Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky a kvantitativních metod

CSS preprocesory jako efektivní nástroj pro tvorbu frontendu

Bakalářská práce

Autor: Václav Černý
Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. Ing. Filip Malý, Ph.D.

Hradec Králové duben 2018

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 18.4.2018 Václav Černý

Poděkování:

Rád bych tímto poděkoval doc. Ing. Filipovi Malému, Ph.D., za odborné a bezproblémové vedení bakalářské práce, vstřícnost, a především cenné rady.

Anotace

Cílem této bakalářské práce je zpracování problematiky takzvaných CSS preprocesorů jakožto nástroje, který efektivně odstraňuje řadu problémů při práci s kaskádovými styly neboli jazykem Cascading Style Sheets (CSS) v oblasti vytváření frontendové části webových prezentací a aplikací. Teoretická část je zaměřena na představení konceptu stylovacího jazyka CSS a následně na vlastnosti CSS preprocesorů. Průběžně poukazuje na vztah mezi těmito dvěma technologiemi a klade důraz na odhalení jejich užitečnosti. Část praktická obsahuje vlastní výzkum zabývající se detailním porovnáním všech základních funkcí a velikosti popularity tří nejpoužívanějších CSS preprocesorů jimiž jsou Sass, Less a Stylus. Výsledkem porovnání je bodové hodnocení těchto zástupců určené dle vhodnosti jejich nabízených řešení.

Annotation

Title: CSS preprocessors as an efficient tool for creation of frontend

The goal for this bachelor thesis is to explain the process of so called CSS preprocessors as a tool, which effectively eliminates a handful of problems regarding the work with Cascading Style Sheets (CSS) in the area of developing front-end parts of website presentations or applications. The purpose of the theoretical section is to introduce the concept of the CSS style sheet language and subsequently for the characteristics of CSS preprocessors. Continuously points out the relationship between these two technologies and tends to emphasize the revealing of their benefits. On the other hand, the practical part contains my own research, diving in details for the comparison of all primary functions and popularity sizes of the three most used CSS preprocessors, in name Sass, Less and Stylus. The outcome of this comparison is a point-based evaluation of these representatives designed by appropriateness to their solutions.

Obsah

1 Úvod 4

2 Kaskádové styly 7

2.1 Co je CSS 7

2.2 Důvod vzniku CSS 8

2.3 Základní principy CSS 10

2.3.1 Syntaxe CSS 11

2.3.2 Aplikování CSS 12

2.3.3 Kaskádování v CSS 13

2.3.4 Zpracování CSS 13

2.4 Vývoj CSS 14

2.4.1 CSS Level 1 14

2.4.2 CSS Level 2 15

2.4.3 CSS Level 2 Revision 1 15

2.4.4 CSS Level 3 15

2.5 Nedostatky CSS 16

2.5.1 Nekonzistence implementace CSS 16

2.5.2 Spravovatelnost a udržitelnost CSS 17

3 CSS preprocesory 19

3.1 Důvod vzniku CSS preprocesorů 19

3.2 Jak funguje CSS preprocesor 20

3.3 Odlišnosti CSS preprocesorů 21

3.3.1 Syntaktická stránka jazyka 21

3.3.2 Poskytované funkce 22

3.3.3 Implementace překladače 22

3.3.4 Méně výrazné rozdíly ... 23
3.4 Možnosti překladu do CSS ... 23
3.4.1 Překlad v prohlížeči ... 24
3.4.2 Překlad na serveru ... 24
3.4.3 Překlad na klientu ... 25
3.4.4 Rychlost překladu ... 25
3.4.5 Spouštění překladu ... 26
3.5 Instalace CSS preprocesoru ... 26
3.6 Přehled dostupných CSS preprocesorů ... 27
3.7 Základní funkce CSS preprocesorů ... 29
3.7.1 Proměnné ... 30
3.7.2 Hnízdění ... 31
3.7.3 Mixiny ... 32
3.7.4 Extend ... 34
3.7.5 Import ... 35
3.7.6 Operátory ... 37
3.8 Pokročilé funkce CSS preprocesorů ... 38
3.9 Praktické použití základních funkcí ... 39
3.9.1 Organizace a spravovatelnost kódu ... 39
3.9.2 Úspora času a konzistence kódu ... 43
3.9.3 Flexibilita ... 46
3.9.4 Knihovny ... 46
4 Porovnání CSS preprocesorů ... 48
4.1 Představení zástupců porovnání ... 49
4.1.1 Sass ... 50
4.1.2 Less ... 52

4.1.3 Stylus ... 53
4.2 Porovnání základních funkcí ... 54
4.2.1 Proměnné ... 55
4.2.2 Hnízdění ... 63
4.2.3 Mixiny ... 68
4.2.4 Extend ... 76
4.2.5 Import ... 81
4.2.6 Operátory ... 87
4.2.7 Shrnutí ... 88
4.3 Porovnání nástrojů ... 90
4.4 Porovnání popularity ... 90
4.4.1 GitHub ... 91
4.4.2 Node Package Manager ... 92
4.4.3 Stack Overflow ... 93
4.4.4 CSS frameworky ... 93
4.4.5 Aplikace třetích stran ... 94
4.4.6 Webové prezentace ... 94
4.4.7 Google Trends ... 95
4.4.8 Shrnutí ... 95
5 Shrnutí výsledků ... 97
6 Závěry a doporučení ... 100
7 Seznam použité literatury ... 101
8 Seznam použitých obrázků ... 104
9 Seznam použitých tabulek ... 105
10 Seznam použitých příkladů ... 106

1 Úvod

Ať už se jedná o menší webové prezentace, nebo komplexní webové aplikace, jejich vytváření znamená použití několika příslušných nástrojů. Jedním z nich jsou takzvané Cascading Style Sheets (CSS), neboli kaskádové styly, které neodmyslitelně patří k základním stavebním kamenům podoby webu už od jeho počátků. Svým zápisem umožňuje deklarativní jazyk CSS popisovat způsob zobrazení HTML elementů strukturovaného dokumentu, tedy jejich grafický vzhled a uspořádání. Nicméně vzhledem k vývoji CSS a ostatních souvisejících technologií se stále objevují nové možnosti publikování na webu. S rozsáhlými možnostmi se pojí rozsáhlé zdrojové kódy. Čím větší se stává webová prezentace či aplikace, tím větší se pochopitelně stává obsahová stránka CSS. V takovém případě se kód CSS stane velice rychle těžko spravovatelným, obzvlášť pracuje-li s ním celý tým lidí. Jelikož samotné CSS nemá doposud vhodné prostředky pro lepší spravovatelnost a udržitelnost kódu, vznikly nástroje zvané CSS preprocesory, které tyto nedostatky zdárně řeší.

Jazyk CSS je ve webovém prostředí nenahraditelný. Jeho využití roste a stejně tak potřeba po co nejvíce efektivní práci s ním. Vývoj CSS se zdánlivě začal od jisté verze ubírat dobrým směrem, přesto zásadní nevýhody doposud přetrvávají a problematika CSS preprocesorů se tak stává stále více důležitější. Proto obsahem této bakalářské práce je právě zpracování tohoto tématu.

První část práce je teoreticky zaměřena na představení CSS a následně CSS preprocesorů jakožto užitečného nástroje, který především řeší slabiny CSS a má tak za úkol zjednodušit a zefektivnit tvorbu frontendové části webových prezentací a aplikací. Část praktická obsahuje vlastní výzkum zabývající se detailním porovnáním tří nejpoužívanějších zástupců CSS preprocesorů, jimiž jsou Sass, Less a Stylus.

Cílem teoretické části práce je přiblížení problematiky CSS preprocesorů z pohledu jejich výhod, které oproti standardnímu CSS nabízí. Dále objasnění důvodu vzniku CSS preprocesorů. Poukázání na zásadní nedostatky CSS a jejich vhodná řešení za pomoci CSS preprocesorů a zároveň zdůraznění příčin vzniku těchto nedostatků a

případné snahy o jejich nápravu ze strany novějších standardů CSS. Dále je v této části cílem podrobné rozebrání řady základních principů souvisejících s fungováním CSS preprocesorů. Podstatnou součástí je pak představení několika hlavních jazykových funkcí, jejichž používání tvoří většinu času práce s CSS preprocesory.

Vlastní výzkum si jakožto druhá část práce bere za cíl vzájemné porovnání CSS preprocesorů Sass, Less a Stylus a vhodné určení jednoho CSS preprocesoru, který bude jako výsledek porovnání považován ze všech tří zástupců za nejlepší nástroj. Důležitým bodem je detailní poukázání na rozdíly, které nejsou pouze předmětem odlišnosti syntaxe, ale i komplikovanějšího chování, a následně navíc objektivní stanovení relevantnosti onoho rozdílu. Účelem porovnání je zároveň určení předností jednotlivých zástupců a vytvoření přehledu, ze kterého jasně vyplývají výhody a nevýhody daného CSS preprocesoru. Dojde k vyvrácení obecného názoru, že CSS preprocesory jsou prakticky stejné. Proto dílčím úkolem porovnání je odhalení skutečnosti, že stejnojmenné funkce mají často výrazně odlišné zpracování a některé z nich se dají skutečně považovat minimálně za lépe uchopitelné.

Práce cílí jak na začínající kodéry, tak na ty pokročilé. Nováčkům v této oblasti práce nabídne nejen úvod do CSS, ale především objasnění vztahu mezi CSS a CSS preprocesory. Přesvědčení, že kterýkoliv CSS preprocesor je opravdu užitečným nástrojem, je právě mnohdy nutným krokem k podnícení jejich studování. Pokročilí kodéři pak mohou najít užitek především v detailním porovnání CSS preprocesorů, které lze využít jako podklad pro výběr správného zástupce (Sass, Less, Stylus).

Aby bylo objasnění vzniku CSS preprocesorů a jejich výhod kompletní, je nutné do obsahu práce zahrnout samostatnou kapitolu věnující se pouze CSS. Ta vysvětluje jeho koncept i vývoj a je nezbytným úvodem do problematiky CSS preprocesorů.

Představení jednotlivých základních funkcí CSS preprocesorů je vždy doplněno o názorný příklad praktického použití pomocí zápisu konkrétního validního kódu (Sass) včetně jeho detailního popisu a očekávaného výsledku. Pro lepší pochopení vlivu CSS preprocesorů na efektivitu práce je vhodné popis základních funkcí zopakovat v rámci typicky problematických situací v kódu CSS, které jsou díky nim často vyřešeny.

Porovnání tří zástupců CSS preprocesorů je jakožto vlastní výzkum provedeno ve dvou různých oblastech. První, a podstatnější z nich, se zaměřuje na porovnání základních funkcí představených v teoretické části práce. Jako důvěryhodný podklad pro stanovení rozdílů mezi CSS preprocesory poslouží jejich oficiální dokumentace. V případě, že je potřeba odhalit detailnější odlišnosti ve složitějším chování funkcí, které není v oficiálních dokumentacích uvedeno, je nutné jej ve všech třech CSS preprocesorech zvlášť řádně otestovat.

Kontrastem k porovnání funkcí je porovnání popularity CSS preprocesorů. Toto porovnání probíhá na základě vybraných webových služeb třetích stran, u kterých se dá popularita jednotlivých CSS preprocesorů jednoznačně číselně vyjádřit, přičemž se takové číslo dá díky velké návštěvnosti těchto služeb považovat za relevantní vzhledem k světovým trendům.

Výsledek porovnání by měl být reprezentován jednoduchou souhrnnou formou, a zároveň by měl naplňovat své cíle. Proto CSS preprocesory v průběhu porovnání získávají bodové hodnocení podle předem zvolených pravidel.

2 Kaskádové styly

Nástroj označovaný jako CSS preprocesor ve své podstatě pouze rozšiřuje myšlenku a možnosti nástroje jiného, konkrétně jazyka Cascading Style Sheets (CSS) neboli takzvaných kaskádových stylů. Svou existencí jej tak dovádí k jisté dokonalosti. Proto problematika fungování samotného CSS, bez přítomnosti preprocesoru, je stěžejní k porozumění tohoto tématu. Stejně tak nedostatky CSS hrají důležitou roli v jeho vývoji a ve vzniku CSS preprocesorů.

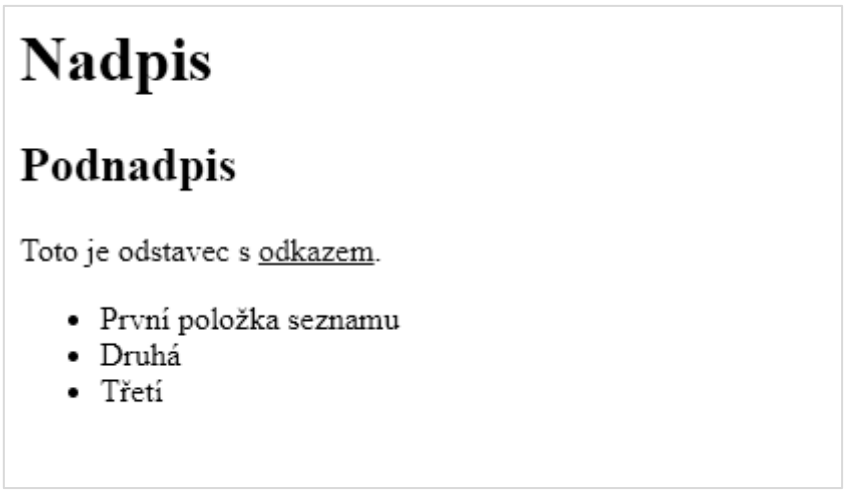
2.1 Co je CSS

Kaskádové styly jsou v teoretické i praktické rovině popsány jako stylovací jazyk, pomocí kterého lze určit, jakým způsobem se bude uživateli zobrazovat daný dokument, tedy jak bude prezentován jeho celkový vzhled a uspořádání. Obvykle je takovým dokumentem textový soubor strukturovaný pomocí značkovacích jazyků – nejčastějším zástupcem značkovacích jazyků je HTML, ale stejně tak se můžete setkat s jinými značkovacími jazyky, jako je SVG nebo XML [1].

Vzhledem k tomu, že CSS je zpravidla nejčastěji využíváno k tvorbě webových stránek, jejichž struktura je popisována pomocí jazyka HTML, dává nám možnost definovat vzhled HTML dokumentu a určit tak jeho styl od barev, písem, uspořádání až po speciální vizuální efekty či animace. Taktéž nám umožňuje přizpůsobit webovou stránku různým typům zařízení, jako jsou velké obrazovky, malé obrazovky nebo tiskárny [2] a to z důvodu odlišných potřeb pro různá zařízení.

Prezentace dokumentu uživateli znamená jeho přeměnění na co nejpoužitelnější formu pro dané publikum [1], což je účelem právě CSS.

Na obrázku č. 1 je vidět, jakým způsobem zobrazí webový prohlížeč několik základních HTML elementů bez jakýchkoliv přidružených kaskádových stylů. Obrázek č. 2 ukazuje zobrazení těchto HTML elementů webovým prohlížečem při definování jejich vzhledu pomocí CSS.



Obrázek 1: Zobrazení HTML elementů v prohlížeči bez použití CSS [vlastní zpracování]

Obrázek 2: Zobrazení HTML elementů v prohlížeči s použitím CSS [vlastní zpracování]

2.2 Důvod vzniku CSS

Kolem r. 1990 vytvořil Tim Berners-Lee tři specifikace, které utvořily základ projektu World Wide Web: HyperText Markup Language (HTML) byl vyvinut jako dokumentový formát pro web; Universal Resource Locator (URL) byl přidán pro reprezentaci vazeb mezi dokumenty; a HyperText Transfer Protocol (HTTP) byl vyvinut pro přenos dokumentů mezi počítači v internetu [3].

Následně roku 1993, s příchodem webového prohlížeče NCSA Mosaic, začal být web používanější než kdy dříve. Čím dál tím více lidí začínalo prohlížet a číst stále

rostoucí počet webových dokumentů, a web se tak stával populárním. Zároveň vzhledem k rostoucímu počtu uživatelů se mnohem více lidí začalo o web zajímat z hlediska možnosti tvorby obsahu, díky čemuž se tak používání HTML rozšířilo. To samozřejmě vedlo postupem času ke stále větším požadavkům ze strany autorů na možnosti vzhledových uprav, které by mohly definovat, jak dokument vypadá.

Ovšem HTML bylo navrženo jako formát popisující logickou strukturu dokumentu, nikoliv vizuální, a proto výsledná podoba, zahrnující například velikost a barvu písma, byla vždy určena jednotlivými webovými prohlížeči zvlášť. To však znamenalo velmi omezené možnosti při tvorbě dokumentů.

Jako reakce na stížnosti autorů ohledně nemožnosti ovlivnění vzhledu svých dokumentů se v HTML začaly objevovat prvky řešící tento problém. Zavedení takzvaných prezentačních značek v HTML byl krok směrem dolů na žebříčku abstrakce [3]. Jelikož weboví autoři stále požadovali více prezentačních možností pro své dokumenty, HTML se průběžně začalo vyvíjet spíše do prezentačního než do strukturálního jazyka [3]. Aby se tomuto kroku dolů na žebříčku abstrakce zabránilo, bylo vyvinuto CSS jako stylovací jazyk pro web [3].

Hlavní myšlenkou vzniku CSS je ty tedy především oddělení popisu struktury dokumentu (HTML) od popisu vzhledu (CSS). Oddělení kódu HTML od CSS usnadňuje údržbu stránek, sdílení stylů mezi stránkami a přizpůsobení stránek různým prostředím [2]. Jedná se o oddělení struktury (nebo obsahu) od prezentace [2].

V příkladu č. 1 je zastaralý zápis HTML v případě použití jeho prezentačních možností za účelem definování vzhledu jednotlivých elementů. V příkladu č. 2 je zápis tytéž HTML struktury, s rozdílem použití externího CSS pro definování vzhledu mimo HTML namísto prezentačních značek. Zápis HTML na obou obrázcích vede ke stejnému výsledku, přičemž ten druhý z nich je jednoznačně na první pohled přehlednější.

```
<body bgcolor="#fafafa" text="grey">
<h1 align="center"><font color="#0a66f9">Nadpis</font></h1>
<h2 align="center">
      <font color="#0c77f6">Podnadpis</font>
</h2>
<p align="center">
      <font size="4" color="#0a294b" face="arial">
            Toto je odstavec
      </font>
</p>
<ul align="center" border="1" bgcolor="darkgrey"
bordercolor="#22435f">
      <li type="circle" valign="center" align="left">
            <font size="3" color="#345c7b">
                  První položka seznamu
            </font>
      </li>
      <li type="circle" valign="center" align="left">
            <font size="3" color="#345c7b">Druhá</font>
      </li>
      <li type="circle" valign="center" align="left">
            <font size="3" color="#345c7b">Třetí</font>
      </li>
</ul>
</body>
```

Příklad 1: Zápis HTML s použitím prezentačních prvků [vlastní zpracování]

```
<body>
      <h1>Nadpis</h1>
      <h2>Podnadpis</h2>
      <p>Toto je odstavec.</p>
      <ul>
            <li>První položka seznamu</li>
            <li>Druhá</li>
            <li>Třetí</li>
      </ul>
</body>
```

Příklad 2: Zápis HTML v případě použití CSS [vlastní zpracování]

2.3 Základní principy CSS

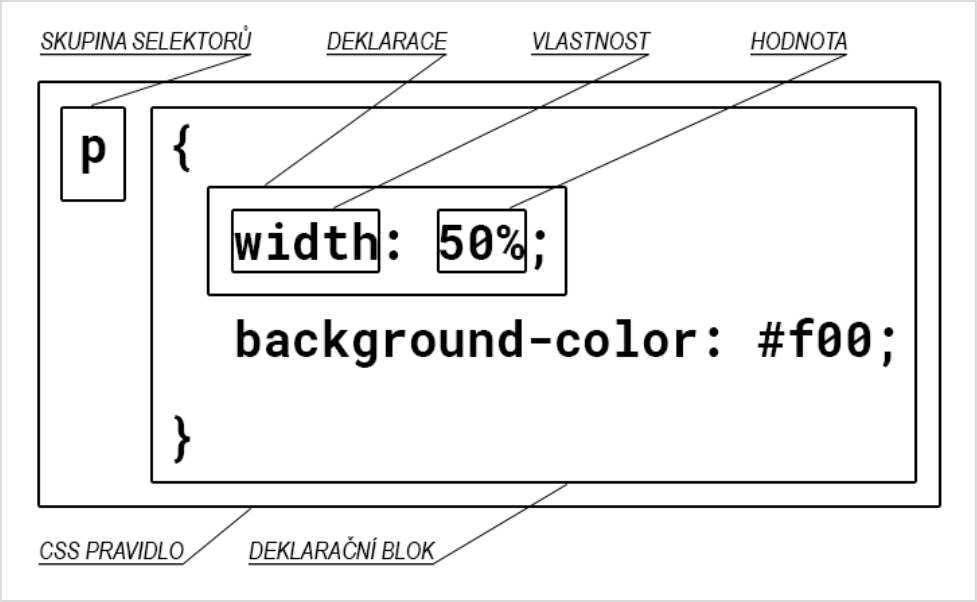
Kaskádové styly jsou velmi intuitivním a snadno uchopitelným jazykem. Jeho používání vychází z potřeb tvůrců webových stránek co nejjednodušeji definovat

vzhled jejich dokumentům, a to deklarativním způsobem. Proto jsou základy CSS například oproti mnohým programovacím jazykům značně méně komplexní.

2.3.1 Syntaxe CSS

Pomocí definování takzvaných CSS pravidel je možné docílit vytvoření vzhledu webech stránek. Přesněji jsou tyto pravidla schopna ovlivnit, jakým způsobem se budou jednotlivé elementy HTML dokumentu zobrazovat v rámci webových prohlížečů. Pro vytvoření takového pravidla je nutné dodržovat formát zápisu popsaný v CSS specifikaci, která je doporučeným standardem vytvořeným mezinárodním konsorciem W3C, jež má na starost vývoj webových standardů včetně právě CSS. Zápis pravidla se skládá ze dvou hlavních částí:

- Sada vlastností, které mají nastavené své hodnoty, určující, jak se bude HTML obsah zobrazovat, například je-li chtěno, aby šířka prvku byla 50 % šířky jejího nadřazeného prvku a jeho pozadí bylo červené [1].
- Selektor, který vybere jeden nebo více prvků, na kterých budou použity hodnoty vlastností [1]. Například je-li chtěno uplatnit CSS pravidlo na všechny odstavce v HTML dokumentu [1].



Obrázek 3: Popis částí syntaxe CSS pravidla [vlastní zpracování]

Jak je znázorněno na obrázku č. 3, skupina selektorů určuje, na které HTML elementy se deklarační blok vztahuje. Následující deklarační blok je ohraničen složenými závorkami a obsahuje jednotlivé deklarace, vzájemně oddělené středníkem, popisující konkrétní styl zobrazení. Deklarace se skládá ze dvou částí – vlastnost a její hodnota, přičemž tyto části jsou oddělené dvojtečkou. V tomto případě by byl každý odstavec zobrazen na 50 % šířky svého nadřazeného elementu a měl by červené pozadí.

2.3.2 Aplikování CSS

Aby bylo možné CSS pravidla použít, musí být někde zapsána. K dispozici jsou tři odlišné způsoby použití CSS v rámci HTML dokumentu. Prvním a nejpoužívanějším způsobem je použití externích stylů, kdy zápis CSS je uveden v samostatném souboru s danou příponou *„css“*. Následně je na tento soubor odkázáno v HTML dokumentu pomocí elementu link, aby došlo k jejich propojení. Tato metoda je jednoznačně nejlepší a přináší velkou výhodu, protože může být propojen jeden CSS soubor s několika HTML dokumenty, a tím pádem být použit pro změnu stylu více dokumentů najednou, přičemž potřebné změny v CSS pravidlech se tak provádí pouze na jednom místě v jednom souboru.

Druhým způsobem použití jsou interní styly. V tomto případě dochází k zápisu CSS pravidel přímo do HTML dokumentu. Zápis je vepsán do elementu style, který je umístěn uvnitř elementu head. U více stejně vypadajících dokumentů, či některých jejich částí, by muselo dojít k duplikování CSS do všech jednotlivých dokumentů, což je velmi nepraktické z hlediska provádění změn.

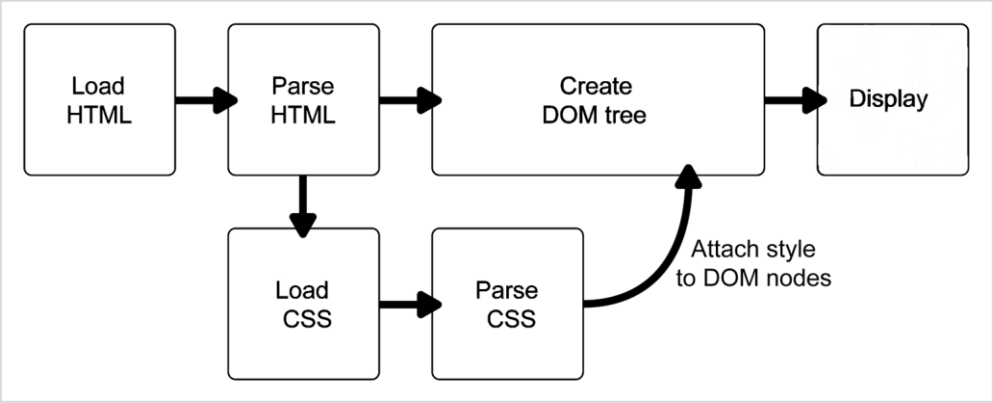
Poslední možností, jak CSS aplikovat, jsou inline styly. Jedná se o způsob, kdy jsou deklarace zapsány pomocí atributu style přímo do konkrétního elementu, který ovlivňují. Takový postup je z hlediska udržování ještě horší než interní styly, protože v případě nutnosti změn se musí upravit každý jeden element dokumentu obsahující danou deklaraci. Jelikož dochází k míchání CSS a HTML, celý dokument se stává nečitelným a těžce pochopitelným. Tento přístup odporuje důvodu vzniku CSS, kterým je oddělení popisu vzhledu od popisu struktury.

