preprocesorů se již na internetu nenachází, ty ještě dostupné jsou většinou dlouho neaktualizované a obsahují pouze základní funkce.

Název preprocesoru	Jazyk	Stabilní verze	Komunita	Poslední aktualizace	Přispěvatelé
LESS	JavaScript	2.6.0	13 375	29.1.2016	193
SASS	Ruby/C,C++	3.4.21	7 600	12.1.2016	175
Stylus	JavaScript	0.53.0	6 751	14.12.2015	146

Tabulka 1 - Přehled vybraných sledovaných parametrů (Zdroj: vlastní, leden 2016)

Jazyk – ve kterém je preprocesor napsán

Stabilní verze – verze, která je vyzkoušená a odladěná do použitelné míry (občasné drobné chyby, které se opravují téměř ihned)

Komunita – počet vývojářů, kteří se zajímají o danou technologii na serveru github

Poslední aktualizace – představuje datum vypuštění poslední stabilní verze.

Přispěvatelé – počet vývojářů upravující kód na serveru github

[2] http://lesscss.org/

[3] http://sass-lang.com/

[4] http://stylus-lang.com/

3.5.1 **LESS**

LESS neboli Leaner CSS, je dynamický jazyk pro tvorbu stylesheetů [15], autorem je Alexis Sellier, v intenetové komunitě vystupuje jako cloudhead. Jedná se o open-source software. Všechny zdrojové kódy jsou dostupné v repositářích na serveru Github.com. Prvotní verze byla napsána v jazyce Ruby, v následujících verzích byl jazyk Ruby nahrazen jazykem Javascript. LESS běží jako jediný z vybraných preprocesorů jak na klientské straně (Chrome, Safari, Firefox), tak na straně serveru, s Node.js a Rhino [16]. LESS funguje uvnitř javascriptové knihovny s názvem Node. Existuje ale i mnoho nástrojů třetích stran, které umožňují kompilaci a změny souborů mimo prostředí Node. Tento preprocesor spadá, i když se jedná technicky o imperativní jazyk, se svými funkcemi do kategorie deklarativního jazyka [17].

3.5.2 **SASS**

Jazyk vymyslel Hampton Caitlin. O vývoj se stará Natalie Weizenbaum a Chris Eppstein. Avšak dnes se na vývoji podílí plno další developerů, jinak by jazyk nebyl tak rychle vyvíjen

	RUBY SASS 3.2	RUBY SASS 3.3	RUBY SASS 3.4	LIBSASS 3.1	LIBSASS 3.2	LIBSASS 3.3
PASSED	8	54	68	33	67	68
FAILED	60	14	0	35	1	0
PERCENTAGE	11.76%	79.41%	100.0%	48.53%	98.53%	100.0%

Obrázek 4 - Test funkčnosti SASS jader (Zdroj: sass-compatibility.github.io, 2015)

a nestačil by konkurovat ostatním preprocesorům. První commit Sassu se datuje na konec roku 2006, tedy skoro před 10 lety. Netřeba snad ani dodávat, že od té doby uplynula dlouhá doba. SASS je psán ve dvou odlišných jazycích. Nejúspěšnější z nich, LibSass (napsaný v C/C++) je nyní blízko k plné kompatibilitě s původní Ruby verzí. V roce 2014 se Ruby Sass a LibSass týmy se rozhodly počkat a sjednotit obě verze před dalším postupem. Od té doby LibSass aktivně uvolňuje verze kompatibilní s jeho starším bratrem. Zpět k volbě kompilátoru při vytváření nového projektu. Pokud je založen na Ruby on Rails, je lepší použít Ruby Sass, který se na takovýto případ perfektně hodí. Také je nutné brát v potaz, že Ruby Sass bude vždy referenční implementací a bude vždy udávat směr funkcí LibSassu. Na projektech, které Ruby nevyužívají, bude použití LibSassu pravděpodobně lepší nápad [18].

3.5.3 Stylus

Stylus je nejmladším z vybraných preprocesorů. Poučil se z chyb LESS a SASS preprocesorů. Autorem Stylu je TJ Holowaychuk. Na internet vyšla první verze v roce 2010. Zdrojový kód je napsaný v JavaScriptu jako LESS. Stylus se řadí do kategorie open-source a je šířen pod licencí MIT.

3.5.4 Jak používat preprocesory

3.5.4.1 Příkazová řádka

LESS

Pro instalaci je nutné nainstalovat node.js, který lze stáhnout z oficiálních stránek nodejs.org. Instalace pak probíhá velmi snadno, pomocí jednoho řádku a pouze jednou.

```
$ npm install -g less
```

Jakmile je LESS nainstalovaný, lze ihned kompilovat vytvořené less soubory. Kompilace se provede znovu jedním krátkým příkazem.

```
$ lessc app.less
```

Tento příkaz zkompiluje less vstup do univerzálního souboru stdout. Je tedy nezbytné na konec příkazu přidat ještě název výstupu, do kterého se bude less kompilovat.

```
$ lessc app.less app.css
```

Často se využívá také tzv. minimalizování kódu, to je odstranění mezer, řádků, komentářů a další věci, které ušetří i několik desítek kilobajtů a u větší projektů dokonce stovky kilobajtů. Tato minimalizace vede k menší velikosti daného souboru a rychlejšímu načítání webové stránky. K tomu je třeba nainstalovat plugin clean-css opět pomocí npm. V konzoli se přidá přepínač --clean-css, aby se vstup minifikoval na výstupu. Lze také kompilaci provádět pomocí přepínače –x, který je defaultně součástí lessc příkazu, ale není tak efektivní jako zmíněný plugin.

```
$ lessc --clean-css app.less app.min.css
```

SASS

Uživatelé distribuce linuxu potřebují nainstalovat nejprve Ruby. Lze Ruby nainstalovat pomocí balíčkovacího nástroje, rbeny nebo rvm.

```
$ sudo su –c gem install sass
```

Předtím než se začne na OS Windows používat Sass, je třeba nainstalovat Ruby. Pouhým jedním kliknutím na instalátor se vše rychle nainstaluje a nastaví. Instalátor ruby také umožní pracovat s Ruby knihovnami. Sass je sice závislý na Ruby, ale na Mac zařízeních je již Ruby předinstalované a není jej třeba instalovat. Ruby používá pro instalaci balíčků tzv. gemy. Takže instalace SASS vypadá následovně:

```
$ gem install sass
```

Pokud vyskakuje chybová hláška, většinou nemá uživatel přístupová práva a je třeba použít příkaz ve tvaru:

```
$ sudo gem install sass
```

Pro ověření, že SASS byl nainstalován správně, se dá ujistit pomocí příkazu:

```
$ sass -v
```

Konzole by měla vrátit `„Sass 3.4.21 (Selective Steve)“`

Následně je možné začít kompilovat SASS soubory do CSS pomocí příkazu:

```
$ sass --watch app/sass:public/stylesheets
```

Takto napsaný příkaz bude sledovat změny ve složce app/sass, takže jakmile se upraví nějaký soubor v této složce, dojde ke kompilace. Výstup se zkompiluje do složky public/stylesheets.