2.3.3 Kaskádování v CSS

Zkratka CSS je utvořena z celého názvu Cascading Style Sheets, česky překládáno jako kaskádové styly. Spojení style sheets vyplývá samozřejmě z faktu, že CSS je takzvaný style sheet language, v češtině stylovací jazyk, který obecně popisuje vzhled a uspořádání dokumentu. Nicméně slovo cascading, česky kaskádové, má taktéž svůj význam, který definuje jednu z důležitých funkcí CSS. Kaskádové styly tedy proto, protože na jednu konkrétní část HTML dokumentu může být aplikována více než jedna stylová deklarace. Pokud se více stylových deklarací pokouší nastavit konkrétní hodnotu vlastnosti určitému prvku, mechanismus kaskád vybere pouze jednu vítěznou deklaraci [3]. Vítězná deklarace bude mít plnou kontrolu nad danou hodnotou [3].

2.3.4 Zpracování CSS

Když prohlížeč zobrazuje dokument, musí zkombinovat obsah dokumentu s informacemi o jeho stylu [1]. Tento proces probíhá v několika krocích, které jsou vyobrazeny na obrázku č. 4.



Obrázek 4: Proces zobrazení dokumentu prohlížečem [1]

Nejprve dojde k načtení samotného HTML dokumentu, ten je vzápětí rozebrán a převeden na takzvaný DOM (Document Object Model), který odráží skladbu dokumentu v podobě stromové struktury. Každý prvek, atribut a kus textu zapsaný ve značkovacím jazyce se stane bodem ve stromové struktuře DOM [1]. Některé body pak mohou být ve struktuře reprezentovány jako potomci nadřazených

rodičovských bodů. Také mohou mít sourozenecké body, leží-li dané prvky v dokumentu na stejné úrovni.

V dalším kroku prohlížeč načte CSS, rozebere jednotlivá pravidla a přiřadí příslušné deklarace vlastností a hodnot bodům struktury DOM, které odpovídají prvkům definovaných pomocí selektorů. Vzniká tak další stromová struktura zvaná CSS Object Model.

Jakmile jsou obě tyto struktury vytvořené, prohlížeč pomocí zkombinování modelů DOM a CSSOM provede dva důležité kroky vedoucí k zobrazení webových stránek. Nejprve vytvoří uspořádání prvků, takzvaný layout, ve kterém spočítá přesné souřadnice, na kterých by se DOM body měly na obrazovce zobrazit. Poté začne postupně jednotlivé DOM body vykreslovat, včetně aplikování vizuálních stylů.

2.4 Vývoj CSS

Dnešní podoba CSS se pochopitelně v mnohém výrazně odlišuje od verze, která byla vydána jako první. Stejně jako většina dalších jazyků, i CSS prochází již řadu let vývojem a v průběhu této doby se mu dostalo několika změn a rozšíření, ať už více či méně povedených. Je zajímavé sledovat, jakým směrem se vývoj CSS ubíral a dále ubírá.

2.4.1 CSS Level 1

První návrh podoby CSS byl přednesen roku 1994. O rok později se mezinárodní konsorcium W3C ujímá vývoje kaskádových stylů a následně roku 1996 představuje specifikaci zvanou CSS Level 1, kterou vydává jako oficiální doporučení.

Vydáním CSS 1 začalo být podporováno: vlastnosti písem, atributy textu, zarovnání textu, obrázky, barvy textu a pozadí, mezery mezi slovy, písmena a linky, vnější odsazení, rámečky, vnitřní odsazení, pozicování a jednotná identifikace a klasifikace skupin atributů [4]. Samotné pozicování však bylo řešeno zvlášť o něco později v podobě přídavku k CSS 1 nazvaném CSS-P. Dále stanovilo základní používání selektorů pro určení elementů v dokumentu za pomoci tříd a identifikátorů.

Po tomto vydání vytvořilo W3C konsorcium oddělení Style Sheets and Formatting Properties Working Group, které se zaměřilo výhradně na CSS [4].

2.4.2 CSS Level 2

Roku 1998, W3C konsorcium vydalo CSS Level 2, které přidalo nové možnosti včetně z-index, typy médií, obousměrný text, absolutní, relativní a fixní pozicování a také podporu pro zvukové styly [4]. Rozšířilo vlastnosti pro formátování písma a možnosti dalších stávajících funkcí. Různé typy médií poté umožní odlišit definici stylů pro obrazovky, tisk nebo třeba mobilní zařízení či projektory.

Novinkou bylo zavedení možnosti použití CSS na jiné značkovací jazyky než jen HTML, jako například XHTML nebo hojně využívané XML.

2.4.3 CSS Level 2 Revision 1

Bohužel specifikace CSS Level 2 obsahovala spoustu chyb, a tak muselo dojít k její revizi. To vedlo k vytvoření CSS Level 2 Revision 1, tedy zkráceně CSS 2.1, které odstranilo nežádoucí chyby předchozí verze. Zároveň do specifikace nově zahrnulo vlastnosti, které prohlížeče běžně implementovaly, ačkoliv nebyly součástí doporučení W3C konsorcia. Standardizace této revize se potýkala s problémy, a proto k prohlášení za plně dokončenou verzi došlo až roku 2011.

2.4.4 CSS Level 3

Nejnovějším a aktuálně používaným standardem je CSS Level 3. První návrhy této verze byly zveřejněny roku 1999, krátce po dokončení CSS 2. Způsob, jakým je CSS 3 vyvíjeno, představuje zásadní změnu. Jednotlivé vlastnosti byly rozděleny do samostatných modulů, které jsou vyvíjeny nezávisle na ostatních. Standard už není spravován jako velký celek, ale v podobě částí zvaných moduly. Každý modul poté musí projít schvalovacím procesem zvlášť, a může se tak vydat jako konečné doporučení nezávisle na ostatních modulech, které jsou například ve fázi prvotního návrhu. Tato změna byla zavedena jako řešení problému, kdy specifikace a její následná implementace, jež byly vytvářeny a vydávány jako celek, nestíhaly reagovat na rychlý vývoj webu a jeho potřeby.

Mezi lety 2011 a 2012 byly následující moduly vydány jako oficiální doporučení: barvy, selektory úrovně 3, jmenné prostory a *„media queries“* [4]. Další moduly dosáhly stavu pracovního návrhu nebo kandidáta na doporučení pro fázi vydání a

jsou považovány za stabilní [4]. Moduly buď navazují na stávající vlastnosti a rozšiřují jejich možnosti, nebo vznikají moduly nové. Doposud vzniklo několik nových modulů, mezi které patří například zaoblené rohy, stíny, animace či 2D/3D transformace.

Protože CSS Level 3 bylo rozděleno na moduly, nedojde k žádnému vydání CSS 4 [4]. Vydané moduly pod označením Level 4 vychází z předchozích specifikací funkcí dostupných jako Level 3. Moduly budou nadále průběžně vydávány za účelem přidávání a aktualizování vlastností CSS [4]. Pojmenování CSS 3 tedy momentálně odkazuje na možnosti CSS jako celku, bez ohledu na rozdílné verze jednotlivých modulů.

2.5 Nedostatky CSS

Vývoj CSS nabral směr, který zapříčinil vznik několika nepříjemných nedostatků jazyka, které stále nemají tendenci se zlepšovat, spíše naopak. Právě odstranění těchto nedostatků, za účelem efektivnější práce s CSS, je jedním z mála důvodů pro používání CSS preprocesorů.

2.5.1 Nekonzistence implementace CSS

Už od samotného počátku vývoje CSS je jeho hlavním nedostatkem podpora webovými prohlížeči. K interpretaci CSS vlastností dochází až v samotném prohlížeči uživatele, a proto je CSS závislé na podpoře konkrétního prohlížeče.

Microsoft Internet Explorer 3, vydaný roku 1996, byl prvním komerčním prohlížečem, který podporoval CSS. V tento okamžik se specifikace CSS 1 doposud nestala doporučením W3C konsorcia a diskuse v rámci HTML ERB měly za následek změny, které vývojáři společnosti Microsoft nemohli předvídat [5]. Internet Explorer 3 spolehlivě podporuje většinu vlastností jako barvy, pozadí, písmo a text, ale neimplementuje většinu funkce zvané box model [5].

S výrazně lepší podporou se kaskádové styly začaly postupně objevovat ve čtvrté verzi prohlížečů Internet Explorer a Netscape Navigator, který byl roku 1998 druhým prohlížečem, jenž oznámil podporu CSS. Implementace Netscape podporuje

širokou škálu funkcí, například plovoucí prvky, ale vývojáři neměli čas na plné testování všech funkcí, které údajně podporují [5].

Třetím prohlížečem, který se odvážil pracovat s CSS, byla Opera [5]. Malý a přizpůsobitelný prohlížeč, který podporoval většinu funkcí, nalezených ve větších nabídkách společností Microsoft a Netscape [5]. Opera 3.5 byla vydána roku 1998 a podporuje většinu vlastností CSS 1 [5].

Následně průběžně prohlížeče představovali nové verze, které ovšem stále trpí stejným problémem. Každý z nich interpretuje jednotlivé CSS vlastnosti v rámci doporučené specifikace W3C konsorciem různým způsobem. V horším případě některé neaplikuje vůbec, nebo naopak zavádí vlastní, které nejsou ve specifikaci obsaženy. Někteří dokonce zcela mění samotné jádro zpracování kaskádových stylů, čímž sice napravují spoustu omezení, ale zároveň nová omezení a chyby vznikají. Vydání nového standardu CSS 2 přináší spoustu dalších vlastností a situace se tak nezlepšuje. Žádný z prohlížečů pochopitelně nepodporuje vše z nové specifikace, nicméně se snaží podporu rozšiřovat.

Tato nekonzistentnost mezi prohlížeči představuje problém při vývoji webových stránek s použitím CSS. Jejich tvůrci jsou tak například nuceni definovat některé části CSS odlišně pro vícero prohlížečů v rámci docílení stejného efektu, nebo používat různé nepraktické triky k docílení identického výsledku skrze různé prohlížeče či jejich verze.

2.5.2 Spravovatelnost a udržitelnost CSS

Dnešní situace je již o mnoho přijatelnější a prakticky už není potřeba rozlišovat většinu CSS vlastností mezi prohlížeči. Moderní webové prohlížeče odstranily převážnou většinu chyb interpretace CSS a snaží se velmi rychle zavádět nové vlastnosti dle vydávaných specifikací W3C konsorciem. Tomu částečně napomáhá systém vydávání CSS 3 pomocí jednotlivých modulů, jelikož se mohou vývojáři soustředit na postupné implementování vlastností tak, jak průběžně vycházejí jako doporučené.

Mohlo by se zdát, že použití CSS bez neduh ošemetných implementací prohlížečů, dělá z kaskádových stylů bezproblémový nástroj. Bohužel vývoj CSS znamená

převážně pouze přidávání nových vlastností použitých k určení prezentaci HTML elementů. Nicméně posun takovým směrem ne zcela odráží prostředí webu a jeho tvorby, které se pochopitelně taktéž vyvíjí. Možnost nových vlastností pro definování spousty různorodých vzhledů už není jedinou potřebou webových kodérů. Dnes se objevují rozsáhlé webové aplikace s mnoha složitými a velkými styly pro spousty různých zařízení [6]. Jejich správa přerůstá nejen možnosti direktivy import, ale i trpělivost kodérů, kteří se někdy místo samotné práce musí věnovat spíše otrockému kopírování, přepisování CSS deklarací a psaní složitých dokumentačních komentářů [6].

Problém tedy nastává v těžkopádné spravovatelnosti a udržovatelnosti kódu. Často je možné se setkat s problémem, kdy jedna deklarace vlastnosti přepisuje jinou. Dále je běžné například použití celých stejných deklarací či stejných hodnot (typicky barvy nebo písma) vlastností pro různé elementy, nebo rozdělení CSS do jednotlivých celků, uložených v samostatných souborech. Proto jazyk CSS vyžaduje při jeho psaní jakousi disciplínu. Samotná syntaxe je jednoduchá, ale logika aplikování stylů může být v případě komplexnějších webových stránek obtížná.

Novější specifikace se sice částečně snaží tento problém usnadnit, ale jedná se spíše o nové dílčí funkce, které nedostačují dnešním potřebám. Z těchto důvodů vznikají kompilované jazyky zvané CSS preprocesory, které zmíněné problémy efektivně řeší. CSS je jazyk čistě deklarativní, což se sice pro popis vzhledu webové stránky náramně hodí, ale zároveň nás to omezuje ve znovupoužitelnosti a v praktikování přístupu označovaného jako DRY („neopakuj se“) [6]. Nově navržené jazyky, jako LESS, proto do stylů přináší něco málo z imperativního a objektového programování [6]. Především ona objektovost rapidně zvyšuje možnosti správy a udržovatelnosti kódu [6].

3 CSS preprocesory

Kaskádové styly (CSS) jsou takzvaným stylovacím jazykem, který se používá pro definování vzhledu dokumentů strukturovaných pomocí značkovacích jazyků. Momentálně se tento jazyk jako jediný hojně využívá v souvislosti s vytvářením vizuálního stylu webových stránek, nicméně další oblasti využití při vytváření uživatelských rozhraní se objevují stále častěji. Bohužel vývoj CSS ne zcela reflektoval pokrok a potřeby samotného používání jazyků kodéry a má tak z historických důvodů jisté neduhy, které jsou výraznou překážkou při vytváření moderních aplikací v profesionálním prostředí.

Do dnešního dne nabízí CSS skrze své specifikace velkou spoustu různorodých vlastností, ale některé funkce zřetelně chybí. Nedostatky CSS řeší právě nástroj zvaný CSS preprocesor, který byl představen jako kompilovaný jazyk rozšiřující čisté CSS o podporu tradičních programovacích konstruktů, jako jsou například proměnné a funkce. Těchto preprocesorů je celá řada, všechny však mají za cíl co nejvíce zefektivnit práci s CSS.

3.1 Důvod vzniku CSS preprocesorů

Přestože se první CSS preprocesory začaly objevovat už roku 2006, jejich skutečné využití se rozšířilo až kolem roku 2010, kdy se několik z nich velmi rychle stalo oblíbenými. Od té doby je potřeba CSS preprocesorů stále aktuální téma, a to díky rostoucím možnostem, kde všude lze CSS použít, a především díky rostoucí komplexnosti práce při vytváření webových stránek a aplikací obecně.

Prakticky nedávno se CSS začalo používat jak v designu desktopových aplikací (např. pomocí knihovny WinJS), tak obdobně v mobilních aplikacích (např. prostřednictvím frameworku PhoneGap), což rozšiřuje jeho použití v širokém spektru aplikačních domén [7]. Právě to je důvodem, proč budou CSS preprocesory stále více důležité a užitečné i do budoucna. Pokud by před několika lety nedošlo ke vzniku CSS preprocesorů, pravděpodobně by k jejich vzniku – kvůli aktuálně nastupujícímu trendu používání CSS nejen na webových stránkách – zákonitě došlo právě teď, jakožto potřeba kodérů rozšířit ono CSS o chybějící klíčové funkce. A

tímto je dána hlavní podstata vzniku CSS preprocesorů, kterou zapříčinilo samo CSS ve vztahu k potřebám kodérů při jejich práci.

Zápis CSS pravidla se skládá pouze ze selektoru a přidružené deklarace, určující konkrétní vlastnost, která bude pro daný selektor použita. Jinými slovy, stačí pouze určit jeden či více prvků právě stylovaného dokumentu a deklarovat vlastnosti, které budou pro tyto určené prvky platit. Syntaxe CSS je tedy velmi jednoduchá a intuitivní, což má své důvody.

Zpočátku bylo CSS navrženo pro webové designery s minimálními zkušenostmi s programováním [7]. V důsledku toho postrádá mnoho základních programovacích konstrukcí, jako jsou proměnné, funkce, cykly a podmínky, které umožňují opětovné použití kódu a strukturované programování [7]. Přímý důsledek tohoto nedostatku programovacích funkcí je, že CSS kodéři jsou náchylní ke kopírování deklarací stylů z jednoho selektoru na jiný (tzv. klonování kódu) [7]. Proto se spravovatelnost a udržitelnost CSS kódů může v případě větších a komplexnějších aplikací stát v průběhu času otravnou až nezvladatelnou.

Jelikož jsou tvůrci nuceni dělat spoustu věcí zdlouhavě a opakovaně, velmi rychle nastane v CSS chaos, což způsobuje snížení orientace v kódů a jeho nepřehlednost. Pokud je navíc CSS používáno ve více lidech, kteří v rámci týmu pracují na společném projektu, stává se špatná práce s kódem dvojnásobným problémem.

Zde přicházejí na řadu právě CSS preprocesory, které byly představeny jako reakce na chybějící funkce CSS. Prakticky napravují tři podstatné výše zmíněné nedostatky CSS, jimiž jsou absence typických programovacích konstrukcí, obtížná udržovatelnost kódu a jeho přílišná opakovatelnost. Jednoduše zavádí spoustu nových možností, které potřebovali vývojáři oproti klasickému CSS navíc. Výsledkem použití CSS preprocesorů je co největší efektivita práce s CSS.

3.2 Jak funguje CSS preprocesor

Jakožto nástroj pro určení prezentace strukturovaných dokumentů, tedy jejich vzhledu a uspořádání, se řadí CSS preprocesory do takzvaných stylovacích jazyků, respektive dle anglického označení mezi „style sheet languages“. Zároveň ale přinášejí do CSS spoustu funkcí z jazyků typicky programovacích, což klasické CSS

nemůže jako jazyk čistě deklarativní nabídnout, a proto CSS preprocesory fungují na bázi skriptovacích jazyků, anglicky označovaných jako „scripting languages“.

Konkrétní CSS preprocesor tedy definuje svůj vlastní jazyk postavený nad CSS, respektive jeho syntaxi, a zároveň tak přidává právě onu možnost skriptování pomocí programovacích konstruktů, jako například proměnné, podmínky či poskytnutí řady užitečných funkcí. Často je syntaxe jazyka CSS preprocesorů velmi podobná a v některých případech skoro i totožná se zápisem čistého CSS, což z CSS preprocesorů dělá intuitivní rozšíření.

Nicméně CSS je vždy aplikováno přímo v prohlížeči na straně klienta, při načítání a sestavování daného dokumentu. Avšak prohlížeče implementují zobrazování vizuálních stylů webových stránek pouze pomocí zpracování běžných CSS a jeho vlastností definovaných standardy konsorcia W3C jako stromovou strukturu v kombinaci se strukturou DOM (Document Object Model). To znamená, že na vstupu očekávají zápis klasického CSS, nikoliv jazyka CSS preprocesorů, se kterým si neumí poradit.

Při využívání CSS preprocesoru je nutné vždy jeho výsledný kód převést na odpovídající CSS, které prohlížeč zpracuje obvyklým způsobem. To za kodéra udělá takzvaný překladač, který je jakýmsi jádrem preprocesoru, tedy jeho hlavní podstatou. Překladač preprocesoru tedy jednoduše vezme kód zapsaný ve svém skriptovacím kompilovaném jazyce a náležitě jej převede do obyčejného CSS.

3.3 Odlišnosti CSS preprocesorů

Vylepšení CSS o pár dalších funkcí a přidání základních programových konstruktů se může zdát jako jednoznačná věc. Přesto existuje CSS preprocesorů celá řada a každý z nich to dělá tak trochu po svém. Vzájemně se liší nemalým výčtem vlastností, které mohou být důvodem komplikovaného vybírání té nejvhodnější varianty.

3.3.1 Syntaktická stránka jazyka

Nejviditelnější odlišností při srovnávání CSS preprocesorů je pochopitelně syntaxe jejich vlastního preprocesorového jazyka, která je zpravidla rozdílná, avšak občas naopak v některých částech velice podobná. Zápis jazyka se může lišit například

v definovaných předpisech a názvech pro jednotlivé dílčí funkce, nebo v nutnosti psát v průběhu kódu závorky či středníky.

Syntaxe může dokonce fungovat i na principu takzvané nadmnožiny CSS, která stále přidává nové možnosti, ale jakýkoliv zápis i obyčejného CSS je automaticky korektním zápisem právě v daném jazyce preprocesorovém.

Malou zvláštností pak může být, že některé CSS preprocesory nabízí k použití více než jednu syntaxi svého jazyka. Pří výběru tak rozhoduje především osobní preference kodéra na vyhovující styl zápisu jazyka.

3.3.2 Poskytované funkce

Dalším, pravděpodobně očekávaným rozdílem mezi CSS preprocesory, je množství funkcionalit, které pro práci s CSS poskytují. Některé z nich mohou skutečně nabízet oproti jiným alternativám podstatně méně možností. To může být způsobeno délkou vývoje konkrétního CSS preprocesoru. Zejména ty nejstarší obvykle čítají nejvíce funkcí, na rozdíl od těch novějších, které ale na druhou stranu často nahrazují tento malý nedostatek větším důrazem na uživatelskou přívětivost.

Obecně ale platí, že aktuálně ty nejpoužívanější CSS preprocesory obsahují všechny klíčové vlastnosti, jako jsou například proměnné, vnořování kódu a jeho znovu použitelné bloky, podmínky, cykly a další. Tudíž při rozhodování mohou rozhodnout opravdu specifické možnosti navíc.

3.3.3 Implementace překladače

Poměrně nevyhnutelným rozdílem, na který lze narazit hned při samotné instalaci, je fakt, v jakém jazyce byl napsán překladač CSS preprocesoru. Každý kód zapsaný v určitém preprocesorovém jazyce je následně nutné převést pomocí takzvaného překladače do běžného CSS, které zvládne prohlížeč interpretovat. Nicméně tento překladač může být jeho autorem vytvořen téměř v libovolném programovacím jazyce. Od toho se odvíjí, jakým způsobem, respektive pomocí jakých technologií, se provádí právě onen překlad v průběhu používání CSS preprocesoru. Více o jednotlivých možnostech překladu se věnuje kapitola 5.4.

3.3.4 Méně výrazné rozdíly

Při srovnávání jednotlivých CSS preprocesorů je možné dále narazit na další drobnější rozdíly, které ovšem mohou být taktéž v některých případech v rámci srovnávání důležité. Jedním z nich je například podpora nejnovějšího standardu CSS 3, který je rozhodně nezbytnou součástí moderních aplikací a při jeho absenci se jedná o velký nedostatek. Nejpoužívanějším CSS preprocesorům však tato možnost samozřejmě nechybí, avšak u některých lze na tuto nedokonalost narazit.

Stejně tak důležitým prvkem je způsob, jakým CSS preprocesor umožňuje odlaďování chyb, tedy do jaké míry jsou užitečná a propracovaná jeho chybová hlášení, která nám pomohou vyřešit nedostatky v našem kódu. V neposlední řadě mohou být odlišnosti i ve formátu výstupu překladače, popřípadě spíše v počtu výsledných formátů CSS, do kterých lze preprocesorový jazyk převést. Do těchto formátů patří především různé možnosti textového uspořádání převedeného CSS.

Nakonec je určitě dobré brát v potaz, jakou uživatelskou podporu CSS preprocesor nabízí, přívětivost a rozsah jeho dokumentace či zda má stále aktivní komunitu a rozvíjí se.

3.4 Možnosti překladu do CSS

Preprocesor v kontextu CSS je ve své podstatě pouze jednoduchý transformační nástroj, který převezme kód ve formě preprocesorového jazyka a převede jej do CSS, které je následně interpretováno prohlížečem. Pokud je z nekompilovaného jazyka udělán najednou jazyk překládaný, vyvstává tak otázka, kam do životního cyklu webové aplikace umístit onen překlad [6].

Překladu se dá docílit několika způsoby a na různých místech, nicméně záleží to především na tom, v jakém programovacím jazyce byl vytvořen samotný překladač CSS preprocesoru. Jelikož právě ten zajišťuje zmíněnou transformaci kódu do CSS, překlad lze uskutečnit pouze tam, kde běží sám překladač, což je závislé na jazyku, ve kterém byl napsán.

Existují tři různé varianty překladačů CSS preprocesorů, které fungují v odlišných běhových prostředích. Prvním z nich je překladač postavený na interpretovaném

jazyce, který je spuštěný na straně klienta, tedy například přímo ve webovém prohlížeči. Jako takový jazyk je nejčastěji využíván JavaScript.

Druhým případem je překladač vytvořený v jazyce, který je přesně naopak spouštěn v rámci serveru, tedy na zařízení, které poskytuje služby klientům. Velmi často je k tomu volen jazyk PHP.

Poslední možností je podoba překladače jako klasického programu, vytvořeného například v jazyce Ruby, který si uživatel instaluje přímo na pracovní stanici. Samozřejmě ne všechny CSS preprocesory jsou k dispozici ve všech výše uvedených formách, ale z důvodu možnosti výběru typu překladu jsou často z původního formátu navíc předělané i do těch dalších. To ale není pochopitelně samozřejmost.

3.4.1 Překlad v prohlížeči

Pokud má použitý nástroj k dispozici interpretovaný překladač v JavaScriptu, je možné jej připojit ke stránce v hlavičce jako běžný skript a rovnou poslat web do ostrého provozu [6]. To je sice velmi snadné a rychlé, ale lepší představou je fungování stylů bez zapnutého JavaScriptu a lepší kontrola nad možnými chybami [6]. Používání této varianty tak určitě není nejlepší volbou přímo v produkčním nasazení. Naopak při samotném vývoji webové stránky najde tento způsob své využití. Při každé změně kódu v preprocesorovém jazyce dojde automaticky na straně klienta znovu k překladu do CSS a změny se projeví okamžitě, což může do jisté míry sloužit jako jistá automatizace spouštění překladu.

3.4.2 Překlad na serveru

Druhou možností je využít překladač napsaný v nějakém serverovém jazyce [6]. Pokud web stejně poběží na Djangu, Ruby on Rails nebo Nette, nabízí se vyhledat preprocesor CSS implementovaný v Pythonu, Ruby nebo PHP a začlenit jej přímo do projektu [6]. To sice umožňuje větší vládu nad stylem, například přidávání CSS pravidel dynamicky za běhu, nicméně nalezení vhodného překladače pro konkrétní jazyk není leckdy jednoduché [6].

Právě díky uvedeným jazykům mohou být CSS preprocesory integrovány do různých redakčních systémů a frameworků, což je pro vývojáře výhodné, jelikož k překladu není nutné používat další nástroje a zavádět tak další vrstvu komplexity.

Ve spojitosti s předchozí uvedenou variantou překladače, byl zmíněný JavaScript, považován především jako jazyk klientský, nikoliv serverový. Avšak zásluhou několika běhových prostředí (například Node.js) lze JavaScript spouštět i na serverech, což je další používanou metodou překladu preprocesorových jazyků.

3.4.3 Překlad na klientu

Asi nejlepší možností je nainstalovat si do svého počítače přímo původní kompilátor jako běžný program a využívat jej k překladu stejně jako v případě jakéhokoliv jiného jazyka [6]. Takovým programem může být opět klidně interpret jazyka Ruby, nebo Node.js pro využití JavaScriptu. Na ostrou verzi webu potom samozřejmě kopírujeme jen výsledné CSS styly [6].

Pokud je zvážen fakt, že původní překladač většinou obsahuje na rozdíl od vedlejších možností všechny prvky, včetně těch nejnovějších, jeví se tato možnost jako nejvýhodnější.

U diskuzí věnujících se CSS preprocesorům panuje často obava, že k jejich použití je nutný webový server podporující konkrétní programovací jazyk. To je samozřejmě jedna z možností, nicméně instalace daného jazyka na lokální počítač v podobě běžného programu je další a jednodušší možnost se stejným výsledkem.

3.4.4 Rychlost překladu

Různé způsoby překladu jazyka CSS preprocesoru do CSS jsou zásadním rozdílem, který může mít vliv nejen na přívětivost využívaných pracovních nástrojů, ale i na rychlost onoho překládání. U menších projektů nezpozorujeme žádný zásadní rozdíl, který by nás mohl nějakým způsobem omezit. Ale v případě vývoje rozsáhlých a komplexních webových aplikací, čítajících až několik tisíc řádků kódu, které je potřeba přeložit, může dojít mezi CSS preprocesory k opravdu viditelným rozdílům v rychlosti zpracování. Tyto rozdíly mohou dosáhnout až několika jednotek či desítek sekund, kdy jeden CSS preprocesor přeloží stejně obsáhlý kód

rychleji než CSS preprocesor jiný. Pokud je nutné spouštět překlad opakovaně, může se jednat o značnou nevýhodu.

3.4.5 Spouštění překladu

Překladač CSS preprocesoru může být napsán v různých programovacích jazycích a spouštěn buď na straně klienta, serveru, nebo jednoduše přímo na počítači kodéra. Samotný překlad je však potřeba nějakým způsobem spustit, když je to potřeba. V tomto ohledu CSS preprocesory nabízí několik základních možností.

Prakticky nejpoužívanější metodou je práce s příkazovou řádkou, kde je pomocí stanovených příkazů možné CSS preprocesor ovládat, včetně spuštění překladu. Méně častou variantou je pak uplatnění speciálního grafického uživatelského rozhraní, skrze které taktéž lze provádět překlad. Vhodnou volbou pak může dále být i rozšíření vývojového prostředí (IDE) o práci s CSS preprocesorem v podobě přídavného pluginu. Ne všechny CSS preprocesory samozřejmě nabízejí všechny uvedené možnosti, ale zmíněná příkazová řádka nechybí snad u žádného z nich.

Spuštění překladu je ve většině případech velice jednoduché. Pokud je však nutné v průběhu používání CSS preprocesoru sledovat výsledek, musí překlad z preprocesorového jazyka do CSS proběhnout při jeho každé úpravě. Zahájit překlad pokaždé ručně by bylo značně nepraktické. Z tohoto důvodu se často využívá automatizace překladu. V lepším případě tuto možnost nabízí přímo samotný CSS preprocesor v podobě speciální funkce, při jejímž zapnutí hlídá soubory s uloženým preprocesorovým kódem a ve chvíli, kdy dojde k jeho sebemenší změně a jejich následnému uložení, automaticky spustí překlad. I když tato funkce není k dispozici, stejného efektu se dá snadno docílit i pomocí takzvaných automatizačních nástrojů, jako například Gulp a Grunt, určených pro spouštění pravidelně se opakujících úloh při vývoji frontendové části aplikace.

3.5 Instalace CSS preprocesoru

Před použitím CSS preprocesoru je pochopitelně nutná jeho instalace. Ta se sice liší v závislosti na konkrétním použitém nástroji, ale obecně je těchto způsobů instalace jen pár. Tyto způsoby se odvíjí od zpracování překladače CSS preprocesory, tedy

v jakém programovacím jazyce byl vytvořen. V předchozí kapitole 5.4 jsou popsány možnosti běhu překladače v různých prostředích právě ve vztahu ke zpracování. Jeho instalaci je tedy nutné provést přesně v daném prostředí, ve kterém pracuje.

Je-li překladač zprostředkován v podobě interpreta fungujícího v jazyce JavaScript, pak jako instalace postačí jeho pouhé připojení jako obyčejného skriptu k dokumentu, jenž CSS preprocesor upravuje.

Mnohem používanějším prostředím je poté případ překladače běžícím na straně serveru, tedy postaveném na některém ze serverových jazyků, jako je například PHP, Ruby, nebo dokonce i zmíněný JavaScript, který za pomoci platformy Nodej.js může taktéž bez problému operovat na serveru. V tento moment je pro instalaci nutné použít libovolnou formu linuxového serveru, na kterém je zprovozněn některý z uvedených jazyků, který je potřeba pro fungování překladače daného CSS preprocesoru. Dále už jen stačí konkrétní CSS preprocesor na serveru stáhnout a začít používat. K jeho stažení slouží různé distribuční kanály a správci balíčků programovacích jazyků, případně manažeři balíčků linuxových distribucí. Velmi oblíbenou formou stahování nejen CSS preprocesorů, se stal takzvaný Node Package Manager (NPM), který působí jako správce balíčků pro jazyk JavaScript.

Třetí možností je instalace CSS preprocesoru na klientský počítač. Jelikož jazyky jako Ruby a JavaScript (Node.js) jsou podporovány nejen na straně serveru, ale i v podobě běžných programů pro standardní operační systémy desktopových počítačů. Taková instalace se pak příliš neliší od předchozí situace se servery. Probíhá prakticky stejně s výjimkou použití jiných příslušných instalačních zdrojů.

Nejnovější objevující se metodou je použití programů třetích stran, které samostatně řeší jak instalaci CSS preprocesoru, tak jeho následné používání včetně několika dalších užitečných funkcionalit. To vše se snaží nabídnout uživateli v co nejsnadnější a nejpřívětivější podobě. Často i zároveň pro více než jeden CSS preprocesor v rámci jednoho jediného programu.

3.6 Přehled dostupných CSS preprocesorů

Historie CSS preprocesorů sahá až k roku 2006. Od té doby jich vzniklo nespočet, nicméně spoustu z nich už dnes na internetu nelze najít. Ty momentálně dostupné,

které lze bez problému vyhledat a začít používat, jsou uvedené v následující tabulce č. 1. Nejedná se o výčet všech existujících CSS preprocesorů, ale pouze několika z nich, kterým se na internetu dostává určité pozornosti, a to ať už té největší, nebo zdaleka menší. Hodnoty uvedené v tabulce č. 1 jsou platné ke dni 25. 1. 2018.

Název	Jazyk	Verze	Aktualizace	Popularita	Pokročilý
Less	JavaScript	3.0.0	6. 1. 2018	15 345	Ano
Sass	Ruby	3.5.4	23. 1. 2018	10 981	Ano
Stylus	JavaScript	0.54.5	21. 12. 2017	8 815	Ano
Myth	JavaScript	1.5.0	12. 7. 2015	4 389	Ne
Rework	JavaScript	1.0.1	1. 2. 2015	2 756	Ne
Garden	Clojure	1.3.3	7. 1. 2018	886	Ne
CSS Crush	PHP	3.0.0	1. 1. 2018	525	Ano
Pleeease	JavaScript	4.3.0	19. 12. 2017	460	Ne
Roole	JavaScript	0.9.0	24. 10. 2013	362	Ne
Stylis	JavaScript	3.4.8	11. 1. 2018	349	Ne
Clay	Haskell	0.10.1	17. 9. 2017	263	Ne
Spiffing	PHP	-	23. 6. 2016	225	Ne
Hitch	JavaScript	0.6.3	15. 2. 2014	176	Ne
Banana CSS	JavaScript	0.8.0	28. 8. 2016	142	Ne
Turbine	PHP	1.1.0	13. 6. 2013	141	Ano
Stylecow	JavaScript	7.3.2	20. 6. 2017	122	Ne
Scaffold CSS	PHP	-	9. 12. 2016	24	Ano
CSS PP	PHP	1.0.5	13. 2. 2013	20	Ne
CSS-On-Diet	Python	1.8.4	22. 5. 2015	9	Ne
Sly	JavaScript	0.0.3	26. 9. 2016	7	Ne
Switch CSS	Python	-	30. 7. 2015	-	Ne

Tabulka 1: Přehled dostupných CSS preprocesorů [vlastní zpracování]

Vysvětlivky k tabulce:

- **Jazyk** – Primární vývojový programovací jazyk.
- **Verze** – Nejnovější vydaná verze.
- **Aktualizace** – Datum poslední provedené aktualizace.
- **Popularita** – Počet unikátních hodnocení GitHub repositáře.
- **Pokročilý** – Podpora pokročilých funkcí (podmínky, cykly aj.).

Z tabulky přehledu CSS preprocesorů lze vyčíst několik zajímavostí. Jednoznačně nejčastěji používaným jazykem pro vývoj CSS preprocesoru je JavaScript. S méně než polovičním zastoupením, následuje jazyk PHP, a ještě o něco hůře je na tom Ruby, který je ovšem použit u mnohem častěji používaných zástupců CSS preprocesorů.

Dále je možno sledovat, že ze všech dvaadvaceti uvedených CSS preprocesorů bylo pouze osm z nich aktualizováno během posledního půl roku. Ostatní nebyly aktualizovány nejčastěji jeden až dva roky, což značí o jejich neaktivním vývoji.

Ne vše, co je populární, musí být nutně nejlepší. Ovšem u prvních třech uvedených CSS preprocesorů (Less, Sass a Stylus) tomu tak je. Jedná se aktuálně o to nejlepší, co mohou CSS preprocesory nabídnout, a tak si svou popularitu, která je o několik řádů vyšší než v případě ostatních, rozhodně zaslouží.

Pokud by se vybraly pouze ty CSS preprocesory, které ve svém základu nabízí možnost využití pokročilých funkcí, jako jsou především běžné programové konstrukce (podmínky, cykly, …), zůstal by jich pouhý zlomek. Je to z toho důvodu, že tyto funkce nemusí být považovány za klíčové, a tak se některé CSS preprocesory obejdou i bez nich, stejně jako jejich uživatelé.

3.7 Základní funkce CSS preprocesorů

Kaskádové styly (CSS) mohou být pomocí CSS preprocesorů rozšířené o spoustu více či méně užitečných možností, které samy nedokáží žádným nebo omezeným způsobem nabídnout. Ačkoliv se CSS preprocesory navzájem liší spoustou vlastností, typickým rozdílem, na který je v rámci jejich srovnávání často poukazováno, se stávají právě jednotlivé funkce, které lze využít. I přesto, že některé

CSS preprocesory těchto funkcí nabízí podstatně více, záleží mnohdy pouze na několika z nich, které se dají považovat za základní nutnou součást každého CSS preprocesoru. Tato kapitola se věnuje představení základního principu jednotlivých funkcí. Podrobnější popis chování funkcí je předmětem jejich porovnání napříč CSS preprocesory, kterému je určena kapitola č. 4.1.

3.7.1 Proměnné

Jedním ze základních programových konstruktů, které CSS preprocesory přináší, jsou proměnné. Ty fungují téměř stejně jako v jiných programovacích jazycích. O proměnných lze přemýšlet jako o způsobu ukládání informací, které se opakovaně používají po celou dobu vytváření stylopisu CSS [8]. Mohou ukládat věci jako barvy, písma, nebo libovolné CSS hodnoty, u kterých je pravděpodobné, že budou použity vícekrát [8]. Pokud dojde ke změně hodnoty konkrétní proměnné, tato nová hodnota se projeví na všech místech, kde je daná proměnná použita. Hodnoty proměnných jsou pak často definovány pouze na jediném místě, proto se dají svým způsobem považovat za konstanty.

Následující příklad č. 3 znázorňuje použití proměnné v CSS preprocesoru Sass. Je zde pomocí klíčového znaku dolaru ($) vytvořena jedna proměnná s názvem „primární barva“. Ta drží hodnotu zvolené barvy *„#433“* a je použita opakovaně pro vícero prvků, se selektory *„h1“* a *„a“* v rámci hodnoty vlastnosti *„color“*, určující barvu písma.

```
$primarni-barva: #433;
h1 {
      color: $primarni-barva;
}
a {
      color: $primarni-barva;
}
```

Příklad 3: Proměnná v CSS preprocesoru Sass [vlastní zpracování]

Jakmile je příklad č. 3 zpracován překladačem CSS preprocesoru, jako výsledek vzniká běžné CSS, znázorněné dále v příkladu č. 4. V průběhu zpracování vezme překladač hodnotu barvy *„#433“* definované proměnné nazvané „primární barva“ a

dosadí jí do všech výskytů dané proměnné. Poprvé u nadpisů selektoru *„h1“* a podruhé u odkazů selektoru *„a“*.

```
h1 {
      color: #433;
}
a {
      color: #433;
}
```

Příklad 4: Výsledné CSS po použití proměnné v CSS preprocesoru [vlastní zpracování]

3.7.2 Hnízdění

Pro označení schopnosti CSS preprocesoru využívat vnořená pravidla, se používá výraz hnízdění. Jedná se o možnost zapsat vnořený selektor pravidla CSS, jakožto potomka jakéhokoliv jiného selektoru, který bude působit jako rodičovský, za účelem snadnější specifikace dědičnosti. Vnořený selektor se pak nevztahuje k celému dokumentu, ale pouze k potomkům elementu, který je vybrán selektorem rodičovským. Při psaní kódu HTML je zřejmé, že má jasnou zanořující se vizuální hierarchii [8]. Na druhé straně, CSS žádnou nemá [8]. Hnízdění umožní zanořování CSS selektorů takovým způsobem, který odpovídá stejné vizuální hierarchii HTML [8].

V příkladu č. 5 je ukázáno použití zanoření CSS selektoru v CSS preprocesoru Sass. První selektor *„nav“* je zapsán běžným způsobem a přejímá definování vlastnosti *„background color“* s hodnotou *„#333“*, určující barvu pozadí. Nicméně další selektor *„a“* je již zapsán jako selektor vnořený uvnitř prvního selektoru *„nav“*. Ten dále pracuje s vlastností *„color“* s hodnotou *„#000“*, která stanovuje barvu písma.

```
nav {
      background-color: #333;
      a {
            color: #000;
      }
}
```

Příklad 5: Hnízdění v CSS preprocesoru Sass [vlastní zpracování]

Po přeložení tohoto zápisu z příkladu č. 5. překladačem CSS preprocesoru vznikne CSS, uvedené v příkladu č. 6. Při samotném překládání pak dojde k sestavení zápisu, který je typický při používání dědičností v CSS, tedy nikoliv vnořené, ale samostatné oddělené, respektive nevnořené selektory *„nav“* a *„nav a“* jako selektor využívající oné dědičnosti.

```
nav {
     background-color: #333;
}
nav a {
     color: #000;
}
```

Příklad 6: Výsledné CSS po použití hnízdění v CSS preprocesoru [vlastní zpracování]

3.7.3 Mixiny

Na podobném principu proměnných fungují takzvané mixiny. Ty však neuchovávají jednu konkrétní hodnotu, ale rovnou několik CSS vlastností s jejich hodnotami najednou. Mixin umožňuje vytvářet skupiny deklarací CSS, které je žádoucí používat opakovaně po celou dobu vytváření webových stránek [8].

Některé věci v CSS se píší trochu zdlouhavě a nudně, zvláště s příchodem specifikace CSS 3 a existence několika vendor prefixů [8]. Výskyt těchto předpon značí, že daná CSS vlastnost je v konkrétním vykreslovacím jádře prohlížeče implementována pouze experimentálně či neúplně vůči svému standardu a nemusí tak fungovat dle očekávání. V takovém případě je vhodné danou věc zapsat pouze jednou za pomoci vytvoření mixinu a dále se už pouze na tento hotový zápis odkazovat.

Mixiny mohou být navíc dokonce i parametrické, což značně napomůže jejich flexibilitě. Při použití parametru se pak dá říci, že se mixiny chovají jako typické funkce běžných programovacích jazyků, které vracejí sadu CSS vlastností, jejichž hodnoty jsou proměnlivé právě na základě vloženého parametru.

Příkladem č. 7 je užití možnosti mixin v CSS preprocesoru Sass, a to v rámci již zmíněného problému nutnosti psaní různých vendor prefixů. Pomocí klíčové fráze

„@mixin“ je zde docíleno vytvoření jednoho mixinu s názvem „rotace“. V závorkách následuje zavedení parametru jakožto proměnné s názvem „stupeň“.

Dále už je na řadě určení samotných CSS deklarací, které se budou opakovaně za pomoci mixinu používat. Zde se jedná o vlastnost *„transform rotate“*, určující otočení prvku o určitý počet stupňů, který je proměnlivý na základě zavedeného parametru „stupeň“.

Vytvořený mixin je použit u selektoru *„box“* podobně jako klasická CSS deklarace, začínající klíčovým slovem *„@include“* a pokračující jeho názvem. Jelikož je mixin vytvořen včetně parametru, je při jeho použití nezbytné předat i konkrétní hodnotu onoho parametru, tedy *„30deg“*.

```
@mixin rotace($stupen) {
      -webkit-transform: rotate($stupen);
         -moz-transform: rotate($stupen);
          -ms-transform: rotate($stupen);
              transform: rotate($stupen);
}
.box {
      @include rotace(30deg);
}
```

Příklad 7: Mixin v CSS preprocesoru Sass [vlastní zpracování]

Při zpracování mixinu překladačem CSS preprocesoru dojde k dosazení veškerých CSS definicí z místa jejího vytvoření do místa jejího použití a hodnoty CSS vlastností budou odpovídat konkrétní zvolené hodnotě dosazené za parametr. Z příkladu 5 tak po překladu vznikne CSS odpovídající příkladu č. 8.

```
.box {
      -webkit-transform: rotate(30deg);
         -moz-transform: rotate(30deg);
          -ms-transform: rotate(30deg);
              transform: rotate(30deg);
}
```

Příklad 8: Výsledné CSS po použití mixinu v CSS preprocesoru [vlastní zpracování]

3.7.4 Extend

Práce v CSS je velmi omezená, co se týče znovupoužitelnosti již jednou definovaných deklarací, které je potřeba identicky použít znovu na jiném místě. Poté často dochází k duplikování částí CSS a porušování přístupu zvaného DRY („neopakuj se“), což vede ke značnému snížení spravovatelnosti a udržitelnosti většího množství zápisu CSS. Tento problém je v CSS preprocesorech vyřešen právě prostřednictvím extend.

Jedná se o jednu z nejužitečnějších funkcí [8]. Pomocí direktivy extend lze sdílet sadu vlastností CSS z jednoho selektoru do druhého [8]. Skupina určených vlastností tak může být velmi snadno použita opakovaně, a to napomáhá v dodržování přístupu DRY. Takový zápis v preprocesorovém jazyce je tak oproti CSS mnohem elegantnější a čistší. Podobného efektu se může docílit i použitím již představené funkce mixinů, to však vede k duplicitnímu CSS na výstupu, tedy po překladu, a to je zcela jistě nežádoucí.

V příkladu č. 9 je zapsána běžná situace v CSS, kdy deklarační blok s vlastnostmi *„padding“* a *„border“*, včetně jejich hodnot *„5px“* a *„1px solid #000“*, připadá vícero selektorům, tedy *„zpráva“* a *„varovná zpráva“*, přičemž selektor *„varovná zpráva“* je dále navíc doplněn o vlastnost *„color“* s hodnotou *„#f00“* pro odlišení barvy písma.

Jedná se o standardní metodu CSS, jak alespoň částečně podpořit znovupoužitelnost a umožnit použití stejné sady vlastností pro více selektorů jejich výčtem u deklaračního bloku. Přestože tato technika funguje poměrně dobře, je náročná na udržování a v případě rozsáhlých zápisů CSS téměř nepoužitelná.

```
.zprava, .varovna-zprava {
     padding: 5px;
     border: 1px solid #000;
}
.varovna-zprava {
     color: #f00;
}
```

Příklad 9: Technika znovupoužití vlastností v CSS [vlastní zpracování]

Na druhé straně, v příkladu č. 10, je obdobný zápis s použitím možnosti extend v CSS preprocesoru Sass. Zde má každý ze dvou selektorů pouze jeden vlastní deklarační

blok vlastností. Následně je v rámci selektoru „varovná zpráva“ pomocí klíčového slova *„@extend“* určeno, aby byl rozšířen o selektor *„zpráva“*. To jednoduše znamená, že bude obsahovat všechny jeho CSS vlastnosti. Po přeložení této konstrukce překladačem CSS preprocesoru je výsledkem CSS odpovídající právě příkladu č. 9.

```
.zprava {
      padding: 5px;
      border: 1px solid #000;
}
.varovna-zprava {
      @extend .zprava;
      color: #f00;
}
```

Příklad 10: Extend v CSS preprocesoru Sass [vlastní zpracování]

3.7.5 Import

Podobně jako ostatní programovací jazyky, i CSS preprocesory by měly mít možnost rozčlenit psaní jejich kódu na více logicky uspořádaných dílů. Toho je docíleno pomocí funkce import. Je tedy tak možné celou práci v preprocesorovém jazyce rozdělit na menší části v podobě samostatných souborů, které se jednotlivě snadněji udržují. Tyto části jsou následně propojeny právě pomocí funkce import a ve výsledku tak tvoří jeden celek. Proto například všechny proměnné a mixiny zavedené v importovaném souboru budou dostupné i v druhém souboru, do kterého je tento soubor importován.

Ve své podstatě se jedná o pouhé vložení jedné části kódu na konkrétní místo v jiné části kódu. Typicky je umožněno vkládat nejen kód preprocesorového jazyka, ale i běžné CSS.

Následující tři příklady představují zkrácené využití importu v CSS preprocesoru Sass při rozdělení kódu na dvě části. Příklad č. 11 jako první reprezentuje část oddělenou do souboru *„reset.scss“*, která má za úkol vynulovat hodnoty specifických vlastností u několika vybraných elementů předtím, než se začnou definovat hodnoty vlastní.

V příkladu č. 12 je jakožto v rámci hlavního souboru *„base.scss“* již poté určeno několik libovolných vlastních CSS deklarací. Těm však musí předcházet nejprve nulové deklarace ze souboru předchozího. Toho je dosaženo právě importováním kódu ze souboru *„reset.scss“* na začátek kódu souboru *„base.scss“* pomocí klíčového slova *„@import“* s určením názvu vkládaného souboru. Jeho příponu *„scss“* není nutno uvádět.

```
// reset.scss
html, body, ul, ol {
      margin: 0;
      padding: 0;
}
```

Příklad 11: Soubor CSS s nulovými hodnotami vlastností [8]

```
// base.css
@import 'reset';
body {
      font: 100% Helvetica, sans-serif;
      background-color: #efefef;
}
```

Příklad 12: Import v CSS preprocesoru Sass [8]

V ukázce z příkladu č. 12 dojde jednoduše ke vložení kódu ze souboru určeného direktivou *„@import“* přesně na místo jejího zápisu. Následně proběhne samotný překlad CSS preprocesorem již spojených částí a v případě kombinace předchozích dvou příkladů tak bude výsledkem celistvé CSS znázorněné v příkladu č. 13.

```
html, body, ul, ol {
      margin: 0;
      padding: 0;
}
body {
      font: 100% Helvetica, sans-serif;
      background-color: #efefef;
}
```

Příklad 13: Výsledné CSS po použití funkce import v CSS preprocesoru [8]

3.7.6 Operátory

Nezbytnou součástí všech programovacích jazyků jsou bezpochyby operátory, umožňující provádění ať už základních či pokročilých operací s různými hodnotami. Jelikož CSS je jazyk čistě deklarativní, tato schopnost mu schází. Nicméně v CSS se s čísly pracuje téměř neustále, například v kontextu jednotek a barev, a proto zavedení operátorů do CSS prostřednictvím CSS preprocesorů byl logický krok. Své uplatnění nachází nejen v počítání komplexních layoutů a při práci s barvami.

Matematika je v CSS užitečná, a tudíž samozřejmostí je zavedení operátorů typů sčítání, odečítání, násobení, dělení a počty s procenty. Dále však mohou být praktické i například operátory unární, binární, ternární, logické, rozsahové, existenční a tak podobně.

Jednoduchou ukázkou použití operátorů v CSS preprocesoru Sass je příklad č. 14. Ten zavádí dvousloupcové rozložení webových stránek v podobě levého prostoru pro článek a pravého dodatečného sloupce, jejichž selektory jsou *„article“* a *„aside“*. Obě tyto části disponují nejen vlastností *„float“*, ale hlavně vlastností *„width“*, určující jejich šířku. Ovšem namísto přímého určení a vepsání hodnoty této šířky, je dynamicky spočítána na procenta pomocí operátorů relativně vzhledem k celkové šířce *„1200px“*.

```
article {
      float: left;
      width: 900px / 1200px * 100%;
}
aside {
      float: right;
      width: 300px / 1200px * 100%;
}
```

Příklad 14: Operátory v CSS preprocesoru Sass [vlastní zpracování]

V průběhu zpracování operátorů překladačem CSS preprocesoru dojde ke správnému propočtu číselných hodnot a vlastnosti *„width“* nabydou konkrétních hodnot. Výsledné CSS po překladu znázorňuje příklad č. 15.

```
article {
      float: left;
      width: 75%;
}
aside {
      float: right;
      width: 25%;
}
```

Příklad 15: Výsledné CSS po použití operátorů v CSS preprocesoru [vlastní zpracování]

3.8 Pokročilé funkce CSS preprocesorů

Popsané funkcionality jako jsou proměnné, hnízdění, mixiny, extend, import a operátory patří mezi základ, který nepostrádá téměř žádný CSS preprocesor. Existuje ovšem nespočet dalších, které se hojně využívají. Následující výčet představuje pár z nich, které jsou známé především z nejoblíbenějších CSS preprocesorů Sass, Less a Stylus. Patří mezi ně například:

- Práce s barvami – převody, vytváření, nasycení, zesvětlení, ztmavení, úpravy průhledností či odstínů, práce s RGB a HSL kanály apod.
- Práce s řetězci – kódování, formátování, escapování, převod malých či velkých písmen atd.
- Matematické funkce – převody úhlů sinus a cosinu, číslo PI, nalezení minima a maxima, zaokrouhlení, převody procent aj.
- Logické hodnoty – ověřování typů čísel, jednotek apod.
- Převody jednotek – délky, čas, úhly.
- Komentáře – víceřádkové i jednořádkové.
- Jmenné prostory – vytvoření vlastních jmenných prostorů.
- Práce s polem – vytváření, procházení, vkládání prvků, spojování apod.
- URL – převod souborů do formátu base64.
- Pokročilé programové konstrukty – podmínky, cykly, vlastní funkce

- Datové typy – řetězce, pole, mapy, barvy.
- Debugovací funkce – odlaďování chyb.
- Interpolace – určování názvu vlastnosti spojením řetězců.