Sledování zajišťuje přepínač –watch. Bez tohoto přepínače by bylo nutné pro každou viditelnou změnu příkaz zadávat znovu a znovu.

Přidáním přepínače –style=“compress“ výstup bude zkomprimovaný, který bude menší ale nečitelný pro zkoumání kódu.

Stylus

Instalace Stylus je velice snadná. Stačí nainstalovat balíčkovací nástroj Node.js jako tomu je u preprocesoru LESS.

Do konzole se zadá:

```
$ npm install stylus -g
```

Pro kompilaci:

```
$ stylus -w style.styl -o style.css
```

3.5.4.2 Použití na straně prohlížeče

LESS je jediným z vybraných preprocesorů, který umí překládat stylopis přímo na straně klienta pomocí JavaScriptu. Proto je nutné nalinkovat knihovnu less.js a stylesheet less do webové prezentace.

Připojený stylesheet .less s atributem rel nastaveným na "stylesheet/less"

```
<link rel="stylesheet/less" type="text/css" href="app.less">
```

less.js vložený do hlavičky <head> na stránce

```
<script src="less.js" type="text/javascript"></script>
```

Stylesheety je nutné nalinkovat před skriptem. Tento způsob není však doporučený používat na produkčním serveru, jelikož může docházet k problémům při špatné manipulaci se skriptem.

3.5.4.3 Použití aplikací třetích stran

Při volbě aplikace třetích stran je možné upustit od jakékoliv práce s příkazovým řádkem, sledovacím režimem v prohlížeči apod. Pouze stačí stáhnout a nainstalovat aplikaci, která se o všechny tyto problémy postará sama. Lze si aplikace rozdělit na tři základní skupiny, a to na

LOCAL GUI COMPILERS (LESS, SASS, OR STYLUS CODE TO CSS)			
	Less	**Sass**	**Stylus**
Windows, Mac & Linux	Prepros (trial or $29)		
	Crunch 2 ($19.95)		
	Crunch 2 (free)		
	Koala (free)		
	Netbeans (free)		
		Compass.app (free)	
Windows & Mac		Scout (free)	
Windows	LiveReload (free)		
	WinLESS (free)		
Mac	Codekit (trial or $32)		
	LiveReload ($10)		
		Hammer (trial or $20)	
		Sassquatch (trial or $4)	

Obrázek 5 - Přehled GUI kompilátorů (Zdroj: csspre.com/compile/)

GUI aplikace, online kompilátory a IDE, jinak řečeno editory kódu. Lze také použít online kompilátory, které se ani nemusí instalovat a dosáhne se stejného výsledku. Avšak online kompilátory nejsou pro praktické každodenní používání vhodné. Slouží spíše k otestování či vyzkoušení preprocesorů, pokud se s nimi začíná.

ONLINE COMPILERS (LESS, SASS, OR STYLUS CODE TO CSS)			
Less	Sass		Stylus
	SCSS syntax	Sass syntax	
CodePen.io			
JSbin			
CSS Deck			
Fiddle Salad	Fiddle Salad	Fiddle Salad	Fiddle Salad
LessTester	SassMeister		Try Stylus
Online Less Compiler	Rendera		StylCompile
less2css.org	JSFiddle		
dopefly LESS Converter			
LessCSSisMore			
Leafo.net (lessphp)			
Less.php 2 CSS			
BeautifyTools	BeautifyTools	BeautifyTools	BeautifyTools

Obrázek 6 - Přehled online kompilárorů (Zdroj: csspre.com/compile/)

Pokud je k dispozici pokročilý IDE editor např. PHPStorm, WebStorm, Sublime Text, Atom, Eclipse, VisualStudio, NetBeans a další, je vhodné, využít nástroje pro ně připravené, které se o kompilaci postarají. Pokud se syntaxe nezvýrazňuje, tak je třeba nainstalovat plugin, který zajistí správné zobrazování a samotnou kompilaci.

3.5.5 Přípony preprocesorových souborů

Pro přechod k zapisování stylů pomocí některého z preprocesorů stačí přejmenovat dosavadní CSS soubor, tak aby měl příponu daného preprocesoru.

Název	Přípona
LESS	*.less
SASS	*.sass / *.scss
Stylus	*.styl

Tabulka 2 - Přípony preprocesorových souborů

Výstup všech preprocesorů je samozřejmě s příponou *.css.

3.5.6 Syntaxe

Všechny komparované preprocesory mají téměř stejnou syntaxi. Liší se většinou jen v interpunkci a některých znacích. Pokud se v editoru syntaxe nezvýrazňuje správně, tak je potřeba nainstalovat plugin nebo rozšíření, které syntaxi začne zvýrazňovat.

LESS

Má z preprocesorů syntaxi nejvíce podobnou běžnému zápisu CSS. Tudíž přechod na něj je velice prostý [12].

```
.box{
    vlastnost: hodnota;
}
```

Příklad 1 – LESS syntaxe

SASS

SASS poskytuje na výběr ze dvou možných syntaxí. Více se postupem času však uchytila syntaxe nazývaná Sassy CSS neboli zkráceně SCSS. Je více podobná CSS zápisu, a proto se stala mezi vývojáři oblíbenou. Druhá syntaxe nese označení SASS. SASS syntaxe vypouští složené závorky {}, středníky a je citlivá na tabulátory. SCSS ponechává všechny znaky. Označuje se také jako whitespace syntaxe.

SASS syntaxe:

```
.box
    vlastnost: hodnota
```

SCSS syntaxe

```
.box{
    vlastnost: hodnota;
}
```

Příklad 2 – SASS a SCSS syntaxe

Dále v této práci bude používaná SCSS syntaxe.

Stylus

Syntaxe Stylus je z vybraných preprocesorů nejvíce rozmanitá v zápisu deklarací. Lze psát kód jako při psaní bez CSS preprocesoru nebo upustit od složených závorek, středníků či dvojteček.

Příklad syntaxe Stylus:

```
.box{
  vlastnost: hodnota;
}
```

```
.box
  vlastnost: hodnota;
```

```
.box
  vlastnost hodnota
```

Příklad 3 – Stylus syntaxe

Každá z těchto syntaxí je pro Stylus přijatelná a bude zkompilována. Avšak neměla by se syntaxe kombinovat, že v jednom souboru se bude psát jedním způsobem a v jiném odlišně než v předešlém. Když už se zvolí preprocesor Stylus je doporučena syntaxe ve třetím sloupci. Po kompilaci všech tří zmíněných způsobů by výstup vypadal jako v sloupci prvním.