3.9 Praktické použití základních funkcí

Základní funkce CSS preprocesorů představují jednoznačně popsané možnosti, které lze využít pro rozšíření práce s CSS, jenž nemůže tyto složitější praktiky samo nabídnout. Mají určený jejich zápis a odpovídající výsledek. Nicméně některé z nich se nemusí zprvu jevit jako přínosné. Ovšem zvýšení efektivity a kvality práce je jejich společným důvodem vzniku. Je tedy mnohem vhodnější se na tyto funkce podívat nejen z pohledu postupů použití, ale hlavně z pohledu jejich skutečných benefitů při práci s CSS. Tyto benefity lze obecně sloučit do několika příslušných kategorií, kdy každá z nich má za důsledek vylepšení jiné části pracovních postupů. Do jednotlivých kategorií se pak dají zařadit ty konkrétní základní funkce CSS preprocesorů, které svým použitím přispívají k podobným výhodám.

3.9.1 Organizace a spravovatelnost kódu

Jedním ze dvou nejčastěji používaných argumentů pro použití CSS preprocesorů je fakt, že mají možnost svůj kód oproti samotnému CSS o poznání lépe organizovat. S několika málo funkcionalitami navíc umožní vytvářet znatelně čitelnější a čistší kód. Z toho vyplývá i jeho snadnější udržování, upravování a rozšiřování z dlouhodobého hlediska.

V dnešní době, kdy se CSS používá pro rozsáhlé webové aplikace v profesionálním prostředí je dobrá organizace a spravovatelnost kódu důležitým nepostradatelným prvkem. Proto CSS preprocesory nabízí při nejmenším funkce jako hnízdění, extend a import, díky kterým se práce s CSS stává přehlednou a snadnější. Běžné CSS na to má ovšem taktéž své metody, bohužel většina z nich je nepraktických a ty novější nejsou podporovány ze strany prohlížečů bez problémů.

3.9.1.1 Hnízdění

Pravděpodobně druhým nejčastěji zmiňovaným rysem CSS preprocesorů je hnízdění. Jedná se o možnost vnoření selektoru do jiného selektoru a snadno tak

vidět a reprezentovat vztah mezi nimi v rámci dědičnosti. Na první pohled netypická záležitost pro CSS. Netypická však pouze zdánlivě.

Hnízdění není nový přístup – je to způsob, jakým je postaveno i HTML a pomáhá vývojářům vizualizovat hierarchii ve svém kódu [9]. Je to něco, co CSS postrádá, ale u CSS preprocesorů tomu tak není, což dělá z psaní kódu příjemnější praxi [9]. Při nezkušenosti s touto funkcí se může zprvu zdát jako zbytečná, nicméně opak je pravdou a po jejím pravidelném používání se bez ní stává práce s CSS nepředstavitelnou. Mimo to se dá hnízdění i vhodně kombinovat s různými způsoby pro psaní znovupoužitelného a lépe spravovatelného CSS kódu, jako jsou koncepty Object Oriented CSS (OOCSS) a BEM [10].

Jde o velmi užitečný, a hlavně snadno aplikovatelný nástroj pro správné organizování kódu a zvýšení jeho čitelnosti. Nicméně při jeho nesprávném použití může docílit opačného negativního efektu. Je třeba dbát na maximální hloubku zanoření CSS selektorů, která by se měla pohybovat kolem tří úrovní. V případě nadměrného zanořování dojde nejen ke snížení čitelnosti kódu, ale i ke snížení možností znovupoužitelnosti zanořených selektorů [11]. To stejné pak bude platit i pro výsledné CSS po přeložení preprocesorového kódu překladačem. Je tedy vhodné najít optimální míru zanořování selektorů tak, aby kód zůstal co nejvíce organizovaný, užitečný a zároveň jeho výstup čistý [11]. Příklad použití hnízdění je uveden v kapitole 5.7.2.

3.9.1.2 Extend

Další zajímavá funkce CSS preprocesorů, která pomáhá vyprodukovat kvalitnější kód v kratším čase. Extend funguje v zásadě na principu dědičnosti. Umožní sdílet sadu CSS deklarací vlastností z jednoho selektoru na jiný. Nicméně ve výsledku nedochází k duplikování oné sady mezi selektory, ale ke sloučení všech selektorů, které sadu sdílejí, do složeného selektoru, který používá jedinou původní sadu CSS deklarací. Tímto způsobem je zabráněno zbytečnému opakování stejných částí kódu a stává se tak opět mnohem více organizovaným a spravovatelnějším.

Aplikace může mít například řadu tlačítek [9]. Rozšířením některých vlastností pomocí extend budou mít všechna tlačítka stejný tvar, velikost, písmo a zaoblené

hrany [9]. Tlačítka „provést platbu“ mohou být zelená, tlačítka „přihlášení“ mohou být žlutá a tak dále, ale všechny sdílené vlastnosti budou nakódovány pouze jednou a zděděny, což ušetří vývojářům spoustu času [9].

Jelikož funkcionalita extend svým způsobem pracuje podobně jako import, mohla by stejně tak být zařazena do kategorie podporující úsporu a konzistenci kódu. Její možnost znovupoužití CSS deklarací však má přímý vliv právě i na udržení čitelnosti a organizace kódu. Zároveň takřka stejné metody lze docílit i pouhým CSS, bez nutnosti použití CSS preprocesoru. Ten však tomuto principu dodává oproti CSS právě onu jednoduchost a praktičnost celého zápisu, a přesně proto je extend uveden v této kategorii. Příklad konkrétního zápisu extend je popsán v kapitole 5.7.4.

3.9.1.3 Import

Určitě nejdůležitějším prvkem, jehož jediným cílem není nic jiného než právě co nejlepší možná organizace a spravovatelnost kódu v pravém slova smyslu, je takzvaný import.

Čím složitější se aplikace stane, tím větší se stává i její CSS, což vede ke komplexnějšímu kódu, který je těžké udržovat [9]. U středně velkých projektů může CSS kód běžně čítat i několik tisíc řádků v jednom jediném souboru a u těch ještě rozsáhlejších není problém přesáhnout hranici deseti tisíc řádků. Nepochybně se lze dostat i k větším číslům, nicméně tento počet řádků je dostatečně velký na to, aby v kódu nebyl žádný pořádek [12]. V takovém případě se už musí organizace kódu řešit opravdu nevyhnutelně, zvláště pokud má v budoucnu dále docházet k jeho rozšiřování a přidávání nových věcí, a proto spousta lidí považuje import za klíčovou funkci CSS preprocesorů a vůbec za pádný důvod jejich používaní [12].

Stejně jako dobré logické uspořádání složek projektu velmi napomáhá k jeho lepšímu udržování, je tomu tak i v případě CSS, protože ukládání CSS pravidel do jediného velkého souboru vede zaručeně k problémům [13]. To řeší funkce import možností rozdělit tento velký soubor do několika menších souborů, které budou lépe udržovatelné, pochopitelné a organizovatelné. Ty udrží větší soubory více modulární, ale nebudou přeloženy do CSS, protože nejsou kompletními soubory [9].

K jejich překladu dojde až v místě, kde jsou funkcí import vloženy do souboru překládaného. To navíc také zapříčiní zmenšení problémů při týmové práci více lidí nad jednou společnou webovou aplikací.

Rozdělení do jednotlivých souborů je důležité provést dle částí kódu, které spolu nějakým logickým způsobem souvisejí [13]. Zvolení této struktury se mění v závislosti na použitých prvcích v konkrétním projektu. Typicky například na webových stránkách lze od sebe oddělit části jako „hlavička“, „menu“, „obsah“ a „patička“. Stejně tak je dobré oddělit zvlášť i konkrétní definovatelné funkce CSS preprocesoru, jako jsou proměnné a mixiny. Tyto všechny části jsou pak v praxi nejčastěji vkládány pomocí funkce import do hlavního souboru, který z nich po jeho přeložení překladačem CSS preprocesoru vytváří celistvé CSS v podobě jediného výsledného souboru.

Není tajemstvím, že stejnojmennou funkci import nabízí od určité verze standardu i obyčejné CSS. Ta pracuje velmi podobně jako v případě CSS preprocesorů, ovšem potýká se s jedním zásadním nedostatkem, který z ní dělá v praxi v podstatě nepoužitelnou záležitost. Totiž že při každém jednom použití funkce import dochází k vytvoření nového HTTP požadavku za účelem získání importovaného souboru [12]. To znamená, že při rozdělení struktury CSS například do pěti souborů proběhne při jejich importu celých pět HTTP požadavků, což má nepopíratelně negativní vliv na výkon webových stránek a rychlost jejich načítání. Naopak díky tomu, že CSS preprocesor kompiluje (překládá) svůj vlastní jazyk do CSS, provede už před překladem nejprve pět příkladných importů dílčích souborů do jediného souboru, který následně přeloží opět do jediného souboru jako výsledné CSS. Pouze tento soubor je pak na webové stránce umístěn a stažen za pomoci jednoho HTTP požadavku, a to i přes to, že v rámci projektu je CSS rozděleno do libovolného počtu souborů.

Příklad jednoduchého použití import je popsán v kapitole 5.7.5. Zde je navíc v příkladu č. 16 ukázka typicky používaného rozdělení CSS na několik části a jejich import do společného souboru v CSS preprocesoru Sass.

```
/* promenne */
@import “barvy“
@import “typografie“
/* nastaveni */
@import “mixiny“
@import “reset“
/* rozlozeni */
@import “hlavicka“
@import “menu“
@import “obsah“
@import “paticka“
```

Příklad 16: Způsob použití funkce import v CSS preprocesoru Sass [vlastní zpracování]

3.9.2 Úspora času a konzistence kódu

Argument lepší organizace a spravovatelnosti kódu je bezpochyby dostačujícím pádným důvodem, proč CSS preprocesory používat. Veškeré benefity vyplývající z této možnosti, jako například snadnější čitelnost a úpravy kódu, zároveň automaticky vedou k vylepšení i důležitého faktoru rychlosti vývoje. Čas jsou peníze, a to v profesionálním prostředí platí dvojnásob. Proto způsob rychlé, ale zároveň kvalitní práce s CSS je v případě velkých projektů nemalou výhodou, které lze s pomocí CSS preprocesorů jednoduše dosáhnout.

Správná organizace pochopitelně ve výsledku čas šetří, nicméně CSS preprocesory nabízí řadu dalších funkcí, které výrazně napomáhají rychlejšímu psaní kódu a změnám v něm. Jedná se především o proměnné a mixiny, díky kterým lze ušetřit spoustu času při práci a docílit mnohem lepší konzistence kódu. Není tak potřeba psát spoustu stejného kódu stále pořád dokola, ale naopak lze dodržovat užitečný programovací princip zvaný DRY (Don’t Repeat Yourself), tedy neopakuj se. U samostatného CSS tomu tak bohužel není, a proto je naopak jeho zápis označován jako WET (Waste Everyone’s Time), tedy zápis ztrácející všem čas [14]. Jde tak o druhý ze dvou hlavních důvodů, podle kterého se používání CSS preprocesorů opravdu vyplatí.

3.9.2.1 Proměnné

Princip proměnných není žádnou novinkou, a v každém programovacím jazyce se jedná o jeden ze základních a nejdůležitějších konstruktů vůbec. V CSS, respektive v CSS preprocesorech, nejsou proměnné užitečné o nic méně. Jelikož se jedná o takto známou a stěžejní funkci, často začínají diskuze, dokumentace či návody právě představením proměnných.

Proměnné jsou způsob, který pomáhá omezit opakující se práci [9]. Jakákoliv hodnota v CSS – barva, písmo, řádkování – může být uložena v proměnné a pak znovu použita kdekoliv v celém souboru [9]. Jamile je soubor CSS preprocesoru kompilován do CSS, tyto hodnoty se automaticky zobrazí na každém daném místě v CSS řádek po řádku, aniž by se musely řádek po řádku kódovat [9]. Jednoduchým odkázáním na tyto hodnoty se zajistí jejich konzistence skrze celé CSS s vynaložením minimálního úsilí.

Přínos proměnných se velmi snadno ilustruje v souvislosti s jejich použitím u barev. Některé barvy mohou mít specifické role v designu [13]. Je běžnou praxí, že jedna konkrétní barva je zvolena jako primární a následně se opakovaně používá na webových stránkách pro vícero prvků, například jako stejná barva pro tlačítka, odkazy aj. Při práci s běžným CSS je nutné hodnotu této barvy zapsat přímo do CSS pravidla každého prvku, jemuž se barva nastavuje. Tento přístup funguje, nicméně lehce způsobí určité problémy. V momentě, kdy bude rozhodnuto o změně výběru samotné barvy, nezbývá nic jiného než její hodnotu přepsat v každém jednom jejím výskytu. Horším případem pak už může být jen změna více barev. Představa, že kód CSS má několik stovek až tisíc řádků sama vypovídá o nevhodnosti této metody.

Jelikož nám proměnné v CSS preprocesorech umožňují ukládat hodnoty, je vhodné si hodnotu oné barvy uložit právě jako pojmenovanou proměnnou, kterou lze následně kdekoliv opakovaně využít odkázáním na její název. Všechny konkrétní hodnoty barvy tedy budou v kódu nahrazeny jednou odpovídající proměnnou. Následně, kdy opět dojde na změnu barvy, stačí tentokrát upravit pouze její hodnotu v příslušné proměnné na jediném řádku a změna se po přeložení automaticky projeví na všech místech jejího použití. Konkrétní příklad použití barvy jako proměnné je znázorněn v kapitole 5.7.1.

3.9.2.2 Mixiny

Zbytečnému opakovanému vypisování stejného kódu CSS na různých místech, který dělá totéž, je vždy těžké se vyhnout. V některých případech je to pak bohužel i nemožné. Vyřešením tohoto problému by opět práce s CSS zaznamenala výraznou úsporu času a mnohonásobně lepší konzistenci kódu. Jako řešení CSS preprocesory přináší funkci zvanou mixiny, s níž se snadno lze vyhnout onomu nevyhovujícímu opakování CSS.

Jedná se o velmi silný nástroj, který může být na první pohled trochu matoucí, ale ve skutečnosti funguje podobně jako jednoduchá funkce v kterémkoliv jiném programovacím jazyce. Funkce samotné jsou programovacím konstruktem, jehož použitím dochází právě k možnosti znovupoužití kódu, který drží. Stejně tak je tomu i u mixin. Vytvořením mixin, mohou kodéři seskupovat CSS deklarace a opakovaně je používat v celém souboru, aniž by to vyžadovalo spoustu zdlouhavého kódování [9].

Vše, co pomůže psát méně kódu, je správně. Navíc obdobně jako běžné funkce, mohou i mixiny přejímat takzvané parametry, díky kterým lze daný mixin použít vícekrát s různými hodnotami pro jeho CSS vlastnosti. To dodává další jistou úroveň škálovatelnosti. Pokud zahrneme i fakt, že méně specifické mixiny se často mezi sebou nemusí lišit mezi různými projekty, nabízí se možnost je přenositelně využívat i v rámci jiných projektů, než pro které byly původně určeny, což jim znovu dodává další stupeň flexibility.

Typickým ilustračním příkladem na vhodné použití mixinu CSS preprocesorů jsou takzvané vendor prefixy. Jedná se o slovní předpony, které je nutné doplnit před název CSS vlastností, které nejsou prohlížeči plně podporovány. Takových vlastností naštěstí postupem času ubývá, ale pořád se jich spousta vyskytuje. Tato nešťastná metoda se navíc komplikuje tím, že každé vykreslovací jádro prohlížeče používá jiný název předpony, jako například *„webkit“*, *„moz“* či *„ms“*. Částečně podporovaná vlastnost se pak musí v každé použité deklaraci zapsat s předponami v několika variantách. To samozřejmě nejen zabere spoustu času zbytečně navíc, ale nekonzistence opakovaného zápisu předpon způsobí problém při nutnosti provádění změn.

V případě, že se jedna z dříve částečně podporovaných CSS vlastností později stane plně podporovanou a je tedy pak tak potřeba ve všech jejích výskytech smazat příslušnou zastaralou předponu. Bohužel není jiné cesty než řádek po řádku. V lepším případě je CSS vlastnost zapsána společně se všemi vendor prefixy jakožto vytvořený mixin, který je opakovaně používán namísto přímého zápisu oné CSS vlastnosti. Jeli třeba opět odstranit zmíněný vendor prefix, stane se tak na jediném místě a změna se automaticky promítne na všechna místa použití mixinu. Netřeba si tak ani pamatovat názvy předpon pro jednotlivá jádra prohlížečů.

Použití je velice jednoduché a výsledný kód je zase o něco čistší a udržitelnější. Provádění rozsáhlých změn je pak možné uskutečnit prakticky během několika chvil. Kromě toho vytváření samotných mixin ani neovlivní velikost výsledného souboru, jelikož jsou při překládání do klasického CSS odstraněny. Konkrétní příklad zápisu mixinu je popsán v kapitole 5.7.3.

3.9.3 Flexibilita

Hnízdění, extend, import, proměnné a mixiny. To je výčet funkcí, které stojí za úspěchem CSS preprocesorů a jsou hlavním důvodem jejich používání. Znatelně usnadňují práci s CSS díky lepší úspoře času, organizaci, spravovatelnosti a konzistenci kódu. Tvoří tak nejpodstatnější a nejpoužívanější část CSS preprocesorů. Ty však nabízí další skupinu funkcí, která napomáhá ještě více podpořit flexibilitu práce a možností. Ačkoliv méně často využívanou, tak stále užitečnou. Patří mezi ně například známé programové konstrukty, jako jsou podmínky a cykly.

Pomocí podmínek lze snadno podmiňovat části CSS kódu, které budou platit pouze za určité situace. Cykly poté umožní několikanásobné použití opět nějaké části kódu daným počtem jeho opakování. Stejně tak do této skupiny funkcí patří i operátory, jenž lze využít nejen k matematickým výpočtům různých jednotek při vytváření layoutů či modifikování barev.

3.9.4 Knihovny

Psát celkově méně kódu je vždy výhra, a proto jsou vývojáři zvyklí na nástroje jako knihovny, frameworky a API [9]. Pokud jde o CSS, není to jiné – CSS preprocesory jsou toho schopné pomocí funkcí jako například mixiny [9]. Sada vytvořených mixin,

jenž se v CSS kódu opakovaně používají, se dá považovat za jakousi knihovnu, která se pochopitelně dá v rámci bezproblémové přenositelnosti použít v několika nesouvisejících projektech. Nemusí se však jednat pouze o vlastní projekty, ani vlastní knihovny. Pokud je možnost repetitivně používat knihovny své, stejně tak to bude fungovat i s těmi cizími.

Pro potřeby opakujících se úkonů, je více než vhodné si vytvořit vlastní sadu mixinů, tedy specifickou knihovnu. To už udělala spousta lidí, kteří se ve svých knihovnách obdobně snaží vyřešit běžné problémy a definovat nejčastěji používané CSS prvky. Ve většině případech takovou knihovnu následně bezplatně sdílí mezi ostatní kodéry. Je pak opravdu velice snadné některou z knihoven vzít, jednoduše pomocí funkce import jí vložit do vlastního CSS, jakožto klasický soubor preprocesorového jazyka a v rámci zjednodušení práce začít používat to, co pro ostatní vytvořili jiní.

Samozřejmě lze i přesně naopak sdílet vlastní knihovnu. Ať už mezi spolupracovníky, tak pro všechny, kdo o ní mají zájem a nabídnout jim tak svá řešení. Mezi nejznámější zástupce patří frameworky Bootstrap, Foundation nebo knihovny CSS 3 mixinů Bourbon, Compass či LESShat.

4 Porovnání CSS preprocesorů

Přínos CSS preprocesorů je jednoznačný a samotné CSS nemůže ani dnes po několika letech vývoje nabídnou jim obdobné nástroje. Ačkoliv zlomek z nich už je ve skutečnosti v nějaké fázi vývoje, jejich momentální použití v produkčním prostředí není v žádném případě stoprocentní, ať už se to týká funkčnosti, praktičnosti nebo kompatibility v různých webových prohlížečích. Často stačí argument použití jednoduchých konstruktů, nicméně velmi užitečných, jako například proměnné či import a vzápětí se výhoda CSS preprocesorů nad obyčejným CSS stává nepopíratelnou. S rostoucí komplexností webových aplikací a jejich vývoje jsou pomocné nástroje jako CSS preprocesory čím dál tím více potřeba.

V konečném důsledku je jejich hlavní schopností ulehčení vývoje, tedy co možná největší efektivita pracovních postupů. To může být zásadním přínosem v případě profesního nasazení, kde je potřeba způsoby práce a její usnadnění řešit na denní bázi, a to často i v rámci spolupráce s více lidmi. Proto výběr správné technologie může ovlivnit celý proces vývoje a následně tak i jeho výsledek.

Tabulka č. 1, uvedená v kapitole č. 3.6, zobrazuje přehled téměř všech aktuálně dostupných CSS preprocesorů, které lze na internetu dohledat a bez problému začít používat. Je jich nemalé množství (celkově 21 zástupců). Rozhodnout se pro jejich používání nebývá zpravidla těžké, často pak ale hned poté nastává problém, který z nich vybrat.

Jakkoliv může být přínos CSS preprocesorů jasný, tak není neobvyklé, že právě neschopnost vybrat jeden z nich bývá paradoxně překážkou k jejich používání, a to i v případě omezení výběru na pouze první tři kandidáty ze zmíněné tabulky. Jednou z věcí, která brání nováčkům ve vyzkoušení kteréhokoliv CSS preprocesoru, je právě nerozhodnost ve výběru toho pravého zástupce, což je naprosto zbytečné, protože jejich základní použití je velmi snadno naučitelné [15].

Tato kapitola se věnuje porovnání tří konkrétních CSS preprocesorů. Rozdíly odhalené při jejich srovnávání pak lze považovat za vhodná vodítka při výběru jednoho z nich.

4.1 Představení zástupců porovnání

Takzvaný Sass je prvním CSS preprocesorem, který byl představen širší skupině vývojářů jako volně dostupný a stal se tak jakýmsi průkopníkem těchto nástrojů. Následně jeho myšlenka inspirovala ke vzniku jiného obdobného řešení s názvem Less. O něco později se objevil jistý Stylus, který na sebe upozornil svým novým netypickým konvenčním přístupem. V průběhu času bylo vytvořeno nespočet dalších principiálně totožných CSS preprocesorů. Spousta z nich nezaznamenala žádný větší zájem, případně z nějakého důvodu upadla jejich prvotní pozornost velmi rychle prakticky na nulu.

I přesto lze dnes najít více než 20 funkčních CSS preprocesorů, které se vybízejí k použití. Přehled všech 21 zástupců shrnuje tabulka č. 1 v kapitole č. 3.6, nicméně mezi třetím a čtvrtým místem je poměrně velký skok co se popularity týče. Naopak první tři místa jsou vzájemně téměř vyrovnaná a dá se tak říci, že právě ty lze považovat za nejpopulárnější CSS preprocesory.

Ne náhodou mezi první trojici patří již na začátku zmínění Sass, Less a Stylus. Ty se uchytily u vývojářů zdaleka nejvíce. Je to dáno především díky počtu jejich nabízených funkcí a přívětivosti jejich používání. Ve skutečnosti se od samého počátku rozmachu CSS preprocesorů mluví prakticky pouze o nich a zároveň se tak stávají těmi nepoužívanějšími. To lze ostatně i potvrdit několika nejrůznějšími nezávislými anketami, zabývajícími se používáním CSS preprocesorů širokou odbornou veřejností, kde obvykle obsazují první příčky v kategorii oblíbenosti. Přesně z tohoto důvodu budou Sass, Less a Stylus použity jakožto nejvhodnější varianty pro porovnání tří nejlepších CSS preprocesorů, neboť právě mezi nimi se vývojáři běžně rozhodují. Jejich popularitě se podrobněji věnuje kapitola č. 4.4.

Tabulka č. 2 je přehledem základních informací o CSS preprocesorech Sass, Less a Stylus.

Název	Syntactically Awesome Style Sheets	Leaner Style Sheets	Stylus
Zkratka	Sass	Less	-
Logo			
Web	sass-lang.com	lesscss.org	stylus-lang.com
Autor	Hampton Catlin, Natalie Weizenbaum, Chris Eppstein	Alexis Sellier	TJ Holowaychuk
Vydání	2006	2009	2011
Licence	MIT	MIT	MIT
Jazyk	Ruby	JavaScript	JavaScript
Verze	3.5.5	3.0.0	0.54.5
Aktualizace	5. 1. 2018	10. 2. 2018	28. 4. 2016
Velikost	1,59 MB	1,61 MB	1,86 MB
GitHub	11 094	15 386	8 886

Tabulka 2: Přehled základní informací o Sass, Less a Stylus [vlastní zpracování]

4.1.1 Sass

Prvním ze tří vzájemně porovnávaných CSS preprocesorů je takzvaný Sass (Syntactically Awesome Style Sheets), jenž je s největší pravděpodobností nejznámější a nejvíce zmiňovaný CSS preprocesor vůbec. Představen byl poprvé již v roce 2006 a jako open source projekt pod licencí MIT si za téměř 12 let svého působení vybudoval značně silné jméno a stabilní komunitu. Je tím, který jako první definoval směr moderních CSS preprocesorů tak, jak jsou známé dnes [16].

O původní návrh se postaral Hampton Catlin, nicméně hlavním vývojářem a celkově nejdůležitějším tvůrcem je dnes Natalie Weizenbaum. Druhým vedoucím vývojářem je pak Chris Eppstein, známý především jako tvůrce prvního oblíbeného Sass frameworku s názvem Compass.