3.5.7 **Proměnné**

Obecně proměnné v informatice slouží k uložení nějaké hodnoty, která se definuje a následně se s ní pracuje. Přičemž hodnota může být statická nebo se dynamicky měnit. CSS proměnné nenabízí, a tak hlavní výhodou preprocesorů jsou právě proměnné, které slouží například na definování primární barvy webu, hodnoty se kterou se bude později počítat nějaká šířka sloupce nebo pro definování textového řetězce pro název fontu, který bude určen pro nadpisy apod.

Rozšířeným trendem se v posledních letech stal zápis proměnných do jednoho souboru. Tím pádem jsou všechny hodnoty pohromadě a při pozdějších úpravách stačí editovat jen tento soubor. Při vytváření proměnných je nutné dbát na správný název. Správným názvem je myšleno, aby název dával smysl hned na první pohled. Jelikož je možné, že na projektu bude v budoucnu pracovat úplně jiný člověk nebo i více lidí. Tudíž je dobré si zvolit i nějakou konvenci pro zapisování.

Dále se také používá zápis proměnných přímo v daném místě zápisu. Například existuje komponenta menu, která bude mít velikost písma definovanou v proměnné menu-color. Vše bude v odděleném souboru s názvem menu. O této problematice hovoří kapitola import a parciální soubory níže.

LESS pro proměnné používá znak @. SASS pro ně má deklarovaný znak $. Stylus před proměnný může mít jak @, tak i $, ale tyto znaky jsou pouze textovým řetězcem. Pro určení používá místo dvojtečky znak =.

Příklad: vytvoření proměnné „primarni-barva“, „sirka-stranky“ a následné použití.

LESS:	SASS:	Stylus:

`@primarni-barva: #000;`
`@sirka-stranky:1280px;`
 	
`.hlavni-blok{`
`color: @primarni-barva;`
`width: @sirka-`
	`$primarni-barva: #000;`
	`$sirka-stranky:1280px;`
	
	`.hlavni-blok{`
	`color: $primarni-barva;`
	`width: $sirka-`
		`primarni-barva = #000`
		`sirka-stranky = 1280px`
		
		`.hlavni-blok`
		`color primarni-barva`
		`width sirka-stranky`

stranky; }	stranky; }	

Příklad 2 – Proměnné

Výsledné CSS:

```
.hlavni-blok {
  color: #000;
  width: 1280px;
}
```

Nebo lze například do proměnné uložit v SASS deklaraci pro responzivitu.

```
$medium-up: "only screen and (min-width: 50em)";
@media #{$medium-up} {
  // Kód pro zařízení od šířky 50em
}
```

3.5.8 Zanořování

Nesting v překladu znamená hnízdění, ale je více znám pod pojmem zanořování. Zanořováním dochází ke stylování, tak jak je vytvořena struktura dokumentu v DOM. Bez preprocesoru se musí neustále kopírovat posloupnost selektorů předešlých, aby docházelo ke stylování pomocí zanořování. V preprocesorech se již nic kopírovat nemusí. Stačí do rodičovského selektoru vložit selektor, který se nachází uvnitř a takto lze zanořovat dále a dále. Zde je třeba důrazně upozornit na hrozící nebezpečí, protože s narůstajícím stupněm zanoření stoupá i složitost budoucích úprav a celkové manipulace se stylesheety [19]. Byla tedy stanovena nepsaná konvence, která říká, že by se mělo jít maximálně na 3-4 zanoření. Obecně platí, že je nutné vždy zamýšlet se nad tím, jaký web se vlastně bude vytvářet a stanovit normu pro zanořování.

LESS, SASS:	Stylus:	Výstup CSS:

nav {
  width: 50%;
  li {
  background: #000;
  a {
  color: #777;
  }
  }
}
	nav
	  width 50%
	  li
	  background #000
	  a
	  color #777
		nav {
		  width: 50%;
		}
		nav li {
		  background: #000;
		}
		nav li a {
		  color: #777;
		}

Příklad 3 – zanořování jendoduché navigace

Zde je jasně patrné, že pokud se zápis provede bez preprocesoru, musí se např. v poslední úrovni vypisovat část „nav ul li“ a až pak přidat selektor „a“ nebo kopírovat předešlou část zanořeného selektoru.

Nedílnou součástí při tvorbě kaskádových stylů jsou téměř pokaždé pseudotřídy nebo elementy. V preprocesorech se zapisují tímto způsobem:

LESS, SASS:

```
a{
  color: #777;
  &:hover{
    color: #fff
  }
}
```

Výstup v CSS:

```
a {
  color: #777;
}
a:hover {
  color: #fff;
}
```

Příklad 4 - zanořování pseudo třídy odkazu

Dokonce lze i vytvořit mnohonásobný selektor, pro který budou platit vlastnosti prvku právě tehdy, bude-li existovat stejný selektor bezprostředně za ním. Selektor je známý již z CSS2.

LESS, SASS:

```
.prvek{
  vlastnost: hodnota;
  & + &{
    vlastnost2: hodnota2;
  }
}
```

Výstup v CSS:

```
.prvek + .prvek {
  vlastnost2: hodnota2;
}
```

Příklad 5 - Zanořování stejného selektoru

3.5.9 Mixiny

Lze je chápat jako bloky definic, které se dají použít na více různých místech. Přičemž můžou mít i parametry, kde se hovoří o tzv. parametrických mixinech. Smyslem těchto funkcí je budoucí úprava kódu, která probíhá na jednom místě, tudíž není potřeba měnit stejné definice u každého selektoru zvlášť a zamezit zbytečnému kopírování stejného kódu.

SASS používá pro definici mixinu tvar @mixin <název_mixinu>($parametr: defaultní hodnota parametru). Příklad použití parametrického mixinu, pro vypisování variant kódu kvůli různé podpoře CSS3 vlastností. V tomto případě se jedná o animaci jakékoliv vlastnosti tlačítka.

Příklad použití mixinu pro zrušení tzv. „obtékání“ při tvorbě webu:

LESS:

```
.clearfix() {
  &:before,
  &:after {
    content: " ";
    display: table;
  }
  &:after {
    clear: both;
  }
```

SASS:

```
@mixin clearfix() {
&:before,
&:after {
  content: " ";
  display: table;
}
&:after {
  clear: both;
}
```

Stylus:

```
clearfix()
  &:before,
  &:after
    content: " ";
    display: table;
  &:after
    clear: both;
```

```
}

footer {
  .clearfix;
}
```

```
}

footer {
  @include clearfix;
}
```

```
footer
  clearfix()
```

Příklad 6 – Mixin pro zrušení obtékání textu

Výstup CSS:

```
footer:before,
footer:after {
  content: " ";
  display: table;
}
footer:after {
  clear: both;
}
```

Význam obtékání při tvorbě webu je vysvětlen v praktické části při tvorbě grid systému.

3.5.10 Extend

Extend na rozdíl od mixinů nekopíruje deklarace, ale vytváří kombinované selektory. Jako u mnoha vlastností preprocesorů platí, že při špatném používání se může výhoda extend stát při ladění stylů nevýhodou. Tudíž je třeba dbát na místo, kde použít extend a kde ho nepoužít. LESS v dřívějších verzích extend nepodporoval, ale dnes je tomu již jinak [20].

LESS:

```
.trida{
  vlastnost1: 1;
}
.trida-1{
  &:extend(.trida);
  vlastnost2: 2;
}
.trida-2{
  &:extend(.trida);
  vlastnost3: 3;
}
```

SASS, Stylus:

```
.trida {
  vlastnost1: 1;
}
.trida-1{
  @extend .trida;
  vlastnost2: 2;
}
.trida-2{
  @extend .trida;
  vlastnost3: 3;
}
```

Výstup v CSS:

```
.trida,
.trida-1,
.trida-2 {
  vlastnost1: 1;
}
. trida-1 {
  vlastnost2: 2;
}
.trida-2 {
  vlastnost3: 3;
}
```

Příklad 7 - extend

3.5.11 Import a parciální soubory

Při stoupajícím množství napsaného kódu, který se píše pouze do jednoho souboru, může docházet ke špatné orientaci a celkové úpravě kódu. Proto se začaly dělit kódy do jednotlivých souborů. Dodržovat se ovšem musí určitá logická struktura. Problém opět snadno řeší preprocesory importem ostatních souborů do jednoho. Hlavním rozdílem mezi importem v preprocesoru a bez něj je, že při preprocesorovém nedochází http requestům navíc. Při

importu v CSS s každým novým souborem se sníží rychlost načítání webu. Nejvíce tento fakt můžou pocítit uživatelé mobilních zařízení.

```
index.less
COMPONENTS/
|--gallery.less
|--pagination.less
CORE/
|--variable.less
|--mixins.less
|--helpers.less
BASE/
|--text.less
|--fonts.less
|--table.less
```

Příklad jednoduché adresářové struktury.

Index.less – do kořenové složky se umístí soubor, který má být kompilován.

Components – označuje komponenty, ze kterých se skládá web. Lze je pak snadno použít i na jiném místě než na místě, kde byly prvně vytvořené. Bez použití komponent dochází často k opakování zbytečného kódu.

Core – nese označení pro imperativnější část adresářové struktury. Umisťuje se do ní většinou proměnné, mixiny a globální pomocné třídy.

Base – jde o složku, do níž vkládají obecné části jako styl pro texty, tabulky fonty či formuláře.

Zápis LESS a SASS	Zápis Stylus
`@import "header";`	`@import "header"`

Příklad 8 - import a parciální soubory

Import probíhá ve všech preprocesorech téměř stejným způsobem. Ve Stylus se mohou vynechat středníky. Také je nutné podotknout, že není nezbytné psát u názvů importovaných souborů jejich přípony. Preprocesor si je rozezná sám. Editory dokonce i našeptávají názvy souborů, které jdou v dané struktuře importovat. V SASS lze dávat před názvy souborů podtržítka a tím pádem bude kompilátor vědět, že daný soubor není výstupní, ale pouze slouží pro import. Pokud se importuje běžný CSS soubor, preprocesory zachovají ve zkompilovaném kódu direktivu @import. LESS například umožňuje importovat CSS soubory pomocí vlastnosti LESS takto: @import (less) "fancybox.css";

3.5.12 **Matematické operátory**

Počítání jakékoliv hodnoty, která třeba i závisí na jiné, v CSS není možné. Preprocesory již absenci matematických operací ruší. Základní operátory jsou +, -, *, /. Využívá se jich při práci s proměnnými, barvami a hodnotami.

Příklad: selektor B má být vždy dvakrát vyšší než selektor A. Při změně výšky prvku A dojde automaticky.

Zápis Stylus:	Výstup CSS:

`height = 200px;`
 	
`.element-A`
`  height height`
 	
`.element-B`
`  height height * 2`
	`.element-A {`
	`  height: 200px;`
	`}`
	`.element-B {`
	`  height: 400px;`
	`}`

Příklad 9 - operátory

Nejdříve se vytvoří proměnná height, do které se uloží hodnota, a poté se přidělí třídě element-A. U třídy element-B se pak jednoduše pomocí operátoru násobení dosáhne dvojnásobné velikosti. Pokaždé, když se změní proměnná height, bude výška třídy element-B dvakrát větší než třída element-A.

Matematických funkcí je v preprocesorech plná řada a je zbytečné uvádět všechny. Většinou se ale využívá v praxi jen část z nich. Kompletní přehled se nachází v dokumentacích na oficiálních webech jednotlivých preprocesorů.

3.5.13 **Barevné funkce**

Při tvorbě webů se používají zhruba 2-3 základní barvy. Nemělo by se přehánět s velkými barevnými kontrasty, aby web zůstal čitelný a přístupný. Dále se pak pro různé komponenty jako jsou například tlačítka, používají odstíny základních barev. Preprocesory poskytují opět mnoho funkcí, a proto bude představen nejčastější případ použití barevných funkcí.

Například: barva X je použita na odkazy a podle grafického návrhu je potřeba použít na tlačítka barvu Y, která je o 10% tmavší než barva X.

Zápis LESS:	Výstup CSS:

`@color-x: #333;`	`a {`
	`  color: #333;`
	`}`

```
a{ color: @color-x }
button{ color: darken(@color-x,10%) }
```

```
button {
  color: #1a1a1a;
}
```

Příklad 10 - barvy v LESS

Příklad funkce fadeout(), která provede poloprůhlednost barvy.

```
color: fadeout(red, 10%);
// → color: rgba(255, 0, 0, 0.9);
```

3.5.14 **Podmínky**

Stylus s SASS nabízejí klasické podmínky jako jiné imperativní programovací jazyky. A to if/else konstrukci.

Zápis SASS:

```
$type: svetla;

header {
  @if $type == tmava {
    color: #333;
  } @else if $type == svetla {
    color: #f7f7f7;
  } @else {
    color: blue;
  }
}
```

Výstup CSS:

```
header {
  color: #f7f7f7; }
```

Příklad 11 - if/else SASS

LESS používá pro podmínky tzv. strážců. Fungují stejným způsobem, jen mají jinou syntaxi.

LESS:

```
.if-condition {
  foo: bar;
  & when (1 = 2) {
    baz: qux;
  }
}
```

CSS:

```
.if-condition {
  foo: bar;
}
```

Příklad 12 - LESS mixin guards

3.5.15 **Frameworky**

Frameworky jsou nástroje, které mají řadu předefinovaných tříd a obecně vyladěného stylu. Takovýto nástroj může zjednodušit a urychlit práci při tvorbě webových rozhraní.