Programovacím jazykem, ve kterém byl CSS preprocesor Sass vytvořen, se stal jazyk Ruby. Proto se ihned stal vhodnou volbou pro vývojáře, kteří vyvíjí webové aplikace za pomoci frameworku Ruby on Rails, který je taktéž postaven na jazyku Ruby a tím pádem zprovoznění Sass pro ně nepředstavuje žádný problém. Avšak Sass se snaží být otevřený všem, nehledě na použitý vývojový jazyk. V průběhu své existence byl svým autorem celý předělán do jazyka C/C++ v rámci knihovny pojmenované LibSass. Ta nejen že zrychluje jeho chod, ale umožňuje jeho snadnou integraci i do ostatních jazyků pomocí takzvaných „wrapperů“. Díky tomu lze Sass spouštět stejně téměř v libovolném jazyce, jako je například JavaScript, PHP, Python, Java a další. Právě například JavaScript se momentálně v kombinaci se systémem Node.js stává velmi oblíbenou variantou běhového prostředí nejen pro Sass. Přestože zmíněný JavaScript umožňuje použití Sass přímo v prohlížeči, prakticky skoro v každém případě je při vývoji preferována metoda spouštění na straně serveru nebo na straně klienta v nástrojích mimo prohlížeč (jako Node.js).

Menší zvláštností oproti ostatním CSS preprocesorům pak může být, že Sass nabízí k dispozici rovnou dvě různé syntaxe, tedy dva odlišné styly zápisu svého jazyka. Ta novější, označována jako SCCS (Sassy CSS), se na rozdíl od té původní svým zápisem více blíží standardnímu CSS a od určité verze je už považována za preferovanou. Starší syntaxe, známá jednoduše jako Sass, je druhou funkční variantou se specifickými pravidly, která jsou sice poměrně netypické pro CSS, ale na druhou stranu jsou mnohem stručnější.

Kromě základních konstruktů, jako jsou proměnné, hnízdění, mixiny, extend a import, nabízí spoustu dalších užitečných funkcí. Je často spojován s oblíbeným frameworkem Compass, který Sass rozšiřuje o zajímavou sadu doplňků. Dále mezi další, již méně známé frameworky, patří například Burboun nebo Gumby. Stejně tak lze pro Sass využít celou řadu knihoven, které přináší předem vytvořené funkce nejrůznějšího využití.

4.1.2 Less

Dalším zástupcem, jenž je vybrán pro účel porovnání tří CSS preprocesorů, je takzvaný Less (Leaner Style Sheets). Jedná se o dobře známý CSS preprocesor, který je prakticky stejně oblíbený jako prvně uvedený Sass a lze jej považovat za jeho zdaleka největšího konkurenta. Byl obdobně představen jako open source projekt pod licencí MIT v roce 2009 a je tedy mladší pouze o tři roky oproti jeho předchůdci Sass. Tyto tři roky jsou dostatečně dlouho dobou na to, aby Sass získal nad Less určitý vývojový náskok, nicméně na druhou stranu to není dostatečně dlouhou dobou na to, aby nemohl Less postupem času Sass dohnat, a tak se i v podstatě stalo. Jeho tvůrcem je jistý Alexis Sellier, avšak dnes je Less spravován hlavně skupinou pravidelných přispěvatelů za obrovské podpory a zapojení komunity.

Úplně stejně jako Sass, i Less začínal v komunitě Ruby, jejíž jazyk si zvolil pro svůj vznik. Později byl však na základě rostoucí oblíbenosti Node.js a JavaScriptu předělán a zaměnil tak svůj primární implementační programovací jazyk z Ruby na JavaScript, čímž se stal ideální volbou pro vývojáře, kteří nechtěli jako součást svého vývojového procesu používat zbytečně navíc Ruby [17]. Díky rychle rostoucímu zájmu o Less byl postupně navíc pochopitelně paralelně předělán i do několika dalších vhodných jazyků včetně C++, Java a jiných [17].

Jelikož je Less implementován v jazyce JavaScript, tak není překvapením, že nabízí možnost překladu v samotném prohlížeči na straně klienta. Tato varianta se stejně jako u ostatních CSS preprocesorů nepoužívá z důvodu výkonnostních problémů a neúplné podpory prohlížečů. Proto se překlad jazyka Less do běžného CSS provádí především při vývoji za pomoci Node.js na straně serveru nebo na straně klienta mimo prohlížeč.

Samotná syntaxe je velice podobná základům standardního CSS, takže není potřeba používat pro Less zcela nový jazyk a lze si vystačit prakticky pouze s klasickým zápisem CSS, který je rozšířený o pár intuitivních prvků. Samozřejmostí je rozšíření funkcionality CSS o všechny funkce běžné pro CSS preprocesory, jako jsou proměnné, hnízdění, mixiny, extend a import, ale i několik dalších desítek praktických nástrojů významně usnadňujících práci. Mezi hojně používané knihovny pro Less patří například nejznámější Less Hat a Less Elements.

4.1.3 Stylus

Nejnovějším, respektive nejmladším, a zároveň posledním ze třetice porovnávaných CSS preprocesorů je takzvaný Stylus. Přestože se tomuto CSS preprocesoru dostává oproti jeho konkurentům Sass a Less o něco menší pozornosti, jedná se o stejně vyspělý nástroj, který je dokáže bez problému nahradit. Přitom vznikl až roku 2011, tedy po dlouhých pěti letech od uvedení prvního průkopníka CSS preprocesorů Sass a rovněž dva roky po Less. Stejně jako většina obdobných projektů byl i Stylus vydán jako open source pod licencí MIT, díky čemuž se taktéž snadno a rychle rozvíjí komunitním způsobem. Za jeho zrodem však stojí sám TJ Holowaychuk.

V zásadě opačná situace panuje oproti Sass a Less v oblasti implementace. Stylus od samého počátku funguje na platformě jazyka JavaScript v kombinaci s běhovým prostředím Node.js a doposud nikdy nedošlo k výměně tohoto řešení. Právě Node.js se v posledních několika letech stává jednoznačně nejpoužívanější metodou používání CSS preprocesorů jako takových, a to ať už na straně serveru nebo klienta. Zdánlivou nevýhodou se pak může jevit fakt, že Stylus na rozdíl od svých předchůdců není oficiálně převeden do ostatních jazyků. Existuje sice pár komunitních převodů, které však nejsou prakticky nijak využívané. Nicméně se ukazuje, že s rostoucím zastoupením JavaScript nemají vlastně vývojáři potřebu varianty napsané v jiných jazycích ani používat.

Podpora dvojí syntaxe u Sass není výjimkou a tuto možnost poskytuje i Stylus. Lze používat variantu zápisu jazyka Stylus, která se blíží zápisu klasického CSS, nebo naopak druhým, poněkud kratším způsobem tyto typické prvky jako závorky či středníky opomíjet a řídit tak strukturu dokumentu na základě odsazování jednotlivých řádků. V tomto případě však Stylus nenutí tyto dvě varianty od sebe striktně oddělovat například odlišným jménem souborů či jeho přípony.

Podstatnou výhodou, kterou Stylus disponoval v jeho začátcích je, že se mohl inspirovat u svých dvou předchůdců Sass a Less, kteří do značné míry ovlivnili jeho vývoj. Odnáší si z nich to nejlepší a stává se tak nejméně podobně kvalitním nástrojem s bohatou sadou funkcí. Samozřejmostí je podpora všech základních funkcionalit běžných pro CSS preprocesory, přičemž některé z nich vylepšuje po

svém a jiné přidává jako nové. Stylus může být i vhodně využit v kombinaci s knihovnou zvanou Nib, která obsahuje užitečná rozšíření navíc.

4.2 Porovnání základních funkcí

Jednoznačně nejviditelnějším rozdílem mezi všemi CSS preprocesory je syntaxe, tedy pravidla pro zápis jejich jazyka. Nicméně jiná syntaxe se ve většině případů nedá označit za lepší či horší a nemůže tak danému CSS preprocesoru zpravidla nutně přinášet výhodu ani nevýhodu. Syntaktických rozdílů je opravdu spousta, ale stále je vhodné Sass, Less a Stylus do jisté míry označit za syntakticky podobné jazyky. Přesto jsou zásadně odlišné důležitým způsobem, a to sice že Sass je jazyk imperativní, kdežto Less jazyk deklarativní [18]. Proto mnohem relevantnějším rozdílem, který dokáže skutečně určit, zda je jeden CSS preprocesor vhodnější k použití více než druhý, jsou samotné funkce, které nabízí.

V kapitole 5.7 jsou popsány základní funkce považované za nutnou součást každého CSS preprocesoru. Patří mezi ně proměnné, hnízdění, mixiny, extend, import a operátory. Každá z nich pochopitelně nechybí ani u jednotlivých zástupců porovnání, jimiž jsou Sass, Less a Stylus. Přestože tyto funkce v rámci všech třech CSS preprocesorů fungují prakticky stejně, nejsou úplně identické a v několika ohledech se liší. Jelikož se jedná o nejpoužívanější část CSS preprocesorů vůbec, stává se zajímavým a důležitým předmětem pro porovnání.

U každé funkce budou popsány markantní rozdíly v používání, na základě nichž, získají zúčastnění zástupci v porovnání bodové ohodnocení (čím více bodů tím lépe) jednotlivých funkcionalit zvlášť vzhledem k ostatním. Přidělování bodů probíhá následujícím způsobem:

- Funkce je rozdělena na několik dílčích **možností**, které CSS preprocesory dohromady nabízí. Tyto možnosti se mohou vzájemně lišit a jsou samostatně bodované. Součet bodů všech možností tvoří výsledný bodový zisk CSS preprocesoru u dané funkce.
- Pokud **možnost** funkce obsahuje každý z CSS preprocesorů a znatelně se od sebe liší, dostanou body dle vhodnosti jejich řešení (1, 2 nebo 3).

- V případě, že opět každý z CSS preprocesorů **možnost** funkce nabízí a jsou svým zpracováním rovnocenné, dostanou každý 1 bod.
- Jestliže kterýkoliv, respektive jeden z CSS preprocesorů vybranou **možnost** funkce neobsahuje, dostává 0 bodů a ostatní si rozdělí 3 body dle vhodnosti jejich řešení (1 nebo 2). Při rovnocenném řešení každý 1,5 bodu.
- Poskytuje-li danou **možnost** funkce pouze jeden ze tří CSS preprocesorů, dostává 3 body, ostatní 0 bodů.

4.2.1 Proměnné

U všech třech případů (Sass, Less a Stylus) probíhá práce s proměnnými na první pohled stejně, tedy samozřejmě krom typicky rozdílné syntaxe. Jejich ukládání funguje totožným způsobem, ale přesto se najde několik zajímavých odlišností.

4.2.1.1 Deklarace

Rovnou prvním a nejviditelnějším rozdílem je syntaxe při samotném deklarování proměnné, tedy způsob, jakým lze proměnnou zapsat. Každý z CSS preprocesorů používá pro identifikaci proměnné jiný klíčový znak. Konkrétně Sass používá symbol dolaru (`$`), Less poté symbol zavináče (`@`). Naopak Stylus v základu nevyžaduje žádný klíčový znak, a navíc umožňuje dobrovolné používání symbolu dolaru (`$`). Tyto dvě varianty dokonce lze používat kombinovaně, avšak v ideálním případě je vhodné si samozřejmě zvolit pouze jeden způsob. Jednotlivé proměnné jsou od sebe pak oddělené pomocí středníku (`;`), přičemž Stylus tento středník zavádí opět jako volitelný. Operátor jednoduchého přiřazení hodnoty ke konkrétní proměnné, respektive syntaktické oddělení názvu proměnné od její hodnoty, je u dvojce Sass a Less reprezentováno dvojtečkou (`:`). Pro stejný účel Stylus vymezuje symbol rovnítka (`=`).

Tři různé typy zápisu proměnné nejsou příliš velkým problémem a jedná se spíše o záležitost, na kterou se dá velmi snadno zvyknout. Nicméně Stylus svým přístupem, kdy při deklaraci proměnné nevyžaduje speciální klíčový symbol a pro operátor přiřazení používá symbol rovnítka (`=`), kopíruje přesně syntaxi většiny ostatních programovacích jazyků, které zvolily totéž. To pro Stylus znamená jistou výhodu,

jelikož je tento způsob zápisu pro spoustu lidí intuitivní a znovupoužitelný v rámci více programovacích jazyků, které běžně používají.

Jak bylo uvedeno, Sass a Less definují deklaraci proměnné takřka stejně, krom nepřehlédnutelné výměny symbolu dolaru (`$`) za symbol zavináče (`@`), uvedených před názvem proměnné. Výběr jedné z těchto dvou variant pravděpodobně závisí na osobních preferencích, avšak zatímco symbol dolaru nemá ve standardním CSS zcela žádný význam, symbol zavináče jej bohužel už má. Využívá se při deklaraci pravidel pro animace, takzvaných *„@keyframes“* a stejně tak pro deklaraci takzvaných *„@query“*, pravidel pro specifikování různých typů zobrazovacích zařízení [19]. To může vést k určité záměně těchto dvou stejně zapisovaných, ačkoliv odlišných funkcionalit [19]. Narušení tohoto konceptu znamená pro Less značnou nevýhodu.

Menší zvláštnost u identifikátorů povoluje z historických důvodů Sass. Pomlčky a podtržítka se mohou ve jménech nejen proměnných používat jako vzájemně zaměnitelné znaky. Znamená to, že k deklarované proměnné s pomlčkou v názvu je možné přistupovat i jako k proměnné s podtržítkem namísto pomlčky a opačně. To nelze považovat za výhodu ani nevýhodu, ovšem střídavé zaměňování těchto znaků by mohlo být poměrně matoucí. Dále je tvoření názvů proměnných ve všech třech případech omezeno typickými pravidly. Například nelze začít speciálními znaky. Jediný Sass poté nedovoluje začít ani čísly. S nejčastěji používanými podtržítky, pomlčkami a takzvanou „velbloudí notací“ není problém u žádného z nich.

Vzhledem ke všem uvedeným rozdílům, je bodové hodnocení CSS preprocesorů u možnosti „deklarace“ následující:

Sass: 2, Less: 1, Stylus: 3

4.2.1.2 Datové typy

Už méně zřetelný rozdíl nastává v případě datových typů, které je možné v rámci proměnných využívat. Oproti ostatním programovacím jazykům je u CSS preprocesorů mnohem menší množství druhů hodnot, jenž lze pomocí proměnných ukládat. Rozdíl v nich je nepatrný, protože všichni tři zástupci (Sass, Less a Stylus) podporují především následující hlavní datové typy:

- Čísla (např. 1.2, 13, 10px)
- Řetězce textu, s a bez uvozovek (např. “foo“, ‘bar‘, baz)
- Barvy (např. blue, #04a3f9, rgba(255, 0, 0, 0.5))
- Seznamy hodnot, oddělených mezerami nebo čárkami (např. 1.5em 1em 0 2em, Helvetica, Arial, sans-serif)
- Logické hodnoty (true, false)

Všechny tyto uvedené datové typy patří prakticky mezi nejpoužívanější a potřeba využití některých dalších nastává málokdy. Přesto se u jednotlivých CSS preprocesorů vyskytují i další datové typy, které oproti svým konkurentům nabízí navíc a staví se tak do pozice vyspělejšího nástroje.

Jediný Sass podporuje takzvané mapy. Ty reprezentují souvislost mezi klíčem a hodnotou, kde klíč se používá pro vyhledání přidružených hodnot [8]. Umožňují snadné shromažďování hodnot do pojmenovaných skupin a dynamický přístup k těmto skupinám [8]. Nemají přímou obdobu v CSS, nicméně jsou syntakticky podobné výrazům *„media query“* [8]. Dále opět Sass povoluje navíc vytvářet proměnné jakožto reference na konkrétní funkce. Naopak pouze Less zavádí do proměnných ukládání sady pravidel. Lze tak například do jedné proměnné vložit rovnou celou skupinu CSS vlastností. Bohužel Stylus v tomto ohledu nenabízí nic, co by jej obdobně zvýhodnilo.

Bez výjimky všichni tři ale podporují veškeré ostatní typy hodnost CSS vlastností, jako klíčové slovo *„!important“* či netypické znaky vlastnosti *„unicode-range“*, přičemž Sass a Stylus nerozlišují zápis těchto hodnot od zápisu klasického řetězce textu psaného bez uvozovek. Oproti tomu Less stejný zápis komplikuje o nutnost přidání speciálního symbolu vlnovky (~) a uvozovek. Nakonec je vhodné zmínit, že datovým typům se ve své dokumentaci věnuje pouze Sass.

Při uvážení všech uvedených rozdílů, je bodové hodnocení CSS preprocesorů u možnosti „datové typy“ následující:

Sass: 3, Less: 1, Stylus: 1

4.2.1.3 Interpolace

Proměnné ve všech třech porovnávaných preprocesorových jazycích jsou schopné jako své hodnoty ukládat i typické CSS selektory. Výslednou proměnnou, respektive její hodnotu, kterou uchovává, následně lze v jakémkoliv CSS pravidle použít právě v rámci interpolace namísto konkrétního selektoru. Tato funkce má pochopitelně napříč CSS preprocesory rozdíly v syntaxi, ale i v možných místech jejího použití.

Pro interpolování určité proměnné používá Stylus zapsání jejího jména do složených závorek. Doplnění symbolu mřížky (#) před levou složenou závorku pak vyžaduje Sass a stejně tak činí Less, ovšem se symbolem zavináče (@). Podstatnější rozdíl přichází u zápisu samotných selektorů jako hodnot proměnných. Ve všech případech jsou selektory považovány za textové řetězce, problémem jsou však uvozovky, které se ve výsledném CSS v případě selektoru nesmí objevit. Automatické odstraňování uvozovek při interpolaci proměnné provádí Sass. Je tedy možné hodnotu zapsat oběma způsoby, s uvozovkami i bez nich. Nicméně výjimkou je z nějakého důvodu selektor značící třídu, tedy obsahující tečku. Ten lze bohužel zapsat pouze s uvozovkami. To může působit poněkud matoucně. Odstraňování uvozovek provádí i Stylus, ve kterém je však nutné onu hodnotu zapsat pouze s nimi. Opačně k tomuto přistupuje Less, který se odstranění uvozovek nedopouští a je možné tak hodnotu zapsat pouze bez uvozovek.

Mimo drobné syntaktické rozdíly, může být mnohem důležitější, kde všude lze v rámci CSS preprocesoru funkci interpolace využít. Zatímco Less nabízí využití v místech jako selektory, URL, import a CSS vlastnosti, tak Sass a Stylus jsou omezeny na použití v selektorech a CSS vlastnostech.

Na základě všech uvedených rozdílů, je bodové hodnocení CSS preprocesorů u možnosti „interpolace“ následující:

Sass: 1, Less: 3, Stylus: 2

4.2.1.4 Výchozí hodnota

Přidáním klíčového slova *„!default“* na konec hodnoty lze do proměnných přiřadit hodnotu pouze, pokud nebyla doposud žádná přiřazena [8]. To znamená, že pokud byla proměnné již přiřazena hodnota, nemůže být změněna, ale pokud dosud

žádnou hodnotu nemá, bude jí přiřazena [8]. To může být užitečné, ale poměrně neintuitivní řešení. Takovou možnost nabízí pouze Sass. Ačkoliv bez použití jakéhokoliv klíčového slova, tuto schopnost umožňuje i Less, i když dost odlišným způsobem. Vyplývá to z jeho řešení rozsahu proměnných, kde díky funkcionalitě Lazy Loading je každá hodnota proměnné vždy daná poslední definicí. Hodnotu tak lze kdykoliv jednoduše přepsat novou definicí vytvořenou za tou původní a ve výsledku funkce výchozích hodnot není potřeba, což se zdá být vhodnější variantou.

Proto je bodové hodnocení CSS preprocesorů u možnosti „výchozí hodnota“ následující:

Sass: 1, Less: 2, Stylus: 0

4.2.1.5 Vymezení rozsahu

Jakmile je zavedena libovolná proměnná, nastává možnost jí s okamžitou platností začít využívat. To znamená opakovaně přistupovat k její hodnotě jednoduchým odkázáním se na její jméno. To má ovšem svá omezení a nelze vždy k zavedené proměnné přistupovat naprosto odkudkoliv, ale zpravidla pouze v určitém rozsahu s ohledem na místo, kde byla proměnná vytvořena. Systém určování tohoto rozsahu nese v angličtině běžně označení „scoping“ a u CSS preprocesorů funguje různě. Právě zde dochází pravděpodobně k nejvýznamnějším rozdílům, co se celkové funkčnosti proměnných týče.

Překvapivě Sass a Stylus řeší vymezení rozsahu proměnných prakticky stejným způsobem. Proměnné jsou dostupné pouze v úrovni vnoření selektoru, kde jsou definované [8]. To znamená, že pokud je proměnná definována mezi jednotlivými CSS vlastnostmi uvnitř deklaračního bloku, je platná jednak v tomto bloku a stejně ve všech následujících blocích do tohoto bloku vnořených, tedy ve všech úrovních zanoření směrem níže, respektive nikoliv ve směru k rodičovskému bloku, ale naopak ve směru svých potomků. Pakliže je v rámci kteréhokoliv deklaračního bloku odkazováno na proměnnou, která ve stejném bloku není definována, použije se první definovaná proměnná stejného jména nalezená mimo deklarační blok po směru zanoření výše, tedy ve směru rodičovských bloků. Jestliže jsou proměnné definovány mimo kterýkoliv selektor (deklarační blok), jsou dostupné kdekoliv [8].

Mohou být také definovány pomocí klíčového slova *„!global“* na konci jejich hodnoty. V takovém případě jsou taktéž dostupné kdekoliv, a to i v případě definování uvnitř (zanořeného) deklaračního bloku. To platí pouze pro Sass. Nesmírně důležitým principem, který nepřímo souvisí s vymezením rozsahu proměnných je fakt, že proměnná je dostupná až poté, co byla definována. Nelze k ní přistupovat před samotným definováním. Proto se ve většině případů provádí definování veškerých proměnných hned na začátku kódu před vším ostatním.

V některých částech se naprosto opačnou cestou vydal Less, který řeší rozsah proměnných pomocí funkcionality zvané Lazy Loading. Ta zapříčiňuje, že proměnná může být bez problému použita i před jejím samotným definováním. Omezení rozsahu dostupnosti proměnné na jeden deklarační blok funguje stejně včetně situace, kdy nemůže být nalezena v aktuálním rozsahu, a hledá se tak její hodnota v rozsahu rodičovském, dokud není nalezena, nebo nedojde k chybě po bezúspěšném dosažení konce nejvyšší úrovně. K menšímu rozdílu dochází, pokud je proměnná definována v jednom rozsahu vícekrát. Na rozdíl od použití první nalezené definice před místem odkazování se na danou proměnnou (Sass a Stylus), dojde automaticky k použití definice poslední.

Vzhledem k tomu, že ani jednomu z přístupů se obecně nadají vzájemně vymezit žádné výhody a nevýhody, dají se považovat za rovnocenné a jejich hodnocení je spíše předmětem osobních preferencí. Za takových okolností je bodové hodnocení CSS preprocesorů u možnosti „vymezení rozsahu“ následující:

Sass: 1, Less: 1, Stylus: 1

4.2.1.6 Proměnlivé proměnné

Tento zdánlivě matoucí název odkazuje na velmi jednoduchou funkcionalitu, kterou ze všech tří porovnávaných CSS preprocesorů nabízí pouze Less. A to možnost definovat jméno jedné proměnné pomocí hodnoty jiné proměnné. Toť vše. Přestože její využití může být v některých ohledech diskutabilní, specifické příklady se určitě najdou. Syntaxe zápisu názvu proměnné pomocí proměnné je pojat vcelku intuitivně. Začíná symbolem zavináče (@) stejně jako v případě běžné proměnné. Místo však samotného názvu pokračuje typický zápis jiné proměnné (pochopitelně

včetně dalšího symbolu zavináče), jejíž hodnota představuje onen název. Dochází tak k zápisu dvou symbolů zavináče (@) za sebou.

Z důvodu absence této funkce u ostatních zástupců je bodové hodnocení CSS preprocesorů u možnosti „proměnlivé proměnné“ následující:

Sass: 0, Less: 3, Stylus: 0

4.2.1.7 Vlastnosti jako proměnné

Snaha udělat kód takzvaně co nejlehčí a nejčistší je správným krokem. Tomu z části napomáhá užitečná možnost zacházet s CSS vlastnostmi jako s proměnnými. Jedná se o schopnost odkazovat na hodnoty již definovaných vlastností bez nutnosti přiřazování jejich hodnot do proměnných. Znamená to, že jakmile je v CSS pravidle deklarována kterákoliv CSS vlastnost s libovolnou hodnotou, lze se na ní odkázat v rámci specifikování hodnoty jiné CSS vlastnosti. To je možné provést zapsáním vyhrazeného symbolu, za kterým následuje jméno odkazované CSS vlastnosti. Pro tento účel využívá Less symbol dolaru ($) a Stylus symbol zavináče (@). Kupodivu Sass tuto funkci zatím nenabízí.

I přes to, že se na první pohled jedná o jednoznačnou malou funkci, se její použití liší více než jen v syntaxi. Prvně zmíněný Less implementuje toto chování skutečně jako proměnné. Důležité je, že veškerá pravidla ohledně rozsahu proměnných zde platí úplně stejně. Při hledání hodnoty odkazované CSS vlastnosti je použita její poslední definice v rámci rozsahu. Zároveň při nepřítomnosti CSS vlastnosti v aktuálním rozsahu probíhá její hledání i rodičovské rozsahu, tedy v zanoření stále směrem výš. Naopak Stylus takto s touto funkcí nepracuje, a bohužel se tak nelze odkázat na CSS vlastnost definovanou v rodičovském deklaračním bloku. Navíc samotné odkazování, totožně jako u proměnných, musí být provedeno až po definování oné CSS vlastnosti.

Vzhledem k rozdílu ve zpracování je bodové hodnocení CSS preprocesorů u možnosti „vlastnosti jako proměnné“ následující:

Sass: 0, Less: 2, Stylus: 1

4.2.1.8 Shrnutí

Deklarace je nejčastějším úkonem, který se k proměnným vztahuje. Jednoznačně nejintuitivnější zápis proměnných má Stylus, protože se nejvíce blíží ostatním programovacím jazykům a nenarušuje zaběhnutý koncept. Standardní datové typy podporují všichni zúčastnění. Přestože jich Sass podporuje o něco více než ostatní, jejich použití není tak časté jako například ukládání sady CSS vlastností, které naopak nabízí pouze Less. V interpolaci proměnných dochází k rozdílům, ty jsou ale spíše zanedbatelného rázu. Specifikování výchozí hodnoty poměrně krkolomným způsobem nabízí Sass, nicméně Less tento problém nemusí vůbec řešit díky svému celkově odlišnému přístupu ke zpracování proměnných. Zápis jména proměnné pomocí jiné proměnné umožňuje Less. Využití však nastává zřídka a ve většině případech se u ostatních CSS preprocesorů dá stejného výsledku dosáhnout i bez této funkce.