CSS frameworky nezůstali pozadu a své zdrojové kódy převedli do preprocesorů. Není to žádná novinka. Frameworky vydávají neustále nové verze. Je tedy o ně velký zájem, ale často

se stává, že jsou použity tam, kde nemají. Respektive mohou, ale většinou se z těchto nástrojů stávají ¾ nepoužitelné části, které se na webu ani nevyužívají. Proto existují preprocesory a této problematice předcházejí. Všechny jednotlivé preprocesorové soubory jsou vloženy do jednoho souboru pomocí funkce import a zde si jen stačí okomentovat ty, které nebudou používány.

Jako nejúspěšnější Framework se považuje ve vývojářské komunitě Bootstrap jinak nazávaný Twitter Bootstrap. Dlouho dobu existoval v preprocesorové verzi jen pro LESS a tak mnozí vývojáři považovali tímto LESS lepším než ostatní preprocesory. Nyní už tomu je jinak. Bootstrap existuje i pro SASS komunitu. Nejdříve jej vyvíjeli v nadšenci SASS preprocesoru, ale dnes má oficiální kódy přímo od vývojářů Bootstrap.

SASS nabízí ještě také velmi populární mezi vývojáři Compass. Compass je tedy rozšířením SASS preprocesoru, ale nelze jej použít pro jiný z preprocesorů. Nabízí opět nespočet užitečných návrhových vzorů, funkcí apod. Na rozdíl od Bootsrapu Compass zvládne například vytvářet i tzv. sprity. Sprity jsou obrázky, ve kterých se nachází ostatní obrázky, které jsou na webu použity a slouží výhradně pro grafické doplnění webu. Rozhodně se do nich neukládají fotografie. Principem je skládání obrázků za sebe pomocí nějakého algoritmu, který vypočítá, kde který obrázek má být umístěný a Compass vytvoří CSS souřadnicemi v podobě výšky, šířky, a umístění obrázku ve spritu. Všechny obrázky, které tvoří grafiku webu, jsou v jednom, což je velice výhodné, pro celkovou rychlost načítaní samotného webu. Jelikož pak ideálním případě dochází pouze k jednomu http požadavku.

Stylus zatím má jen podporu od svých příznivců v podobně předělaného Bootrstrap frameworku, ale oficiální podpory se zatím nedočkal.

4 Vlastní práce

Předchozí teoretická část pojednávala stručně o historii HTML a CSS a obecně o CSS preprocesorech. Nyní bude na reálných příkladech vyzkoušena efektivita těchto vývojářských nástrojů.

Vlastní práce je rozdělena do několika jednotlivých reálných příkladů z praxe. Je ukázán způsob zapisování kaskádových stylů za pomocí preprocesorů. Samotné formátování je vysvětleno jen stručně, protože není cílem bakalářské práce.

4.1 Rychlý redesign

Klasickým případem bývá, že při dokončování či průběhu prací na webu dojde k situaci, kdy vývojář dostane pokyn od grafika nebo marketéra, aby se design změnil. Místo jedné hlavní barvy, která je použita všude na webu, má použít jiný její odstín nebo zaměnil za barvu zcela novou.

Vývojář bez CSS preprocesoru bude při rozsáhlejším projektu přemýšlet, kde všude musí barvu změnit, a pokud přebral práci po někom jiném, bude tento úkol ještě o to delší a může zabrat vývojáři až v řádu hodin.

S preprocesorem se dá problém takového rázu vyřešit jednou změnou a kompilací, pokud se psal kód určitou konvencí a změna je pak v řádu sekund vyřešena. Z toho vyplývá, že je třeba přemýšlet dopředu. Vkládat proměnou při vývoji tam, kde má z logiky věci být, v tomto případě u barvy. Jinak se zmíněné rychlé změny designu na jednom řádku nedosáhne.

Realizace se provádí i při menším projektu z již připravené základní struktury, která je obecná a dá se snadno modifikovat a přenášet. Tento problém vysvětluje kapitola „Import a parciální soubory“ v teoretické části bakalářské práce.

V souboru _variables.scss dojde, k vytvoření, proměnné „primary-color“, která značí primární barvu identifikující web. Většinou se jedná o barvu, kterou jsou obarvena tlačítka, odkazy, pseudo hover třída či části komponenty.

Ukázka kódu v souboru _variables.scss	Použití proměnné primary-color

```scss
$gray: #000;
$green: #57AD23;
$blue: #6ea3ff;
$red: #f00;
$orange: orange;

$primary-color: $green;
$secondary-color: $blue;

$border-radius: 4px;
```

```scss
.btn{
  color: #fff;
  background: $primary-color;
  display: inline-block;
  padding: 0.5rem 1rem;
  text-decoration: none;

  &:hover{
    background: darken($primary-
color,15%);
  }
}
```

Příklad 13 - změny barev pomocí proměnných

Bez preprocesoru by bylo nutné ručně vyhledat, kde všude je barva použita a změnit ji na každém místě zvlášť. Sice jde použít například v editoru funkce pro nahrazení fráze za jinou, ale to by mohlo způsobit obarvení i u elementů, u kterých by se barva měnit neměla.

Nyní lze při správném zapsání měnit primární barvu webové prezentace pomocí editace jedné proměnné. Častým příkladem v praxi může být například sezónní design webu. Například v zimě si klient webu se zájezdy rozhodne, aby například oranžová barva, jenž se používá doposud, změnila na modrou barvu, kvůli sezónní marketingové strategii.

4.2 Grid systém

Grid systém lze chápat jako tzv. „mřížku“ webu. Lze tak snadno web rozdělit na části v určitém poměru. Hojně používaným se stal grid se strukturou 12 sloupců. Tento příklad bude vytvářen SASS preprocesorem v syntaxi SCSS.

Nejprve se vymyslí logika dané problematiky. Grid se jednoduše skládá z řádků a sloupců. V mnoha případech při návrhu se základní layout skládá v podstatě ze sloupců. Proto se tedy hojně začal používat grid systém. Klasicky se jedná o určité velikosti, jako například ¾ jsou pro hlavní obsah a ¼ pro postranní panel nebo rozložení produktů na webu, aby byly 2 nebo 4 vedle sebe. Tudíž se dá logicky odvodit, že pokud je sloupec přes celou šířku elementu, ve kterém je vložený, zaujímá 100% šířky. Nutno podotknout, že tento příklad počítá s responzivním designem, takže se bude šířka definovat v procentech. Při 12 sloupcovém layoutu znamená, že největší by se vypočítal ze vztahu (12/12)*100%. Z toho vyplývá například velikost sloupce, který by měl zaujímat polovinu prostoru v obalovém elementu, a to ze vztahu (6/12)*100%. Z toho se dá odvodit jednoduše vzorec pro výpočet a lze jej zakomponovat do preprocesorů.

Nejdříve je třeba vytvořit onen obalový element v tomto smyslu řádek, který se nazve třídou row. Jelikož sloupce se budou skládat za sebe do leva pomocí CSS vlastnosti float, je nutné řádkem obtékání zrušit tzv. clearfixem. Clearfix funguje tak, že použije vlastnost clear před třídu row a na konec třídy row, pomocí after a before pseudo elementů. Do elementu se třídou row se skládají floatované elementy.