Každý ze zástupců zpracovává některé možnosti lépe a některé naopak hůře než ostatní. Na základě toho jsou řádně obodovaní. Nicméně žádná z možností jako deklarace, datové typy, interpolace, výchozí hodnota a proměnlivé proměnné není natolik důležitější než jiná, aby její vhodnější fungování daný CSS preprocesor razantně zvýhodnilo. Vždy je vhodné brát v úvahu, které možnosti budou v rámci používání CSS preprocesoru více využívány. Stejně tak je dobré přihlédnout na osobní preference syntaxe.

Opačně tomu je u porovnání vlastností jako proměnných. Jedná o princip odkazování na hodnoty CSS vlastností, který dokáže být rozhodně užitečný, a proto ve chvíli, kdy Sass tuto funkci nenabízí, stojí její přítomnost za větší zvážení. Podobně velkému zvážení by se mělo dostat rozsahu proměnných, kde Less volí naprosto odlišný přístup zavedením systému zvaného Lazy Loading.

Tabulka č. 3 shrnuje průběh bodování CSS preprocesorů v jednotlivých možnostech funkce proměnných. Po celkovém součtu bodů se na první místo s jistou rezervou dostává Less.

PROMĚNNÉ			
	Sass	**Less**	**Stylus**
Deklarace	2	1	3
Datové typy	3	1	1
Interpolace	1	3	2
Výchozí hodnota	1	2	0
Rozsah	1	1	1
Proměnlivé proměnné	0	3	0
Vlastnosti jako proměnné	0	2	1
Celkem bodů	**8**	**13**	**8**

Tabulka 3: Souhrn bodování funkce proměnných [vlastní zpracování]

4.2.2 Hnízdění

V momentě, kdy nastane v klasickém CSS potřeba zápisu selektorů pro vícero elementů, jenž mají společný rodičovský element, nezbývá jiná možnost než v každém jednom selektoru referenci na rodiče stále dokola opisovat. Hnízdění tento problém velice intuitivně odstraňuje a podporu jeho základního principu nabízí Sass, Less a Stylus naprosto bez rozdílu. Mimo to se však CSS preprocesory rozchází, co se týče menších dodatečných funkcí rozšiřujících možnosti hnízdění.

4.2.2.1 Syntaxe

U všech tří zástupců se provádí specifikování selektoru jakožto potomka rodičovského selektoru pouhým zapsáním celého CSS pravidla (potomka) do vnitřku deklaračního bloku jiného CSS pravidla (rodiče). Nic víc není potřeba. Jde o jednoduchý, přehledný, a především logický způsob.

Bodové hodnocení CSS preprocesorů u možnosti „syntaxe“ nemůže být jiné než následující:

Sass: 1, Less: 1, Stylus: 1

4.2.2.2 Vnoření vlastností

Syntaxe, a hlavně základní funkčnost je sice totožná, ale jak již bylo řečeno, rozdíly se objevují až v možnostech poskytovaných navíc. V tomto ohledu posouvá Sass metodu hnízdění o krok dál a povoluje hnízdění jednotlivých CSS vlastností neboli jejich zanořování. Samotné CSS má docela dost vlastností, které jsou v takzvaných jmenných prostorech, například vlastnosti *„font-family“*, *„font-size“* a *„font-weight“* jsou všechny ve jmenném prostoru *„font“* [8]. Pokud je potřeba v CSS nastavit skupinu vlastností ze stejného jmenného prostoru, musí se pokaždé znovu jméno jmenného prostoru napsat [8]. Proto Sass poskytuje následující zkratku: stačí jednoduše zapsat jmenný prostor pouze jednou a poté do něj vnořit („zahnízdit“) všechny dílčí vlastnosti [8].

Syntaxe hnízdění vlastností je opravdu jednoduchá a obdobná s hnízděním selektorů. Za specifikováním jména určitého jmenného prostoru následuje dvojtečka a složené závorky, ve kterých jsou běžným způsobem nadále deklarovány dílčí CSS vlastnosti s jejich přidruženými hodnotami. Je velká škoda, že toto zpestření hnízdění nabízí pouze Sass, který tímto minimálně zase trochu napomáhá dodržování důležitého přístupu známého jako DRY.

Vzhledem k podpoře funkce ze strany jediného zástupce, je bodové hodnocení CSS preprocesorů u možnosti „vnoření vlastností“ následující:

Sass: 3, Less: 0, Stylus: 0

4.2.2.3 Odkaz na rodiče

Výchozí chování hnízdění ve své podstatě říká, že vnořený selektor je potomkem svého rodiče. Jelikož selektory odkazují na existující elementy ve strukturovaném HTML dokumentu, použití hnízdění tak reflektuje skutečné zanoření těchto daných elementů jakožto právě rodičů a potomků. Tomu budou pochopitelně odpovídat i výsledné selektory po přeložení do standardního CSS překladačem CSS preprocesoru, a to ve všech případech.

Někdy je však užitečné použít rodičovský selektor zanořeného pravidla jiným způsobem než výchozím [8]. Například může být v zájmu zavést odlišné CSS vlastnosti pro selektor, který je záměrně doplněný o některou z pseudo-tříd.

Obdobná je situace při potřebě určit, že konkrétní HTML element má jistou třídu. V těchto případech se nejedná přímo o reprezentaci zanoření selektorů a tím pádem i příslušných elementů, ale spíše o vytvoření kombinace ze selektoru rodiče a jeho potomka. Tohoto efektu lze docílit explicitním specifikováním, na jaké místo selektoru potomka má být vložen selektor rodiče, aby došlo ke zkombinování. Důležité při tom je, že se na ono místo nevkládá pouze nejbližší rodičovský selektor, ale zároveň i rodičovský selektor rodiče a tak dál až na první úroveň zanoření. Stejně tak je pochopitelné, že pokud se rodičovský selektor skládá z více oddělených selektorů, je na místo symbolu postupně vložen každý z nich a vytvoří se tak zvlášť všechny kombinace. Specifikování se provádí zástupným klíčovým symbolem v podobě ampersandu (&), který k tomuto účelu používá jak Sass, tak Less a Stylus. Přestože dojde při překladu k nahrazení symbolu ampersandu (&) za rodičovský selektor, bude pochopitelně nejprve tento rodičovský selektor klasicky vyhodnocen v rámci hnízdění jako selektor vnořeného elementu, má-li taktéž svého rodiče a alespoň jednu CSS vlastnost.

Ke vší jednoduchosti této funkce i zde jsou určité implementační rozdíly u jednotlivých CSS preprocesorů. Sice menšího rázu, ale o to překvapivější. Prvním upřesněním, které se přímo vybízí, je možnost použití rovnou několika rodičovských odkazů v jednom jediném selektoru. Symbol ampersandu (&), se skutečnou funkcí odkazu na rodičovský selektor, se opravdu může objevit v rámci jednoho selektoru bez problému vícekrát. Bohužel Sass se v tomto ohledu odlišuje drobným negativním nedostatkem. Zapsání dvou a více symbolů ampersandu (&) bezprostředně za sebou totiž končí z nepochopitelného důvodu chybou, a proto prakticky pouhého spojení rodiče a potomka nelze docílit.

Podobně udivující rozdíl nastává ve snaze vygenerovat všechny možné permutace z rodičovských selektorů. Pakliže je rodičovský selektor selektorem složeným, tedy jedná se o více jednotlivých selektorů oddělených čárkami, je jeho potomek schopen tyto permutace vytvořit odkázáním se na svého rodiče ve specifickém tvaru. Odkaz je složen ze dvou symbolů ampersandu (&), mezi které je vepsán speciální typ CSS selektoru v podobě znaku plus (+), určeného pro výběr dvou elementů umístěných ve stejné úrovni zanoření ihned za sebou. Při identickém zápisu u zástupců Sass a

Less budou výsledkem kombinace každý selektor s každým, přičemž Stylus stejnou situaci zpracuje jako kombinace každý selektor pouze sám se sebou. Shoda řešení dvou CSS preprocesorů oproti třetímu nutně neznamená správné východisko, naopak je výsledek této operace z logického hlediska diskutabilní a nejasný.

Nakonec bez sebemenšího rozdílu lze v rámci selektoru využít odkaz na rodiče v jakémkoliv pořadí, ve smyslu jeho libovolného umístění, ať už před, nebo za onen selektor. Nepochybně je při nejmenším spravedlivé dva výše zmíněné rozdíly v odkazování na rodiče zmínit. Nicméně nejedná se o zásadní problémy, které by mohly používání této funkce výrazně narušit.

Zpracování se dá považovat za totožné, a proto je bodové hodnocení CSS preprocesorů u možnosti „odkaz na rodiče“ následující:

Sass: 1, Less: 1, Stylus: 1

4.2.2.4 Odkaz na selektor

Předchozí možnost odkazu na rodiče funguje skvěle. Jedná se ve své podstatě o jakýsi odkaz na selektor, přičemž za tento selektor se nepovažuje pouze nejbližší rodičovský, ale veškeré selektory ze všech nadřazených úrovní zanoření. Tento způsob chování je vždy stejný, a nelze tak žádným způsobem omezit použití všech úrovní na konkrétní volitelný počet, což může být v některých případech žádoucí. Zatímco Sass ani Less toto neumožňují, Stylus ano. Mimo odkazu na rodiče zavádí funkcionalitu odkazu na vybraný selektor, který leží v určité libovolné úrovni zanoření. Takový odkaz lze vytvořit rovnou v několika různých variantách. První z nich je uváděný jako částečný odkaz. Dalšími jsou pak takzvaný počáteční odkaz, relativní odkaz nebo kořenový odkaz.

Částečný odkaz pracuje podobně jako rodičovský odkaz, ale mezi tím, co rodičovský odkaz obsahuje všechny selektory, částečný odkaz obsahuje pouze prvních N sloučených úrovní zanořených selektorů, takže lze k těmto zanořeným úrovním přistupovat individuálně [20]. Číslo N pak může být jak kladné, tak i záporné a odpovídá konkrétní úrovni zanoření. Celý odkaz je v selektoru reprezentován symbolem stříšky (^), za kterou následuje v hranatých závorkách právě číslo N. Důležité je opět zmínit, že stejně jako u odkazu na rodiče se i zde na místo odkazu

nevkládá pouze selektor z úrovně N, ale všechny rodičovské selektory až po specifikovanou úroveň N. Navíc v částečném odkazu je možné použití rozsahu úrovní, tedy namísto samostatného N formát N až M. Zápis zůstává stejný, nicméně jediné číslo v hranatých závorkách je zaměněno za čísla dvě, oddělených dvěma tečkami.

Počáteční odkaz je pouhou zkratkou pro částečný odkaz s číslem N rovným nule. Zapisuje se symbolem vlnovky a dopředného lomítka (`~/`). Relativní odkaz, značený dvěma tečkami a dopředným lomítkem (`../`), ukazuje na selektor předchozí úrovně zanoření relativně vůči svému místu použití. Poslední variantou je kořenový odkaz symbolizovaný dopředným lomítkem (`/`). Ten se odkazuje na kořenový kontext, respektive způsobí ignorování všech úrovní zanoření.

Pokročilejší práci s úrovněmi zanoření nabízí pouze Stylus. Bodové hodnocení CSS preprocesorů u možnosti „odkaz na selektor“ tak nemůže být jiné než následující:

Sass: 0, Less: 0, Stylus: 3

4.2.2.5 Shrnutí

Hnízdění se stalo jednou z mála funkcí, jejíž způsob použití je u všech třech CSS preprocesorů prakticky totožný. Sass, Less a Stylus se shodují nejen v syntaxi, ale až na drobné detaily taktéž v možnosti použití takzvaného odkazu na rodiče. Sám Stylus posouvá práci s odkazy o něco dále a zavádí odkazování na selektory dle výběru konkrétní úrovně zanoření. Naopak Stylus rozšiřuje hnízdění o jeho aplikaci i na samotné CSS vlastnosti, které lze obdobně zanořovat v rámci jejich jmenných prostorů. Přestože Sass a Stylus získávají stejné bodové ohodnocení, nabírají tyto body rozdílně ve dvou různých možnostech, kde každý podporuje právě jednu z nich. Pokud by se měl v případě hnízdění vybrat lepší z nich, byl by to určitě Sass díky jeho možnosti vnoření CSS vlastností, která je z praktického hlediska rozhodně užitečnější než možnost pokročilých odkazů na selektory, kterou poskytuje Stylus.

V tabulce č. 4 je zobrazeno bodování CSS preprocesorů ve všech možnostech funkce hnízdění. První místo se dělí mezi Sass a Stylus, jejichž celkový součet bodů je stejný.

HNÍZDĚNÍ			
	Sass	**Less**	**Stylus**
Syntaxe	1	1	1
Vnoření vlastností	3	0	0
Odkaz na rodiče	1	1	1
Odkaz na selektor	0	0	3
Celkem bodů	**5**	**2**	**5**

Tabulka 4: Souhrn bodování funkce hnízdění [vlastní zpracování]

4.2.3 Mixiny

Téměř veškeré funkce CSS preprocesorů jsou typické programové konstrukty známé z většiny ostatních programovacích jazyků a používají se obdobným způsobem. Ovšem dvě z těchto funkcí jsou velice specifické hlavně pro moderní CSS preprocesory a mixiny jsou jednou z nich. Jedná se o funkcionalitu, které se dostává širokého využití, a proto je její počet nabízených možností a variant v porovnání s ostatními dostupnými funkcemi o dost větší.

Jelikož všechny tři CSS preprocesory disponují nepřeberným množstvím specifik pro používání mixin, především pak Less, není možné jednoznačně a smysluplně porovnat každé z nich mezi jednotlivými zástupci. Nicméně především první tři možnosti porovnávat lze, protože se jejich zpracování výrazně od sebe liší. Patří mezi ně základní syntaxe, rozsah a přístup k proměnným. Tyto části jsou porovnány stejným způsobem, jako všechny předešlé funkce.

Poté následuje popis několika dalších neméně důležitých možností mixin. Ten ale nabývají spíše informativního charakteru o plné podpoře těchto dílčích možností ze strany daného CSS preprocesoru. Většina z nich se pochopitelně syntakticky liší. Tyto drobné rozdíly však nebudou zdůrazňovány. Proto je v tomto případě mírně upraveno bodování. Jeden bod získá každý CSS preprocesor, jenž konkrétní možnost

v rámci funkce mixin alespoň částečně poskytuje. Pakliže neposkytuje a zároveň stejného efektu se dá docílit jinou cestou, získává i tak 0 bodů.

4.2.3.1 Základní syntaxe

Odlišná syntaxe je vždy prvním rozdílem, který je při porovnání kterýchkoliv funkcí ihned vidět. Jinak tomu není ani u mixin. V tomto případě však tento rozdíl přímo vyplývá ze samotné funkcionální konstrukce. Ta se principiálně výrazně liší a ovlivňuje tak podobu zápisu mixinu.

První způsob definování mixinu, který zavádí Sass, je jeho vlastní ojedinělá syntaxe určená direktivou *„@mixin“*, za níž následuje název mixinu, volitelně parametry uvedené v kulatých závorkách a dále blok obsahující všechen obsah mixinu, tedy nejčastěji sada CSS vlastností. Následně na jakémkoliv místě v dokumentu, kde má být daný mixin použit, je vkládán pomocí direktivy *„@include“*. Ta je opět doplněna o název mixinu a při použití parametrů i o příslušné vstupující hodnoty v kulatých závorkách. V nepřítomnosti parametrů nejsou kulaté závorky nutné v místě definice mixinu ani jeho použití.

Naopak zcela odlišný přístup volí Less, který automaticky považuje ve výchozím chování každý selektor za definici mixinu, a to ať už například selektor třídy nebo identifikátoru. Mixin může být jednoduše vytvořen úplně stejně jako obyčejný selektor, nebo právě lze využít již existující selektor. Pokud jsou za selektor dopsány kulaté závorky, je explicitně označen jako mixin a ztrácí funkci klasického CSS pravidla, kterou bez závorek pochopitelně zároveň plní. Závorky jsou volitelné stejně jako vstupní parametry, které ovšem závorky vyžadují. Na místě vložení mixinu je zapsán příslušný selektor opět doplněný o libovolné závorky, ve kterých jsou případné hodnoty parametrů.

Další přístup k syntaxi mixinů přináší Stylus. Mixiny a funkce jsou významově definovány stejně, ale používají se různými způsoby [20]. Libovolnou funkci je tedy možné zavést pouhým zapsáním jejího názvu a obsahu. Pro použití mixinu postačí zápis jeho názvu. Jako u všech ostatních, i zde lze volitelně dopsat kulaté závorky a případně parametry, a to na místě definice mixinu i pozdějšího použití. Takový postup je správný, ale Stylus jej z určitého hlediska považuje za přebytečný. Při

používání mixinů je možné zcela vynechat zmíněné kulaté závorky, což poskytne podporu transparentních vlastností [20]. Za takových okolností vypadá zápis volání parametrického mixinu v CSS pravidle úplně stejně jako typický zápis CSS vlastnosti.

Ani jedna ze tří metod definování mixinů se nedá označit za jednoznačně lepší či horší. Například Less nenutí pro vytvoření ani použití mixinu používat nové direktivy, jako to dělá Sass. Nicméně někdo by mohl namítat, že právě explicitní vyjádření pomocí direktivy vhodně značí mixiny, a nelze si je tak při práci splést se selektory. Takových rozepří může nastat více a většinou mají všechny svůj důvod. Proto výhody a nevýhody syntaxe mixin záleží spíše na osobních preferencích kodéra.

Bodové hodnocení CSS preprocesorů u možnosti „základní syntaxe“ je následující:

Sass: 1, Less: 1, Stylus: 1

4.2.3.2 Vymezení rozsahu

Podobně jako u proměnných, i u této funkce dochází k určitému vymezení jejího rozsahu, tedy ke specifikaci vlastnosti zvané „scoping“. Jedná se o popis chování a používání mixin v různých úrovních zanoření selektorů a určení jejich dostupnosti v rámci dokumentu. Zpracování vymezení rozsahu není u CSS preprocesorů zcela shodné a nastává zde jeden razantní rozdíl.

V místě vložení neboli volání mixinu, se nejprve začne v dokumentu dle volaného jména vyhledávat již vytvořený mixin, který bude následně v tomto místě použit. Hledání probíhá pouze v úrovni zanoření volání mixinu. Přičemž Sass a Stylus hledají definici mixinu v dané úrovni směrem nahoru od zápisu onoho volání a Less naopak prohledává celý kontext úrovně. Pokud není v aktuální úrovni definován žádný mixin, začne hledání v úrovni rodiče, nejčastěji rodičovského selektoru. Takto se pokračuje, dokud není příslušný mixin nalezen.

Na základě způsobu vyhledávání mixinu v úrovni, směrem nahoru od místa volání, mohou Sass a Less kdykoliv průběžně změnit obsah mixinu jeho opětovným definováním. Lze tak ve stejné úrovni pod místem prvního volání mixinu znovu definovat stejný mixin s jiným obsahem a bez problému provést druhé volání. Vždy se použije první nalezený mixin, tedy jeho nejbližší definice před místem volání. Na

začátku zmíněný razantní rozdíl se pojí na Less, ve kterém se vícenásobné definice mixinů stejného jména v rámci úrovně nepřepisují, ale jsou pomyslně považovány za jednu. Ve chvíli, kdy Less při hledání mixinu prohledává celý kontext úrovně, aplikuje všechny nalezené definice mixinů najednou, nikoliv pouze první nalezenou. Důsledkem je možnost použití obsahu z více definicí mixinů zároveň.

Opět není takřka možné korektně označit kterékoliv ze dvou odlišných chování za bezprostředně lepší. Oba uvedené přístupy poskytují rovnocennou reprezentaci této funkce, a proto je bodové hodnocení CSS preprocesorů u možnosti „vymezení rozsahu“ následující:

Sass: 1, Less: 1, Stylus: 1

4.2.3.3 Přístup k proměnným

Třetím rozdílem, který se týká spíše podstaty chování implementace mixin jako takových nežli podpory určité možnosti navíc, je zacházení s proměnnými, které nejsou definovány přímo jako součást obsahu mixinu.

Jestliže je v rámci mixinu použita proměnná, která zde není definována a ani nevstupuje jako předaný parametr, všechny tři CSS preprocesory začnou hodnotu této proměnné hledat o úroveň výše, tedy v rodičovském rozsahu mixinu. Pokud není nalezena ani tam, pokračují postupně v hledání stále ve vyšší a vyšší úrovni zanoření, až po kořenovou úroveň dokumentu. Jakmile není proměnná nikde k nalezení, Sass chybovou hláškou oznámí, že použitá proměnná není definována. Nicméně Less a Stylus stejnou chybou nekončí a stejně tak mimo vnější rozsahy mixinu prohledají úroveň (rozsah), kde bylo použití mixinu zavoláno. V případě že opět ani zde není proměnná definována, Less jako jediný zachází ještě dále. Od místa volání mixinu začne podobně jako od místa definice mixinu postupně navíc prohledávat rodičovské úrovně.

Na situaci chybějící definice proměnné v mixinu je nejlépe připraven Less a o něco méně Stylus. Naopak Sass je o jeden krok pozadu. Tudíž bodové hodnocení CSS preprocesorů u možnosti „přístup k proměnným“ je následující:

Sass: 1, Less: 3, Stylus: 2

4.2.3.4 Další možnosti

Selektory: Mixiny mohou obsahovat více než jen vlastnosti, mohou obsahovat i selektory [21].

Sass: 1, Less: 1, Stylus: 1

Odkaz na rodiče: V případě, že je možné v mixinu používat selektory, je zároveň možné využití jejich principu odkazování se na rodiče.

Sass: 1, Less: 1, Stylus: 1

Jmenné prostory: Vytvořený mixin je možné umístit pod identifikátor, respektive selektor, a to zajistí, že nebude v rozporu s jiným mixinem z importované knihovny [21].

Sass: 0, Less: 1, Stylus: 0

Podmíněné jmenné prostory: Jestliže má jmenný prostor v rámci své definice specifikovanou podmínku, jeho mixiny jsou použity pouze pokud je tato podmínka splněna.

Sass: 0, Less: 1, Stylus: 0

Klíčové slovo „!important“: Ne často používané klíčové slovo *„!important“* je možné jednoduše automaticky přidat ke každé definované vlastnosti v mixinu, a to pouhým jeho zapsáním ihned za samotné volání mixinu.

Sass: 0, Less: 1, Stylus: 0

Parametry: Mixiny mohou přijímat hodnoty jako argumenty, které jim jsou předány při vkládání mixinu a následně jsou v rámci mixinu dostupné jako proměnné [8].

Sass: 1, Less: 1, Stylus: 1

Výchozí hodnota parametrů: Mixiny mohou také specifikovat výchozí hodnoty pro své argumenty pomocí běžné syntaxe proměnných [8]. Pokud při vkládání mixinu nebude argument předán, bude místo něj použita právě jeho výchozí hodnota [8].

Sass: 1, Less: 1, Stylus: 1

Pojmenované parametry: Při volání mixinu je možné předávat hodnoty parametrů na základě jejich jména namísto pozice [21]. Na kterýkoliv parametr může být odkazováno jménem, a zároveň nemusí být v žádném zvláštním pořadí [21].

Sass: 1, Less: 1, Stylus 0

Proměnlivý počet parametrů: Název této možnosti mluví sám za sebe. Jedná se o schopnost mixinu přebírat předem neznámé množství parametrů. Zároveň je možné prvních N parametrů specifikovat a zbytek ponechat právě jako proměnlivý.

Sass: 1, Less: 1, Stylus: 0

Proměnná s parametry: Všechny předané argumenty jsou následně v rozsahu mixinu dostupné jako jedna proměnná.

Sass: 1, Less: 1, Stylus: 1

Shoda parametrů: Tento nepříliš přesný název představuje funkcionalitu, která umožňuje měnit chování mixinu na základě předaných parametrů. Prakticky jde o vytvoření více mixinů se stejným názvem, ale odlišným obsahem, a hlavně odlišnými parametry. Následně se dle shody předaných parametrů při volání mixinu použije příslušná varianta obsahu.

Sass: 0, Less: 1, Stylus: 0

Návratové hodnoty: Proměnné a mixiny definované uvnitř mixinu jsou dostupné k použití v celém rozsahu úrovně ze které je daný mixin volán [21]. Proměnné definované v mixinu mohou tedy bez problému sloužit jako návratové hodnoty [21]. To umožňuje vytvořit mixin, který může být použit téměř jako funkce [21].

Sass: 0, Less: 1, Stylus: 0

Vzájemné mísení: Definice mixinu může zahrnovat volání jiného mixinu.

Sass: 1, Less: 1, Stylus: 1

Rekurze: Mixin je schopen ve své definici volat nejen jiný mixin, ale i sám sebe.

Sass: 1, Less: 1, Stylus: 1

Podmínky: Schopnost vytvářet mixin doplněný o podmínku. K použití takového mixinu dojde následně při jeho volání skutečně pouze tehdy, je-li splněna určitá podmínka.

Sass: 0, Less: 1, Stylus: 0

Obsahové bloky: Krom předávání hodnot pomocí parametrů lze při volání mixinu předat rovnou celý blok CSS vlastností, který poté může být v obsahu mixinu vložen kdekoliv. Navíc je možné tento blok předat zároveň s libovolnými parametry, nikoliv místo nich.

Sass: 1, Less: 0, Stylus: 1

4.2.3.5 Shrnutí

Mixiny patří mezi přední funkci CSS preprocesorů, a to se samozřejmě odráží na počtu možností, které ji doprovází. Jelikož Sass, Less a Stylus přistupují k samotné implementaci mixin zcela odlišně, nelze pominout výrazné rozdíly v základní syntaxi. Vymezení rozsahu je hned po syntaxi další část, kde opět dochází k razantní odlišnosti zpracování mixin. Nicméně do této doby je nemožné na základě těchto dvou prvků označit provedení jednoho zástupce za lepší, poněvadž jsou jednotlivá řešení prakticky rovnocenná a případné výhody vyplývají z osobních preferencí. Přístup k proměnným je do jisté míry stejný, ale Less jeho možnosti posouvá o něco dál.

Detailní porovnání společných funkcionalit mixinů by mohlo být opravdu rozsáhlé. Ovšem skutečně relevantním rozdílem, na kterém při obsáhlosti mixinů záleží, je které z jednotlivých možností každý z CSS preprocesorů nabízí. Proto je součástí porovnání bodovaný výčet celých šestnácti možností rozšiřujících chování mixinů. V tomto ohledu je zdaleka nejpropracovanější Less. Je však nutné vždy brát ohled na to, které podporované možnosti jsou nezbytné pro efektivní práci. Ne všechny totiž musí být nutně důležité a je tedy vhodné zvážit, jestli ty, které Less nabízí navíc mezi ně patří.