Sloupce se řadí za sebe pomocí CSS vlastnosti float s vypočítanou šířkou, která se bude lišit u každého prvku.

Zápis vypadá v SASS takto:

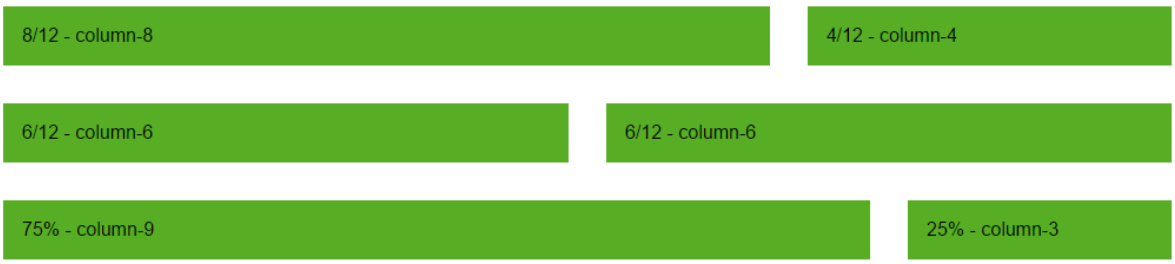
```
$columns: 12;
$gutter: 2em;

.row{
  @include clearfix;
  margin-left: -$gutter/2;
  margin-right: -$gutter/2;
}
.column{
  min-height: 1px;
  padding-left: $gutter/2;
  padding-right: $gutter/2;
}
```

```
@media screen and (min-width: $screen-small){

  .column{ float: left; }

  @for $i from 1 through $columns {
    .column-#{$i} {
      width:  ($i/$columns) * 100%;
    }
  }

}
```

Příklad 14 - grid systém

Cyklus for má na starosti výpis třídy column-*, kde * značí proměnnou i, která se při každé iteraci zvětší o 1. Celý cyklus je obalený media queries, která zařídí, že na zařízení s displejem menším jak proměnná $screen-small dojde k přerušení šířky stanovené cyklem. Použije se tedy 100% šířka a na mobilních zařízeních bude obsah více čitelný.

Výsledkem je tento výstup:



Obrázek 7 - Názorná ukázka gridu

4.3 Prefixy v praxi

Běžná věc při vývoji stránek je práce s prefixy. Dříve se používaly například online generátory, kterými vývojáři pomocí interaktivního ovládání vygenerovali kus kódu, jenž se vložil do stylů. Nebo se psaly kódy do zvláštního souboru, a když bylo potřeba nějaké CSS3 vlastnosti, která měla prefixy, bylo ji třeba odtud zkopírovat na dané místo. Velice výstižnou ukázkou se jeví problém zaoblení rohů. Po chvíli vývoje se rozhodne, že místo 5px zaoblení rohů bude border-radius 6px a opět bez preprocesoru nastává problém se složitou úpravou. Zde se dá využít již zmíněných proměnných, jelikož většinou grafika má být konzistentní, což v tomto případě znamená, že zaoblení rohů by mělo být na většině míst stejné. Až na výjimky, kdy je třeba například použít 50% zaoblení, které způsobí, že objekt bude zcela zaoblený neboli zakulacený. Stanoví se tedy nějaká defaultní hodnota pro zakulacené rohy do proměnné s názvem border-radius. Zpět k prefixům. Tento neduh, který CSS3 vývojářům přinesl, se dá preprocesory obejít. Protože se ve své podstatě jedná o kód, který je všude stejný, akorát se mění velikost zaoblení. Použije se tedy parametrického mixinu. Následně bude příklad ukázán v preprocesoru LESS.

Do souboru mixin se vloží mixin, který se bude starat o zaoblení rohů. Po takto připraveném mixinu jej stačí použít. V tomto případě je použit na komponentě button označující tlačítka webu.

Zápis v LESS:

```
@border-radius: 4px;

.border-radius(@size: @border-radius){
  -webkit-border-radius: @size;
  -moz-border-radius: @size;
  border-radius: @size;
}
//Zcela kulatý element pomocí
jiné hodnoty parametru
.circle{ .border-radius(50%) }
//Zaoblené elementy defaultní
hodnotou
.btn{ .border-radius(); }
.links a{ .border-radius() }
```

Výstup v CSS:

```
.circle {
  -webkit-border-radius: 50%;
  -moz-border-radius: 50%;
  border-radius: 50%;
}
.btn {
  -webkit-border-radius: 4px;
  -moz-border-radius: 4px;
  border-radius: 4px;
}
.links a {
  -webkit-border-radius: 4px;
  -moz-border-radius: 4px;
  border-radius: 4px;
}
```

Příklad 15 - zaoblení rohů pomocí parametrického mixinu

Při jakékoliv změně v LESS stačí upravit jedna hodnota bez preprocesoru o značnou dobu déle.

Ve vygenerovaném CSS je vidět, že po použití na více místech vzniká stejný kód a neustále se musí ručně psát, kopírovat nebo editovat hodnoty. Toto je velmi nepraktické při práci na projektu, který může být kdykoliv třeba modifikovat.

4.4 Komponenta produkt

Pro komponentu produktu je nejdříve nutné si definovat nějaký náčrtek, nebo už mít připravený grafický návrh. Pro ukázku pomůže například hrubý návrh, který je vidět na obrázku.

Obrázek 8 - Náčrt komponenty produktu

Lze takový příklad řešit různými metodami stylování. Na ukázku bude zvolena metoda BEM v porovnání s klasickým zanořováním.

Pomocí metody BEM, která je vysvětlena v teoretické části CSS, se stanoví názvy jednotlivých částí komponenty. Jméno komponenty vyplyne většinou z podstaty věci. V tomto případě se nabízí nazvat komponentu product. Dále se pak postupuje nejlépe od shora návrhu a postupně se pojmenovávají další části komponenty. Všechny produkty budou mít stejný základ nastylovaný a když se bude jednat o produkt, který je v akční nabídce, bude graficky odlišen. Tím se využije i třetí prvek BEM modifikátor.

Blok

- Product – Komponenta produkt

Elementy

- Product__image – obrázek produktu
- Product__title – nadpis produktu
- Product__description – popis produktu
- Product__price – cena produktu
- Product__btn – tlačítko pro zakoupení

Modifikátor

- Product--action – rozlišení akčního produktu od klasického

Komponenta product nebude mít pevnou šířku. Nýbrž bude zaujímat 100% obsahu elementu, do kterého se vloží. Pokud by se jí dali pevné rozměry, nedalo by se s ní, tak dobře pracovat například při responzivním designu. Respektive dalo, ale takto je daleko více použitelná na jiném místě a nemusí se na každé místo stylovat zvlášť.

Pomocí preprocesoru lze totiž jméno třídy, která je na začátku v názvu ostatních elementů, použít, aniž by se musel psát název komponenty znovu. Slouží k takovému zápisu znak &.

Výsledné zkompilované CSS nevyužívá žádných kaskád. Vše na úrovni jedné třídy. Při zápisu stylu metodou BEM je pak i pro programátora, který šablonu bude napojovat například na redakční systém v HTML jasné, který element co znamená a k čemu je.

Klasický kaskádový zápis bude simulovat preprocesor Stylus.

SASS – BEM

```
.product{
  background: #bdbdbd;
  padding: 20px;

  &__image{
    max-width: 100%;
    height: auto;
  }
  &__title{
    font-size: 1.5em;
  }
  &__description{
    font-size: 0.9em;
  }
  &__price{
    float: left;
  }
  &__btn{
    float: right;
  }
  &__footer{
    @include clearfix();
```

Stylus – kaskády

```
.product-cascade
  background #bdbdbd
  padding 20px

  header
    img
      max-width 100%
      height auto
    h3
      font-size 1.5em

  p
    font-size 0.9em

  footer
    &:before,&:after
      content ""
      display table
    &:after
      clear both
    div
      float left
```

```
  }
}
```

```
a
  float right
```

Příklad 16 - srovnání BEM za pomocí SASS a kákád ve Stylusu

Na první pohled je již vidět přehlednost prvního zápisu. Je daleko více srozumitelný, než zápis ve Stylus. Není myšlena nevýhoda Stylus jako takového, ale kaskádového zanořování. Někdo by mohl tvrdit, že výsledek je stejný. Pokud po předání nastylovaného obsahu putuje dále k backend programátorovi a pak většinou i odborníkům na SEO optimalizaci, může v HTML zaměnit nadpis <h3> za <h2>, za <p> apod. Dojde k rozpadu nakódovaného obsahu a vše se bude muset předělávat. Při zápisu BEM se podobná situace stane málokdy. Pouze tehdy pokud někdo vymaže název třídy.

Jde o to ukázat, jak se dá zpřehlednit kód stylů. Preprocesory pomáhají při přehlednosti referenčním znakem & při BEM zápisu.

4.5 Cyklus pro výpis ikon

Práce s cykly je ve Stylus preprocesoru velice intuitivní. Programátorům bude zajisté přívětivější než syntaxe LESS. Nedílnou součástí webových prezentací jsou stále ikony. Návštěvníkům webů ikonky sdělují na první pohled, o čem asi daná sekce bude.

```
icons = telefon, email, mapa;

for icon in icons {
  .{icon}-icon {
    background-image: url('/images/'+icon+'-icon.png');
  }
}
```

Do proměnné icons se vkládají všechny ikonky, které mají být použity na webu. V praxi totiž nastává situace, že přijdou podklady od grafika. Nařeže se grafika a může se začít kódovat front-end. Za nedlouho přijde další grafika a s ní i další sada ikonek. V případě použití podobného cyklu pak stačí jen do proměnné přidat názvy ikonek či jiných grafických prvků, které slouží k designu webu.

Cyklus pak projde všechny proměnné v icons, kde každé přiřadí background-image s příslušnou url adresou. Pro vložení jména ikonky poslouží název proměnné obalené znakem +.

```
.telefon-icon {
  background-image: url("/images/telefon-icon.png");
}
```

```
.email-icon {
  background-image: url("/images/email-icon.png");
}
.mapa-icon {
  background-image: url("/images/mapa-icon.png");
}
```

4.6 Hlášení na webu

Často označován vývojáři jako alert. Jedná se o věc, která je součástí dnes téměř každého webu. Nejedná se o nic jiného než hlášení různého charakteru, jenž se zobrazují návštěvníkům webů. Většinou se používají pro obecné informační sdělení, varování, úspěšné akci nebo například špatném hlášení při nesprávném přihlášení do uživatelské zóny.

```
$alert-danger-color: $red;
$alert-info-color: $blue;
$alert-warning-color: $orange;
```

Je třeba tato hlášení odlišit, aby bylo jasné, co se v dané situaci děje. Nejvhodnějším způsobem je barevné odlišení. V preprocesoru se hodí použít proměnné.

```
.alert{
  padding: 1em;
  border: 1px solid;
}
```

Avšak určitá část je stejná pro všechny hlášení. Zde se využije výhod technické vlastnosti @extend.

```
.alert-danger{
  @extend .alert;
  background: lighten($alert-danger-color,20%);
  color: darken($alert-danger-color,20%);
}

.alert-info{
  @extend .alert;
  background: lighten($alert-info-color,20%);
  color: darken($alert-info-color,20%);
}

.alert-warning{
  @extend .alert;
  background: lighten($alert-warning-color,20%);
  color: darken($alert-warning-color,20%);
}
```

Po krátkém pohledu je vidět, že zápis v preprocesoru SASS je sice obsáhlejší než výstup CSS a je nutné problematiku znát, ale za to lze pak snadno měnit barvy nebo společné parametry.

```
.alert, .alert-danger, .alert-info, .alert-warning {
  padding: 1em;
  border: 1px solid; }

.alert-danger {
  background: #ff6666;
  color: #990000; }

.alert-info {
  background: #d4e4ff;
  color: #0862ff; }

.alert-warning {
  background: #ffc966;
  color: #996300; }
```

5 Závěr

Nároky na kvalitu a rychlost v oblasti tvorby webových prezentací jsou značně velké. Tudíž vývojáři se musí neustále zajímat o nové technologie, aby drželi krok s rychle jdoucími moderními trendy tvorby webů. CSS preprocesory jsou jistě jednou z těchto technologií, která přináší značné efektivní funkce a metody zapisování kaskádových stylů.

I když se na první pohled zdá, že se vývojář musí učit něčemu novému, tak doba pro pochopení a užívání preprocesorů v praxi nebude dlouhá. Pro začínající vývojáře, který se rozhodne pro jeden z preprocesorů, není obtížné přejít na jiný.

V podstatě všechny vybrané preprocesory umí skoro stejné funkce a syntaxi. Při výběru preprocesoru se ukazuje vhodným parametrem velikost komunity, která se o preprocesor stará a udržuje jej v stabilním stavu, poslední datum aktualizace či nutnost potřeby frameworku. Nebo jiné subjektivní vývojářovy preference.

Největší výhodou preprocesorů je zajisté rychlá úprava kódu pomocí proměnných, mixinů a dalších zmiňovaných technických vlastností. Tento fakt se projeví jak na malém, tak i na větším projektu. V neposlední řadě se stávají preprocesory výhodou v projektech, na nichž pracuje více vývojářů, kteří používají kaskádové styly. Neustálé kopírování nebo psaní stejného kódu bez preprocesoru je velice neefektivní. Proto jsou zde preprocesory, které se o kód, který je stejný pro více prvků, postarají. Také vedou k lepší přehlednosti kaskádových stylů díky importu a parciálním souborů.