Shrnutí průběhu bodování CSS preprocesorů v možnostech funkce mixinů zobrazuje tabulka č. 5. S nejmenším možným rozdílem se na první místo dostává Sass.

MIXINY			
	Sass	**Less**	**Stylus**
Hlavní možnosti:			
Základní syntaxe	1	1	1
Vymezení rozsahu	1	1	1
Přístup k proměnným	1	3	2
Další možnosti:			
Selektory	1	1	1
Odkaz na rodiče	0	1	0
Jmenné prostory	0	1	0
Podmíněné jmenné prostory	0	1	0
Klíčové slovo „!important“	1	1	1
Parametry	1	1	1
Výchozí hodnoty parametrů	1	1	0
Pojmenované parametry	1	1	0
Proměnlivý počet parametrů	1	1	1
Proměnná s parametry	0	1	0
Shoda parametrů	0	1	0
Návratové hodnoty	1	1	1
Vzájemné mísení	1	1	1
Rekurze	1	1	1
Podmínky	0	1	0
Obsahové bloky	1	0	1
Celkem bodů	**13**	**12**	**12**

Tabulka 5: Souhrn bodování funkce mixinů [vlastní zpracování]

4.2.4 Extend

Mixiny jsou první funkcí ze dvou, která vybočuje z typicky známých programových konstruktů. Tou druhou, u které toto tvrzení platí dvojnásob, je funkce extend. Její vznik úzce souvisí s principy používání samotného CSS, a proto se jedná o velice specifické zasazení v CSS preprocesorech, a nikoliv o převzatý konstrukt. S tím souvisí oproti ostatním funkcím větší obsáhlost všemožných možností, které extend nabízí. Samozřejmě čím více dostupných možností, tím pravděpodobněji se budou pochopitelně jednotlivé způsoby implementace lišit.

Porovnání funkce extend úplně stejně jako v případě mixin mírně vybočuje z předem stanovených pravidel. První dvě části, mezi které patří syntaxe a shoda selektorů, jsou bodované v souladu se zavedeným způsobem. A to z toho důvodu, že popisují stěžejní rozdíly v chování funkce extend. Následujících několik dalších možností je pak už pouze představeno kratším popisem vystihující jejich účel. U některých z nich může docházet taktéž k určitým rozdílům, který jsou ovšem většinou minimální a je tak mnohem více vhodné sledovat samotnou přítomnost či nepřítomnost oné možnosti u jednotlivých CSS preprocesorů. Proto je v této části upraveno rozdělování bodů. Každý CSS preprocesor získává za podporu konkrétní možnosti 1 bod. Pakliže danou možnost nepodporuje, získává 0 bodů, a to i za situace, že umožňuje stejného výsledku docílit úplně jiným řešením.

4.2.4.1 Syntaxe

Hlavní syntaktický rozdíl v základním používání funkce extend se tentokrát objevuje pouze mezi dvojicí zástupců. Mezitím, co Stylus sám označuje Sass za svou inspiraci a přebírá jeho podobu zápisu, Less zavádí stejný princip po svém.

Obdobně jako u jiných konstrukcí jazyka, zavádí Sass a Stylus pro extend novou direktivu. Intuitivně se jedná o direktivu *„@extend“*, za kterou následuje specifikace selektoru, ke kterému se vztahuje. Příslušný selektor CSS pravidla bude seskupen se specifikovaným selektorem a cíleně tak obsáhne jeho všechny CSS vlastnosti. Direktiva *„@import“* může být definována pouze uvnitř deklaračního bloku CSS pravidla.

Oproti tomu Less přichází s odlišným řešením, které má k dispozici rovnou dvě varianty. Zaprvé je možné připojit dvojtečku (:) s klíčovým slovem *„extend“* bezprostředně za jakýkoliv selektor. Následují kulaté závorky, v nichž je uveden cílový selektor určený pro seskupení. Druhý způsob se poněkud blíží direktivě, jenž používá Sass a Stylus. Jedná se o zápis, který je nutné definovat naopak uvnitř CSS pravidla. Je vytvořen spojením symbolu dolaru (&) a celé první varianty. Prakticky jde pořád o připojení k selektoru, ne však přímo ale pomocí odkazu na rodiče.

Jestliže je žádoucí rozšíření (extend) složeného selektoru, tedy selektoru složeného z více jednotlivých selektorů, nastává jistá komplikace. Řešení, které nabízí Less, představuje jistou výhodu. V rámci jeho první varianty zápisu může mít každý z dílčích selektorů bez problému připojen výraz v podobě klíčového slova *„extend“*. Ve chvíli, kdy Sass a Stylus takovou možnost nepodporují, je zapotřebí dosáhnout stejného výsledku poněkud zdlouhavější cestou. Především je nutné složený selektor rozdělit na samostatná CSS pravidla, respektive separovat ty selektory, u kterých je potřeba extend aplikovat. Uvnitř všech výsledných CSS pravidel pak musí být zvlášť definovány odpovídající direktivy *„@extend“*.

Navržená syntaxe pro Sass a Stylus je identická. Pozitivní odlišností se vyznačuje Less, a to konkrétně v poskytnutí více variant syntaxe pro dvě různá místa využití. Na základě toho je bodové hodnocení CSS preprocesorů u možnosti „syntaxe“ následující:

Sass: 2, Less: 3, Stylus: 2

4.2.4.2 Shoda selektorů

Extend je užitečný pro sdílení deklaračních bloků mezi více selektory bez nutnosti duplikování CSS vlastností. Všechny selektory zahrnuté do sdílení jsou ve výsledku seskupeny. Určení příslušných selektorů je předmětem syntaxe funkce extend. Obecně je však potřebný selektor zapsán v konkrétním místě za výraz specifikující použití daného selektoru pro rozšíření. Nicméně chování CSS preprocesorů se v této zdánlivě jednoznačné akci opět rozchází. Pokud se definovaný selektor objevuje v dokumentu vícekrát, Sass zahrne do rozšíření všechny výskyty včetně jeho zanořených selektorů, tedy potomků. Stejného efektu dokáže Less docílit přidáním

klíčového slova *„all“* za definovaný selektor, jelikož ve výchozím chování zahrnuje pouze selektor, který je shodný se zapsanou definicí [19]. Mít možnost výběru se rozhodně dá považovat za výhodu [19]. Oproti tomu Stylus funguje téměř stejně jako Sass. Sice taktéž zahrne do rozšíření veškeré potomky definovaného selektoru, ale pouze ty, které jsou zapsány pomocí hnízdění.

Problém shody selektoru v definici funkce extend a skutečného selektoru, na který má odkazovat, nekončí u jeho potomků. Při hledání těchto dvou selektorů Less připouští pouze přesnou schodu, respektive identickou formu zápisu. Různě zapsané selektory, ačkoliv stejného významu, se nepovažují za shodné. Jedinou výjimkou jsou uvozovky (jednoduché, dvojité, bez uvozovek) v rámci selektoru založeném na atributu, kde Less rozpozná stejný význam a shodu povoluje [21]. Na druhou stranu Sass nezavádí žádná striktní pravidla pro určení shody, ale sám vyhodnocuje, kdy jsou si dva selektory rovné, a to i v případě, že se nejedná o úplně přesnou shodu ve smyslu zápisu. Je tak schopen spojit dva selektory, přičemž jeden z nich je součástí komplexnějšího složeného selektoru. Nakonec Stylus si bere příklad z přísné shody selektorů a volí stejné zpracování jako Less.

Bodové hodnocení CSS preprocesorů je u možnosti „selektory“ následující:

Sass: 2, Less: 3, Stylus: 1

4.2.4.3 Další možnosti

Definice uvnitř direktiv: V případě, že je extend definován kdekoliv uvnitř bloku některé z jiných direktiv, jako je například *„@media“*, bude selektor zvolený za rozšíření hledán nikoliv v celém dokumentu, ale pouze v tomto bloku. Uvnitř dvou různých *„@media“* direktiv je možné volat funkci extend se stejným selektorem jehož obsah může být v rámci každého bloku odlišný.

Sass: 0, Less: 1, Stylus: 1

Detekce duplikací: Při seskupování selektorů je funkce extend natolik chytrá, aby zabránila vytváření zbytečným duplikacím [8]. Jeden selektor se tak nemůže ve výsledku objevit hned dvakrát za sebou, ale bude automaticky použit pouze jednou.

Sass: 1, Less: 0, Stylus: 1

Komplexní selektory: Třída není jediným použitelným selektorem pro extend. Lze rozšiřovat i složitější selektory obsahující například pseudotřídy, specifikaci atributů a jiné pokročilejší výrazy.

Sass: 1, Less: 1, Stylus: 1

Vnořené selektory: Možnost rozšířit selektor o vnořené selektory. Respektive funkci extend lze předat jako parametr selektor ve formě rodiče a potomka.

Sass: 0, Less: 1, Stylus: 1

Nepřekládané selektory: Kterýkoliv selektor je možné označit za takzvaného zástupce. Takový selektor může být běžným způsobem použit jako rozšíření jiného selektoru, nicméně sám o sobě není přeložen do CSS výstupu.

Sass: 1, Less: 0, Stylus: 1

Vícenásobnost: V rámci selektoru je možné volat extend vícekrát. Selektor tak může rozšiřovat více než jeden selektor a tím pádem získá CSS vlastnosti každého z nich.

Sass: 1, Less: 1, Stylus: 1

Řetězení: Pakliže selektor rozšiřuje jiný selektor, který stejně tak sám rozšiřuje další selektor, dochází k takzvanému řetězení. První selektor je pak rozšířen nikoliv o jeden, ale o dva selektory, přičemž druhý z nich je převzatý právě na základě řetězení z toho prvního.

Sass: 1, Less: 1, Stylus: 1

Klíčové slovo „optional“: Jestliže je extend použit pro rozšíření neexistujícím selektorem, dojde za normálních okolností k chybě [8]. Zápisem klíčového slova *„!optional“* přímo za selektor specifikovaný jako parametr se celá definice stává nepovinnou a rozšíření se provede pouze v případě, že je daný selektor skutečně v dokumentu zaveden.

Sass: 1, Less: 0, Stylus: 1

Interpolace: Pro rozšíření mohou být použity selektory, které obsahují interpolaci proměnné.

Sass: 1, Less: 0, Stylus: 1

4.2.4.4 Shrnutí

Extend je funkce, která by se dala s ohledem na propracovanost zařadit hned za mixiny. Výčet jejích možností je o něco menší, ale v porovnání s ostatními funkcemi CSS preprocesorů se zkrátka řadí mezi nejobsáhlejší. I přesto je však její zpracování všemi třemi zástupci více než podobné.

V základním chování představuje Sass pro Stylus značnou inspiraci, a proto je jejich syntaxe prakticky totožná. Naopak Less zavádí odlišnou syntaxi, a to rovnou ve dvou variantách, což mu v určitých situacích zajišťuje výhodu. Ve shodě selektorů se již objevuje rozdílů více. Ačkoliv se jedná spíše o implementační detaily, mění fungování funkce extend.

Porovnání pokračuje popisem devíti různých možností rozšiřujících extend ve způsobu používání. Zdaleka ne všechny jsou u každého CSS preprocesoru podporovány. Mezi nejdůležitější a nejpoužívanější prvky patří vnořené selektory, vícenásobnost a řetězení, přičemž Sass první z nich vůbec nenabízí. Užitečnou možností může být i definice uvnitř direktiv, kterou sice Sass jako ostatní zavádí, včetně popisu v oficiální dokumentaci, ale ve spoustě typických příkladů končí její aplikování chybou.

Průběh bodování CSS preprocesorů v rámci funkce extend shrnuje tabulka č. 6. Na první místo se s minimálním náskokem dostává Stylus.

EXTEND			
	Sass	**Less**	**Stylus**
Hlavní možnosti:			
Syntaxe	2	3	2
Shoda selektorů	2	3	1
Další možnosti:			
Definice uvnitř direktiv	0	1	1
Detekce duplikací	1	0	1
Komplexní selektory	1	1	1
Vnořené selektory	0	1	1
Nepřekládané selektory	1	0	1
Vícenásobnost	1	1	1
Řetězení	1	1	1
Klíčové slovo „!optional“	1	0	1
Interpolace	1	0	1
Celkem bodů.	**11**	**11**	**12**

Tabulka 6: Souhrn bodování funkce extend [vlastní zpracování]

4.2.5 Import

Rozdělení kódu na menší části výrazně zvyšuje jeho organizaci a spravovatelnost. Vždy je vhodné podobné bloky kódu seskupit do samostatných souborů, roztřídit do příslušných složek a následně je takzvaně importovat do jednoho hlavního souboru. Funkce import je součástí i standardu CSS, nicméně její každé použití vyžaduje vytvoření HTTP požadavku, což je nežádoucí chování z důvodu zpomalování načítání frontendu při větším množství těchto požadavků. Import v případě CSS preprocesorů tímto problémem netrpí, a proto se stal jejich důležitou součástí.

Podobu syntaxe není potřeba dlouze rozebírat, protože Sass, Less a Stylus přebírají zápis ze standardu CSS bez rozdílů. Ten začíná direktivou *„@import“*, za níž v uvozovkách následuje cesta k importovanému souboru. Základ zůstává stejný a k

odlišnostem v syntaxi dochází až u drobnějších doplňujících možností rozšiřujících chování funkce import.

4.2.5.1 Určení souboru

Všichni tři zástupci staví svůj preprocesorový jazyk jako určitou nadmnožinu CSS. To znamená, že každý validní CSS zápis je validním CSS zápisem i v kterémkoliv preprocesorovém jazyce. Jelikož Sass, Less a Stylus používají pro sovu funkci import stejnou syntaxi jako standard CSS, je nutné rozlišit, v jakých případech se jedná o funkci klasického CSS a kdy o funkci CSS preprocesoru. K případu, kdy při překladu nenastane importování daného souboru, ale naopak se ve výsledném CSS objeví CSS zápis direktivy *„@import“*, dochází za určitých okolností, které mají CSS preprocesory jednoznačně definovány. Tyto okolnosti jsou celkem 4 a platí u všech tří zástupců. Jedná se o následující podmínky:

- Jestliže přípona importovaného souboru je *„css“*.
- Jestliže název importovaného souboru začíná *„http://“*.
- Jestliže název importovaného souboru je zapsán do direktivy *„url()“*.
- Jestliže je direktiva *„@import“* doplněna o podmínku *„media query“.*

U první podmínky se mírně odlišuje Sass, který navíc na výstupu automaticky zapisuje název CSS souboru do direktivy *„url()“*. Druhá podmínka se chová identicky. Třetí funguje opět bez rozdílu, ať už je importován soubor CSS nebo písmo pomocí URL odkazu. Poslední podmínku ne zcela zvládá Stylus. Bohužel zvládá v kombinaci s importem zpracovat pouze nejjednodušší výrazy *„media query“* a složitější výrazy složené z více požadavků ignoruje. Nakonec poměrně zajímavou podmínku zavádí jako jediný Less. Ta říká, že pokud je přípona importovaného souboru jiná než *„css“* a *„less“*, bude tento soubor naimportován běžným způsobem jako typický Less soubor.

Pokud žádná z výše uvedených podmínek není splněna a přípona importovaného souboru je *„scss“* nebo *„sass“*, potom budou importovány soubory Sass nebo SCSS [8]. Pokud není uvedená žádná přípona, Sass se pokusí automaticky najít soubor s daným názvem a příponou *„scss“* nebo *„sass“* a importuje jej [8]. Úplně stejně

funguje Less a Stylus, kteří však pochopitelně hledají soubory s vlastními příponami, tedy *„less“* a *„styl“*.

Jednou z užitečných metod pro určení cesty k importovanému souboru je použití interpolace proměnné. Cestu k souborům je například pak možné uložit do jedné proměnné, na kterou se následně lze opakovaně odkazovat při specifikování cest a názvů souborů pomocí její interpolace. Takovou možnost nabízí pouze Less a Sass. Ovšem Sass tuto metodu bohužel omezuje jen na použití CSS importů v direktivě *„url()“*.

Dalším užitečným prvkem, který usnadňuje práci s funkcí import, je takzvaný „globbing“. Ten umožňuje v cestě k importovanému souboru použít určité zástupné znaky, které vytvoří jakousi souborovou masku. Na základě zápisu této masky je možné v rámci jediné direktivy *„@import“* jednoduše importovat více souborů najednou. Například všechny soubory z konkrétní složky nebo soubory, jejichž název začíná vybraným textovým řetězcem. Stylus je v tomto případě jediným CSS preprocesorem, který „globbing“ podporuje.

Po uvážení uvedených odlišností je bodové hodnocení CSS preprocesorů u možnosti „určení souboru“ následující:

Sass: 2, Less: 3, Stylus: 2

4.2.5.2 Částečné soubory

Co nejlepší rozdělení struktury kódu do více souborů je rozhodně žádoucí. Proto vznikají samostatné soubory, které udržují jednu konkrétní část kódu, například pouze definice všech dostupných proměnných. Takový soubor je importován do hlavního souboru a dá se tak považovat za jakýsi soubor částečný. Jelikož se po překladu hlavního souboru překladačem CSS preprocesoru stane díky importu jeho součástí, není potřeba jej ještě navíc překládat samostatně. Přesně to se však stane, pokud překladač přeloží celou složku souborů preprocesorového jazyka, nebo hromadně sleduje jejich změny a následně je automaticky překládá. Jestliže bude přeložen pouze zmíněný hlavní soubor, jeho importované soubory přeloženy zvlášť nebudou. Nicméně v uvedeném příkladu hromadného překládání souborů k tomuto nežádoucímu problému dojde.

Pokud je k dispozici soubor SCSS nebo Sass, který je potřeba importovat ale není potřeba jej překládat do CSS souboru, je možné přidat na začátek názvu tohoto souboru podtržítko [8]. Tímto způsobem je Sass informován o jeho nepřeložení do normálního CSS souboru [8]. Navíc podtržítko není nutné připisovat do direktivy *„@import“*, ale pouze do samotného názvu souboru. V případě souboru *„_barvy.scss“* nebude po překladu vytvořen žádný soubor *„_barvy.css“*.

Bez nutnosti úpravy jména souboru řeší částečné soubory Less. Svým jedinečným přístupem pro změnu chování funkce import pomocí nastavitelných možností umožňuje docílit stejného výsledku jako Sass s podtržítkem. Zapsání klíčového slova „reference“ do kulatých závorek před samotným specifikováním souboru na místě jeho importování způsobí vynechání onoho souboru z překladu do CSS. Tato varianta je poněkud výhodnější, protože daný soubor se automaticky nestává částečným, možnost je tak více volitelná a nestává se závislou na jméně souboru. Stejně tak není v případě změn potřeba opakovaně měnit jeho název. Naopak Stylus částečné soubory nezavádí vůbec.

Na základě porovnání, je bodové hodnocení CSS preprocesorů u možnosti „částečné soubory“ následující:

Sass: 1, Less: 2, Stylus: 0

4.2.5.3 Jednorázové vložení

Nic nebrání tomu, aby byl jeden soubor importován na jedno místo hned několikrát. V lepším případě dojde pouze k duplikaci kódu a v případě horším i k výskytu chyb. Proto se importování souboru vícekrát vyskytuje jen ve specifických případech. Je tedy vhodné nějakým způsobem ohlídat, aby několikanásobné importování jednoho souboru nemohlo vůbec nastat.

S tímto problémem si umí CSS preprocesory jednoduše poradit. Importování souboru pouze jednou je dokonce výchozí chování pro Sass a Less. Pokus o vícenásobné importování pomocí základního zápisu selže. Ovšem Less navíc poskytuje i explicitní specifikování této výjimky, a to přidáním klíčového slova *„once“* do kulatých závorek mezi direktivu *„@import“* a název souboru. Jiným způsobem výjimku určuje Stylus, který k docílení stejného efektu zavádí vlastní

direktivu *„@require“*, jenž stačí použít namísto direktivy *„@import“*. Řešení, které nabízí Less, se zdá být uživatelsky nejpřívětivější.

Bodové hodnocení CSS preprocesorů u možnosti „jednorázové vložení“ je následující:

Sass: 1, Less: 3, Stylus: 2

4.2.5.4 Rozšířené možnosti

Klíčové slovo *„once“* pro specifikování jednorázového vložení není jediné, které Less aplikuje. Nabízí několik rozšíření CSS importu pro zajištění větší flexibility nad prací s externími soubory [21]. Doposud byly implementovány následující možnosti importu:

„

- reference: použije Less soubor bez zahrnutí do výstupu
- inline: zahrne zdrojový soubor do výstupu, nezpracuje jej
- less: považuje soubor za Less soubor nehledě na příponu souboru
- css: považuje soubor za CSS soubor nehledě na příponu souboru
- once: zahrne soubor pouze jednou (výchozí chování)
- multiple: zahrne soubor několikrát
- optional: pokračuje v překladu i pokud není soubor nalezen

“ [21]

Každé z klíčových slov se zapisuje do kulatých závorek přímo před název importovaného souboru v rámci direktivy *„@import“*. Je povoleno více než jedno klíčové slovo pro jeden import a pro jejich oddělení se musí použít čárky [21]. Jedná se o hezké řešení, které může postupně snadno přidávat nové možnosti. Ovšem jak tomu u poměrně hodně funkcí bývá, i tato je podporována pouze ze strany jednoho ze tří zástupců, kterým je zrovna v tomto případě Less. Přestože Sass a Stylus dokáží konkrétně možnosti *„once“* a „reference“ nahradit svými odlišnými řešeními, Less s klíčovými slovy obecně nabízí vhodnější a ucelenější způsob pro úpravu chování funkce import.

Jelikož nelze Less s ostatními porovnávat, je bodové hodnocení CSS preprocesorů u možnosti „rozšířené možnosti“ následující:

Sass: 0, Less: 3, Stylus: 0

4.2.5.5 Vnořené vložení

Přestože je většinou nejužitečnější zavádět importy na nejvyšší úrovni (zanoření) dokumentu, je možné jejich zanoření do CSS pravidel a *„media query“* pravidel [8]. Stejně jako u import na základní úrovni, dojde ke vložení obsahu importovaného souboru [8]. Nicméně importovaná pravidla budou zanořena na stejném místě jako původní import [8]. To znamená, že direktivu *„@import“* lze použít nejen v kořenové úrovni, ale také jako vnořenou uvnitř jiných selektorů a pravidel [21]. Direktivy, které jsou povolené pouze na základní úrovni dokumentu jako *„@mixin“* nebo *„@charset“*, nejsou povolené v souborech, které jsou importované jako vnořené [8].

I přesto, že vnoření funkce import nepatří mezi zrovna nejvyužívanější možnost v rámci importování souborů, její podporu bez rozdílu nepostrádá Sass, Less ani Stylus.

Implementace je přítomna u všech tří zástupců, a navíc bez jakýchkoliv odlišností, proto je bodové hodnocení CSS preprocesorů u možnosti „vnoření“ následující:

Sass: 1, Less: 1, Stylus: 1

4.2.5.6 Shrnutí

Importování v CSS preprocesorech je v mnoha směrech oproti standardu CSS odlišné. Ačkoliv se Sass, Less a Stylus na jeho konceptu používání v zásadě shodují, pár jednoznačných rozdílů se nakonec objevuje i zde. Direktiva *„@import“*, jakožto funkce CSS preprocesoru, se od standardu klasického CSS ve stylu jejího zápisu příliš neodlišuje. Pro specifikování importovaného souboru (názvu a cesty) umožňuje Less navíc plně využít interpolace proměnných a Stylus takzvaný „globbing“. Částečné soubory pravděpodobně patří mezi nejdůležitější porovnávanou možnost importu a bohužel Stylus jí nepodporuje. Jednorázové vložení neboli zamezení vícenásobnému importování jednoho souboru, je přítomné u všech tří zástupců, ale každý jej řeší po svém. Rozšířené možnosti jsou výborným způsobem pro změnu

chování importu souboru za pomoci klíčových slov, které definuje Less jako jediný, což určitě může představovat jistou výhodu. Nakonec vnořené vložení je schopnost všech tří CSS preprocesorů používat direktivu *„@import“* uvnitř CSS pravidel, tedy mimo kořenovou úroveň dokumentu.

Bodování CSS preprocesorů u jednotlivých možností funkce import je shrnuto v tabulce č. 7. Se značným rozdílem se na první místo po celkovém součtu bodů dostává Less.

IMPORT			
	Sass	**Less**	**Stylus**
Určení souboru	2	3	2
Částečné soubory	1	2	0
Jednorázové vložení	1	3	2
Rozšířené možnosti	0	3	0
Vnořené vložení	1	1	1
Celkem bodů	**5**	**12**	**5**

Tabulka 7: Souhrn bodování funkce import [vlastní zpracování]

4.2.6 Operátory

Možnost provádění alespoň základních matematických operací značně usnadňuje práci nejen při výpočtech složitějších layoutů. Díky operátorům není nutné hodnoty CSS vlastností počítat ručně, ale je možné namísto výsledku zapsat samotný tvar výpočtu. Nicméně Sass, Less ani Stylus nekončí u sčítání a operátory implementují v opravdu velké šíři. Stejně jako pro čísla zavádí operátory pro textové řetězce či barvy. Zpravidla lze dále použít například operátory logické, existenční, unární a spoustu dalších.

Jako u všech ostatních funkcí, i zde se zpracování operátorů odlišuje. Jednotliví zástupci CSS preprocesorů mohou vyhodnotit stejný výpočet (zápis) různě a výsledek se nemusí shodovat [22]. Jelikož způsobů použití operátorů je nespočetné množství, je velice pravděpodobné, že takových případů nebude málo. Porovnávat výsledky všech těchto způsobů je prakticky nemožné a především bezpředmětné.

Operátory se však řadí mezi základní funkce a aby tato funkce nezůstala jako jediná bez skutečného bodového porovnání, budou CSS preprocesory zhodnoceny dle výsledku jednoduché početní operace. Mezi nejpoužívanější jednotky v CSS patří zejména pixely a procenta. Jejich kombinace bude tvořit zadání výpočtu, jehož výsledek bude porovnán. Jedná se o zápis *„100px - 50% - 50%“*, jakožto hodnoty libovolné CSS vlastnosti. Očekávaným výsledkem je hodnota *„25px“*. Jako jediný se ke správnému výsledku dostává Stylus. Naopak Less provádí nejprve sčítání procent, které vede ke konečné hodnotě *„0px“*. Vůbec k žádné hodnotě se bohužel nepropočítá Sass, jenž zápis řeší chybovou hláškou o nekompatibilitě jednotek. V určitém slova smyslu jedná Sass přesně, protože pixely a procenta skutečně nejsou ekvivalentní [22]. Avšak z pohledu uživatelské přívětivosti preprocesorového jazyka by měla být možnost procentuálních výpočtů dostupná.