Mezi nevýhody CSS preprocesorů se dá zařadit nutná kompilace kódu do CSS. Musí se například stáhnout nějaký GUI nástroj či zakomponovat plugin do používaného editoru, který bude preprocesorový jazyk kompilovat. Také je zde i možnost, že používaný editor nebude preprocesor podporovat a bude nutné od preprocesoru upustit, nebo přejít na editor jiný. Některé GUI kompilátory jsou i placené a mohou některé zájemce tímto odradit. Další nevýhodou pro některé uživatele preprocesorů může být nutnost učit se novému jazyku.

Kaskádové styly jsou velice užitečným nástrojem pro tvorbu vzhledu webových stránek. Avšak jejich nové vlastnosti přináší jistá úskalí, která se preprocesory snaží odstraňovat. LESS je poměrně jednoduchý pro naučení syntaxe jelikož spadá do deklarativního jazyka, kde se snaží o téměř stejný zápis jako při psaní CSS bez preprocesoru. Tudíž není tak obtížné na tento jazyk přejít ze psaní CSS bez preprocesoru. SASS je spíše řazený pro vývojáře, kteří si více rozumí s programátorskými věcmi. Dokáže psát funkce téměř stejným zápisem jako v jiném

programovacím jazyce, a to například funkce if / else nebo cykly for. Stylus ukazuje svojí přednost v syntaxi. Po krátkém používání se vývojář může psaním jeho syntaxe nechat natolik unést, že nebude chtít používat jiný.

Při realizování jednotlivých příkladů na preprocesory, bylo jasně poukázáno na nedostatky CSS jazyka pro vývojáře v oblasti efektivity práce s těmito nástroji.

Ať už se vývojář rozhodne pro SASS, LESS nebo Stylus, přináší preprocesory velmi užitečné prostředky pro psaní stylů a při tvorbě větších projektů se stávají v dnešní době nezbytnou součástí front-end developerů. Nutno podotknout, že nepřináší do CSS nové vlastnosti, ale při krátkém seznámení vývojář pozná rozdíl v rychlosti tvorby stylů za pomoci této technologie. Všechny preprocesory jsou nejvíce používány při práci na větších projektech, kde se o kaskádové styly stará více vývojářů a mění se pracovní tým vývojářů.

6 Seznam zkratek

WWW – World Wide Web

CERN – Centre Européenne pour la Recherche Nucléaire

NCSA – National Center for Supercomputer Applications

HTML – HyperText Markup Language

XHTML – Extensible HyperText Markup Language

CSS – Cascading Style Sheets

DRY – Dont repeat yourself

BEM – Block, Element, Modifikátor

LESS – Leaner CSS

SASS – Syntactically Awesome Stylesheets

SCSS – Sassy Cascading Style Sheets

NPM – Node Package Manager

7 Seznam použitých zdrojů

1. **Connolly, Dan.** A Little History of the World Wide Web. *W3.* [Online] 2000. [Citace: 10. 10 2015.] https://www.w3.org/History.html.

2. **Janovský, Dušan.** HTTP protokol. *jakpsatweb.* [Online] [Citace: 10. 1 2015.] http://www.jakpsatweb.cz/server/http-protokol.html.

3. **Kosek, Jiří.** Historie a vývoj HTML. *htmlguru.* [Online] [Citace: 25. 12 2015.] http://htmlguru.cz/uvod-historie.html.

4. **Prokop, Marek.** Co je XHTML. *sovavsiti.* [Online] 1. 10 2001. http://www.sovavsiti.cz/c01242.html.

5. **Adaptic.** DHTML. *Adaptic.* [Online] [Citace: 26. 12 2015.] http://www.adaptic.cz/znalosti/slovnicek/dhtml/.

6. **Bright, Peter.** HTML5 specification finalized, squabbling over specs continues. *arstechnica.* [Online] 29. 10 2014. [Citace: 3. 10 2015.] http://arstechnica.com/information-technology/2014/10/html5-specification-finalized-squabbling-over-who-writes-the-specs-continues/.

7. **Owebu.** http://www.owebu.org/cze/css/index.php. *owebu.* [Online] [Citace: 12. 10 2015.] http://www.owebu.org/cze/css/index.php.

8. **Kosek, Jiří.** CSS. *Kosek.* [Online] 2000. [Citace: 12. 10 2015.] http://www.kosek.cz/vyuka/4iz228/prednasky/css.pdf.

9. **Janovský, Dušan.** Historie CSS. *Jak psát web.* [Online] [Citace: 26. 12 2015.] http://www.jakpsatweb.cz/css/css-historie.html.

10. **Jahoda, Bohumil.** BEM – způsob zápisu CSS. *jecas.* [Online] 3. 4 2015. [Citace: 14. 10 2015.] http://jecas.cz/bem.

11. **Grudl, David.** SASS, LESS, Stylus nebo čisté CSS? *phpfashion.* [Online] 2013. [Citace: 15. 10 2015.] https://phpfashion.com/sass-less-stylus-nebo-ciste-css-3.

12. **Michálek, Martin.** Průvodce CSS preprocesory. *Vzhůru dolů.* [Online] 27. 12 2015. [Citace: 30. 12 2015.] http://www.vzhurudolu.cz/blog/12-css-preprocesory-1.

13. **catswhocode.** 8 CSS PREPROCESSORS TO SPEED UP DEVELOPMENT TIME. *catswhocode.* [Online] 9. 9 2010. [Citace: 17. 10 2015.] http://www.catswhocode.com/blog/8-css-preprocessors-to-speed-up-development-time.

14. **LESS.** Dynamický jazyk pro tvorbu stylesheetů. *LESS.* [Online] [Citace: 25. 12 2015.] http://www.lesscss.cz/.

15. —. Getting started. *LESS.* [Online] [Citace: 27. 12 2015.] http://lesscss.org.

16. **Slant.** What are the best CSS preprocessors? *slant.* [Online] [Citace: 15. 12 2015.] http://www.slant.co/topics/217/viewpoints/2/~css-preprocessors-postprocessors~less.

17. **Giraudel, Hugo.** O sassu. *Sass-guidelin.* [Online] 2015. [Citace: 18. 10 2015.] http://sass-guidelin.es/cz/#o-sassu.

18. **Long, John.** Avoid nested selectors for more modular CSS. *thesassway.* [Online] 24. 5 2013. [Citace: 15. 12 2015.]

19. **Coyier, Chris.** The Extend Concept. *css-tricks.* [Online] 8. 7 2013. [Citace: 12. 12 2015.] https://css-tricks.com/the-extend-concept/.

20. **Verrecchia, Jonathan.** Deep Dive Into CSS Preprocessors. *slideshare.* [Online] 1. 6 2012. [Citace: 25. 10 2015.] http://www.slideshare.net/verekia/deep-dive-into-css-preprocessors/56-Thanksverekiacomslidescsspreprocessors_by_verekia.