Bodové hodnocení CSS preprocesorů u možnosti „operátory“ je následující:

Sass: 1, Less: 2, Stylus: 3

4.2.7 Shrnutí

Všechny výše porovnané funkce patří mezi základní a zároveň se tak jedná o nejpoužívanější konstrukty CSS preprocesorů. Tyto funkce tvoří převážnou část důvodů používání CSS preprocesorů a jejich rozsah možností může mít pozitivní či negativní vliv na samotnou úroveň pokročilosti daného CSS preprocesoru. Proto by práce s nimi měla být co nejefektivnější a měla by přinášet co největší usnadnění. Při nutnosti výběru toho nejlepšího zástupce CSS preprocesorů je vhodné brát v potaz především právě rozdíly mezi základními funkcemi.

Jednoznačně nejviditelnějším rozdílem, který bez výjimky provází všechny funkce, je samozřejmě syntaxe. Zápis preprocesorového jazyka předchází úplně všemu a je hlavní náplní práce s CSS preprocesory. Způsob zápisu by tak měl být dostatečně vyhovující. To je ovšem pochopitelně těžké posoudit, a proto se volba podoby syntaxe odvíjí spíše od osobních preferencí kodéra. Je určitě pádným důvodem pro označení daného jazyka jako více či méně sympatickým. Nicméně Sass, Less ani Stylus nepoužívají styl zápisu, který by obecně výrazně znesnadňoval pracovní postupy. Proto by se rozdílům v syntaxi obecně neměla přikládat příliš velká

důležitost. Navíc zmíněné osobní preference jsou velmi často pouze návyk, který lze snadno kdykoliv změnit.

Mnohem důležitější rozdíly jsou v možnostech jednotlivých funkcí. V průběhu porovnání se často objevují v rámci funkcí menší dílčí funkcionality, které nabízí CSS preprocesor oproti ostatním navíc. Ovšem některé z nich mohou být v praxi mnohem užitečnější než jiné. Nelze tak brát za výhodu jednoduše vyšší počet podporovaných možností, ale je vhodné sledovat přítomnost těch opravdu podstatných, které by měly být ze strany kodéra požadovanou součástí CSS preprocesoru. Mimo poskytované možnosti dochází u některých funkcí k odlišnostem v jejich samotném principu, který může mít vliv na průběh řešení různých situací a především na výsledek tohoto řešení.

Tabulka č. 8 shrnuje bodový zisk CSS preprocesorů u všech základních funkcí při jejich porovnání. Po celkovém sečtení bodů obsazuje Less pomyslné první místo. Bodové rozdíly nejsou u jednotlivých funkcí příliš velké a lze tak říci, že všechny tři CSS preprocesory nabízí přibližně stejné množství možností. Jedinou výjimkou je funkce import, u které Less získává svůj značný bodový náskok. Nejlepším zpracováním této funkce bezpochyby disponuje Less. Nicméně její odlišnosti, na kterých body ostatní zástupci ztrácí, se nedají považovat za tolik zásadní, aby byl na jejich základě označen Less za jednoznačně lepší nástroj.

ZÁKLADNÍ FUNKCE			
	Sass	**Less**	**Stylus**
Proměnné	8	13	8
Hnízdění	5	2	5
Mixiny	13	12	12
Extend	11	11	12
Import	5	12	5
Operátory	1	2	3
Celkem bodů	**43**	**52**	**44**

Tabulka 8: Souhrn bodování základních funkcí [vlastní zpracování]

4.3 Porovnání nástrojů

Porovnávat CSS preprocesory z hlediska základních funkcí je nejdůležitější částí pro odhalení odlišností v poskytovaných možnostech. Přesto, že se funkce často liší v mnoha aspektech, ve výsledku se dají považovat téměř za rovnocenné. Pokud ani jeden z rozdílů tak nevede k přesvědčení, že jeden CSS preprocesor by mohl být pro použití vhodnější než ostatní, je možné začít porovnávat jednotlivé CSS preprocesory i mimo základní funkce.

Nabízí se porovnání pokročilých funkcí, jako například složitější programové konstrukty nebo práce s barvami. Nicméně tyto funkce jsou minimálně pro Sass, Less a Stylus samozřejmostí. Většina z nich je navíc zpracována velice podobně. Zajímavý rozdíl by však mohl nastat v případě porovnání dodatečných nástrojů ulehčujících vývoj. Tato kapitola představuje výčet několika z nich.

Většinu následujících nástrojů je možné v praxi vhodně nahradit jinými nástroji, a proto jejich chybějící implementace přímo v CSS preprocesoru není až tak závažnou nevýhodou, jak se může zprvu zdát. Porovnávat přítomnost nástroje u CSS preprocesoru by nebylo dostačující a objektivní. Z tohoto důvodu následuje pouze výčet nejpoužívanějších nástrojů, a nikoliv bodované porovnání.

Mezi dodatečné nástroje se řadí především:

- Odlaďování chyb v kódu
- Hromadná kompilace a sledování změn souboru
- Kešování výstupu CSS
- Zaokrouhlování matematických operací
- Konverze do vlastní syntaxe
- Interaktivní konzole
- Komprese a struktura výstupu CSS

4.4 Porovnání popularity

Vysoká popularita nemusí být nutně výsledkem vysoké kvality. Nicméně u webových nástrojů tomu tak bývá. Ty nejlepší patří z dobrého důvodu mezi nejpoužívanější a tím pádem i mezi populární. V takovém případě může znalost těch

nejvíce oblíbených nástrojů mít pozitivní vliv na profesní uplatnění. Zároveň je naopak v případě potřeby samozřejmě snazší kodéry s takovými znalostmi najít a začlenit do práce na týmovém projektu. Druhou významnou výhodnou populárního nástroje je pak samotný počet lidí, kteří s daným nástrojem pracují. Díky velké komunitě je vždy možné nástroj mnohem rychleji rozvíjet a je tak do určité míry postaráno o jeho zdárnou budoucnost.

Porovnávaný Sass, Less a Stylus patří mezi nejpopulárnější CSS preprocesory. V praxi se jedná téměř o jediné tři používané zástupce a výběr vhodného nástroje se ve většině případech omezuje právě na ně. Důvodem je z určitého pohledu prakticky identická škála nabízených funkcionalit. I přesto že jde o rovnocenné konkurenty, objevuje se spousta rozdílů, které mohou mít ať už pozitivní nebo negativní vliv na oblíbenost CSS preprocesoru u větší skupiny kodérů. Typicky se jedná například o výrazné odlišnosti v syntaxi, jenž je hodnocena dle osobních preferencí. Proto tato kapitola přibližuje rozdíly v popularitě. Porovnání je rozděleno na několik kategorií, ve kterých je možné, a především relevantní, oblíbenost CSS preprocesorů nějakým způsobem číselně vyjádřit a porovnat. Jednotliví zástupci porovnání získávají v každé z těchto kategorií bodové hodnocení (čím více tím lépe). Přidělování bodů probíhá následujícím způsobem:

- Nejoblíbenější získává 3 body.
- V oblíbenosti druhý v pořadí získává 2 body.
- Nejméně oblíbený získává 1 bod.
- Ve speciálním případě, kdy CSS preprocesor nemůže ostatním konkurovat, získává 0 bodů.

4.4.1 GitHub

Webových služeb pro online ukládání repositářů nejrůznějších projektů pomocí verzovacího nástroje Git je na trhu několik. Jednoznačně největším a nejpoužívanějším poskytovatelem takové služby je GitHub, kde jsou ve formě veřejných repositářů ukládány zdrojové kódy téměř většiny open source softwaru včetně CSS preprocesorů. Kterýkoliv repositář má možnost od každého uživatele získat ohodnocení v podobě hvězdičky. Počet hvězdiček pak v porovnání s ostatními

značí jakousi oblíbenost repositáře, respektive onoho softwaru. Jelikož je GitHub nejrozšířenější platformou svého typu, lze podle zmíněných hvězdiček prakticky pozorovat technologické trendy v jednotlivých oblastech.

Počet hvězdiček jednotlivých GitHub repositářů CSS preprocesorů ke dni 16. 3. 2018:

Sass: 11 174, Less: 15 430, Stylus: 8 949

Bodové hodnocení CSS preprocesorů u kategorie „GitHub" je následující:

Sass: 2, Less: 3, Stylus: 1

4.4.2 Node Package Manager

Jedním z nejrychleji se rozvíjejícím programovacím jazykem je v poslední době JavaScript. Jeho rozvoj probíhá především ve spojitosti s webovými aplikacemi. Pro Stylus je od jeho počátků JavaScript vývojovým jazykem. Nicméně překladač pro Sass a Less byl do stále oblíbenějšího jazyka JavaScript o něco později stejně tak přepsán. Přestože CSS preprocesory jsou nabízené ve více jazykových variantách, JavaScript momentálně patří mezi nejpoužívanější. Proto i v tomto případě je možné porovnat oblíbenost jednotlivých zástupců. Nejrozšířenějším běhovým prostředím pro jazyk JavaScript je platforma Node.js. A to ať už se jedná o jeho spouštění na straně serveru nebo klienta. Nejrůznější nástroje jsou poté pro Node.js dostupné jako balíčky, které má na starost správce balíčků jako je například takzvaný Node Package Manager (NPM). Mezi balíčky patří i CSS preprocesory, které je tak možné skrze NPM stáhnout. Právě počet stažení bude předmětem porovnání a ukazatelem popularity CSS preprocesorů.

Počet stažení jednotlivých balíčků CSS preprocesorů na NPM v období 16. 2. – 16. 3. 2018:

Sass: 8 175 696, Less: 4 190 114, Stylus: 2 197 149

Bodové hodnocení CSS preprocesorů u kategorie „NPM" je následující:

Sass: 3, Less: 2, Stylus: 1

4.4.3 Stack Overflow

Při zdánlivě neřešitelném problému je dotázání se o pomoc běžným a často jediným řešením. Největší vývojářskou komunitou pro sdílení znalostí je webová služba Stack Overflow. Ta umožnuje pokládání otázek nejen v oblasti programování, se kterými mohou ostatní uživatelé pomoci. Jednotlivé otázky spadají dle svého obsahu do odpovídajících kategorií, mezi které patří i CSS preprocesory. Čím více je na téma určitého nástroje položeno otázek, tím se předpokládá jeho větší popularita.

Počet položených otázek ohledně CSS preprocesorů ke dni 16. 3. 2018:

Sass: 13 560, Less: 6 267, Stylus: 730

Bodové hodnocení CSS preprocesorů u kategorie „Stack Overflow“ je následující:

Sass: 3, Less: 2, Stylus: 1

4.4.4 CSS frameworky

Dalším užitečným nástrojem pro výrazné usnadnění práce s klasickým CSS jsou často používané CSS frameworky. Ty na první pohled nemají s CSS preprocesory nic společného. Jejich výstupem je sada předem vytvořeného CSS, které je možné opakovaně používat v typických příkladech. Nicméně vzhledem k jejich složitosti jsou vytvářené za pomoci CSS preprocesorů. Mezi deset nejlepších patří Bootstrap, Foundation, UI Kit, Semantic UI, Skeleton, Bulma, Materialize, Pure, Material UI a Base. Ten CSS preprocesor, který je použit pro vývoj nejvíce z uvedených CSS frameworků, se z určitého hlediska dá považovat za nejpopulárnější. Ovšem některé více propracovanější CSS frameworky mohou používat více než jeden CSS preprocesor.

Počet CSS frameworků vyvíjených na daných CSS preprocesorech:

Sass: 7, Less: 4, Stylus: 0

Bodové hodnocení CSS preprocesorů u kategorie „CSS frameworky“ je následující:

Sass: 3, Less: 2, Stylus: 0

4.4.5 Aplikace třetích stran

Způsobů, jak nainstalovat a následně spouštět překlad CSS preprocesoru, je několik. Jednou z jednodušších možností je využití aplikací třetích stran, které veškerou problematiku používání CSS preprocesorů řeší [23]. Pro začátek používání CSS preprocesorů lze aplikace třetích stran jen doporučit, jelikož urychlí spoustu věcí [23]. Mezi sedm nejznámějších takových aplikací patří například Code Kit, Koala, Hammer, Live Reload, Prepros, Scout a Crunch. Zpravidla se takové aplikace nestarají jen o překlad preprocesorového jazyka, ale nabízí spoustu dalších přidružených funkcí. Zároveň je pro ně výhodné se neomezovat pouze na jednoho zástupce CSS preprocesorů a podporovat jich vícero. Proto nepřítomnost CSS preprocesoru v jakékoliv ze známějších aplikací značí jeho menší popularitu.

Počet z výše uvedených aplikací podporujících dané CSS preprocesory:

Sass: 6, Less: 5, Stylus: 3

Bodové hodnocení CSS preprocesorů u kategorie „aplikace třetích stran“ je následující:

Sass: 3, Less: 2, Stylus: 1

4.4.6 Webové prezentace

Samozřejmostí každého CSS preprocesoru je poskytnutí oficiálních webových stránek s návody k instalaci, a to především s co nejlepší dokumentací detailně popisující všechny možnosti použití. Relativně zajímavým číslem k porovnání, vyjadřujícím určitý aspekt popularity, by mola být návštěvnost těchto webů. Návštěvnost samozřejmě neodráží skutečný počet uživatelů CSS preprocesorů, nicméně jakousi míru popularity rozhodně naznačuje.

Počet zobrazení jednotlivých oficiálních webů CSS preprocesorů v období 1. 9 – 28. 2. 2018:

Sass: 5 920 000, Less: 1 760 000, Stylus: 610 000

Bodové hodnocení CSS preprocesorů u kategorie „návštěvnost webu“ je následující:

Sass: 3, Less: 2, Stylus: 1

4.4.7 Google Trends

Jednou z ověřených metod zjišťování aktuálních trendů je sledování velmi často vyhledávaných výrazů skrze internetové vyhledávače. Celosvětově nejpoužívanějším vyhledávačem je Google Search. Jeho trží podíl je natolik velký, že nejčastěji vyhledávané výrazy se dají opravdu označit za globální trendy. Nástroj Google Trends pak dává k dispozici přesná čísla jakožto zájem ve vyhledání konkrétního výrazu v Google Search za dané časové období. Podobně jako u návštěvnosti oficiálních webových stránek CSS preprocesorů by mohl být ten nevyhledávanější z nich označen za nejpopulárnější.

Pro měření vyhledávání CSS preprocesorů jsou jako vyhledávané výrazy použity jejich samotná jména. Jelikož může jméno CSS preprocesoru nabývat jiného významu, je měření omezeno pouze na kategorii „návrh a vývoj webu“. Průměrný relativní zájem ve vyhledávání CSS preprocesorů, respektive zástupců Sass, Less a Stylus v období 16. 3. 2017 – 16. 3. 2018:

Sass: 72 %, Less: 39 %, Stylus: 17 %

Bodové hodnocení CSS preprocesorů u kategorie „Google Trends“ je následující:

Sass: 3, Less: 2, Stylus: 1

4.4.8 Shrnutí

Větší popularita rozhodně není to, co by mělo při porovnávání a vybírání vhodnějšího CSS preprocesoru v první řadě rozhodovat. Rozdíly mezi funkcemi jsou skutečně to na čem záleží nejvíce. Nicméně popularita znamená značnou komunitu, která pomáhá daný nástroj stále rozvíjet a zajišťovat tak jeho lepší uplatnění a budoucnost. Popularita samozřejmě nemusí nutně odrážet kvalitu nástroje, avšak ve většině případech je rostoucí popularita přímo odůvodněna nabízenými funkcemi.

Tabulka č. 9 shrnuje průběh bodování CSS preprocesorů v jednotlivých kategoriích porovnání popularity. Krom jedné kategorie získává ve všech ostatních nejvíce bodů Sass, a proto se po celkovém součtu dostává na první místo.

POPULARITA			
	Sass	**Less**	**Stylus**
GitHub	2	3	1
Node Package Manager	3	2	1
Stack Overflow	3	2	1
CSS frameworky	3	2	0
Aplikace	3	2	1
Webové prezentace	3	2	1
Google Trends	3	2	1
Celkem bodů	**20**	**14**	**6**

Tabulka 9: Souhrn bodování popularity [vlastní zpracování]

5 Shrnutí výsledků

Skutečnost, že vznik jazyka Cascading Style Sheets (CSS), neboli kaskádových stylů, je důležitým bodem v historii vývoje webu, nelze popřít. Myšlenka oddělení struktury dokumentu a jeho specifikace po vzhledové stránce věci zůstává a stává se stále podstatnějším principem. Vzhled moderního webu je i dnes nadále výsledkem CSS. Nicméně mezi tím, co složitost vývoje webových prezentací a aplikací v minulosti razantně rostla a stále roste, stanovený model vývoje CSS jakožto vytváření rozsáhlých standardů konsorciem W3C příliš nestíhal reflektovat skutečné potřeby kodérů. Průběžně tak došlo ke vzniku spousty nedostatků jazyka CSS, mezi které patří především obtížná spravovatelnost a udržitelnost kódu.

Komplikovaný vývoj CSS skutečně zapříčinil vznik nástrojů zvaných CSS preprocesory, které se snaží nedostatky CSS odstranit a zároveň si ponechat jeho podobu. Nevhodný přístup k vývoji CSS se výrazně změnil k lepšímu a snaha o nápravu jeho nevýhod je viditelná. I přesto je použití většiny nových funkcí tohoto záměru nestabilní a v některých případech téměř nepoužitelné. Proto rozhodně po představení základních funkcí CSS preprocesorů, včetně jejich popisů použití v konkrétních praktických situacích, není o nevýhodách CSS žádný pochyb a stejně tak naopak nemůže být pochyb o užitečnosti CSS preprocesorů. Samotná problematika CSS preprocesorů nabízí spoustu dílčích komplexních témat. Avšak ty stěžení z nich, nutné k začátku používání tohoto nástroje, by pro nováčky neměly být příliš velkou překážkou.

Účelem CSS preprocesorů je především usnadnění práce, a proto by se výběru toho nejvhodnějšího z nich měla věnovat dostatečně velká pozornost. Výsledkem této práce je v prakticky orientované části detailní porovnání CSS preprocesorů Sass, Less a Stylus. Předmětem porovnání byly především základní funkce, které navzdory jejich stejnému pojmenování a myšlence skrývají nespočet rozdílů s nimiž se uživatel CSS preprocesorů dříve nebo později setká. Syntaktické rozdíly v zápisu funkcí se objevují prakticky neustále, nicméně jejich případné výhody a nevýhody vyplývají většinou z osobních preferencí kodérů a neměly by se tak považovat nikterak za zásadní. K velkému překvapení se ovšem funkce často neliší pouze

syntakticky, ale v samotné podstatě jejich chování, a to poměrně výrazně. Za takových okolností je opravdu v některých případech možné stanovit jedno z řešení za lépe zpracované. Jednoznačnou nevýhodou je pak situace, kdy je jeden CSS preprocesor oproti ostatním v některých z možností funkce ochuzen. Takové rozdíly jsou časté a je na ně mezi kodéry kladen značný důraz.

Velké množství rozdílů vyplyne z pečlivého prostudování oficiálních dokumentací CSS preprocesorů, které jsou zpracované skutečně přívětivě. Ty však zdaleka neprozrazují vše a spousta více i méně podstatných rozdílů byla objevena vyzkoušením třech stejných zápisů kódu ve všech vybraných CSS preprocesorech a porovnáním jejich výsledků.

V průběhu porovnání základních funkcí získal každý CSS preprocesor bodové ohodnocení na základě vhodnosti zpracování dané funkce vzhledem k ostatním zástupcům. **V konečném součtu veškerých bodů se s několika bodovým náskokem ukazuje Less jako nejlepší CSS preprocesor.** Je však nutné zdůraznit, že za zmíněným náskokem stojí pouze bodování funkce import, které by se mohlo v určitém ohledu zdát nepodstatné. Nicméně pokud by se tato skutečnost měla pominou, Less by stejně obsadil první příčku, ale již bez tohoto značného bodového rozdílu, který u funkce import získává. Na druhém místě se pak v porovnání dle součtu bodů nachází Stylus a na třetím a zároveň posledním místě Sass.

Tabulka č. 10 zobrazuje výsledný součet bodů jednotlivých CSS preprocesorů získaných v průběhu porovnání jejich základních funkcí.

ZÁKLADNÍ FUNKCE			
	Sass	**Less**	**Stylus**
Celkem bodů	**43**	**52**	**44**

Tabulka 10: Výsledné bodování porovnání základních funkcí [vlastní zpracování]

Přestože výsledkem porovnání je jednoznačné pořadí CSS preprocesorů, celkový rozsah jejich funkcí se dá označit přinejmenším za velice podobný. Proto je při výběru CSS preprocesoru nezbytné ujasnit kritéria, která jsou očekávána a podle kterých se bude případný výběr řídit.

Druhou formou porovnání CSS preprocesorů bylo porovnání popularity. Všudypřítomná prohlášení, že Sass je nejznámějším CSS preprocesorem se ukázala jako pravdivá. **Krom jedné kategorie se ve všech ostatních Sass ukázal jako nejstahovanější, nejnavštěvovanější či nejpoužívanější.** Výjimkou je prozatím webová služba GitHub, kde v počtu získaných hvězdiček vede Less. Je zajímavé sledovat, že CSS preprocesor, který je dle získaných bodů v porovnání základních funkcí vyhodnocen jako nejhorší, je i přesto následně naopak v případě porovnání popularity jednoznačně na prvním místě.

Tabulka č. 11 zobrazuje výsledný součet bodů jednotlivých CSS preprocesorů získaných v průběhu porovnání jejich popularity.

POPULARITA			
	Sass	**Less**	**Stylus**
Celkem bodů	**20**	**14**	**6**

Tabulka 11: Výsledné bodování porovnání popularity [vlastní zpracování]

V případě tří nejznámějších CSS preprocesorů nemusí zdánlivě první místo v porovnání popularity hrát příliš velkou roli. Nicméně pokud prokazatelně vyšší oblíbenost zapříčiní upřednostnění použití CSS preprocesoru jako součásti ostatních nástrojů, jakou jsou například CSS frameworky, může tato skutečnost výskytu CSS preprocesoru v potřebném nástroji být důležitým faktorem při jeho výběru.

6 Závěry a doporučení

Potřeba nasazení CSS preprocesorů do pracovních postupů roste společně se zvyšováním nároků na vývoj webových prezentací a aplikací. Jejich použití se stalo běžnou praxí nejen v profesionálním prostředí. Nicméně rozvoj CSS preprocesorů se v aktuální chvíli odráží především od změn v CSS. Při nejmenším Sass, Less a Stylus dosáhly z hlediska funkcí svých možností. Prakticky už není potřeba nic nového zavádět, ale pouze zlepšovat.

Přestože jsou CSS preprocesory v oblasti CSS nejpoužívanějším nástrojem, samotné CSS se velmi rychle postupně začíná objevovat v nejrůznějších podobách a prostředích, které je při nejmenším dobré začít sledovat, ne-li rovnou začít používat. V budoucnu totiž může jít o přímou náhradu za CSS preprocesory. Jmenovitě se jedná především o typ nástroje jako je například knihovna JSS, která umožňuje používání CSS jakožto zápisu v jazyce JavaScript.

Stejně tak na důležitosti nabývají nástroje používající čistě syntaxi CSS včetně jeho nejnovějších funkcí schválených dle posledních specifikací. Nové funkce pak nejsou zprvu často podporovány ze strany prohlížečů, a proto je právě tyto nástroje po dobu špatné podpory převádí do starších alternativních zápisů. Patří mezi ně například CSS Next a Post CSS.

Na úplný závěr je určitě vhodné doporučit taktéž nezapomenout sledovat směr vývoje samotného CSS ve verzi 3, které se rozhodně snaží přinést co nejvíce nových funkcí a zapracovat tak na svých typických problémech, ačkoliv se mu to ne vždy daří.

7 Seznam použité literatury

[1] MOZILLA AND INDIVIDUAL CONTRIBUTORS*. Learn to style HTML using CSS* [online]. 2017. [cit. 2017-10-27]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/CSS

[2] W3C. *HTML & CSS* [online]. 2016. [cit. 2017-11-02]. Dostupné z: https://www.w3.org/standards/webdesign/htmlcss

[3] LIE, Håkon Wium. *Cascading Style Sheets. Series of dissertations submitted to the Faculty of Mathematics and Natural Sciences, University of Oslo.* [online]. 2005. [cit. 2017-11-10]. ISSN 1501-7710. Dostupné z: http://www.wiumlie.no/2006/phd/

[4] V, Victor. *A Brief History of CSS* [online]. 2016. [cit. 2017-11-15]. Dostupné z: http://www.css-class.com/a-brief-history-of-css/

[5] HÅKON WIUM LIE a Bert BOS. *Cascading style sheets: designing for the Web.* 2nd ed. Harlow, Essex, England: Addison-Wesley, 1999. [cit. 2017-11-24]. ISBN 0201596253.

[6] JAVOREK, Honza. *CSS preprocesory: méně psaní, vyšší efektivita* [online]. 2011. [cit. 2017-12-02]. Dostupné z: https://www.zdrojak.cz/clanky/css-preprocesory-mene-psani-vyssi-efektivita/

[7] MAZINANIAN, Davood a Nikolaos TSANTALIS. *An Empirical Study on the Use of CSS Preprocessors.* In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER) [online]. IEEE, 2016, 2016, s. 168-178 [cit. 2017-12-15]. DOI: 10.1109/SANER.2016.18. ISBN 978-1-5090-1855-0. Dostupné z: http://ieeexplore.ieee.org/document/7476640/

[8] CATLIN Hampton, Natalie WEIZENBAUM a Chris CHRIS. *Sass* [online]. 2017. [cit. 2017-12-19]. Dostupné z: http://sass-lang.com/

[9] WODEHOUSE, Carey. *6 Reasons Sass Should Be In Every Front-End Developer's Toolbox* [online]. 2016. [cit. 2017-12-21]. Dostupné z: https://www.upwork.com/hiring/development/reasons-to-use-sass-css-preprocessor/

[10] MICHÁLEK, Martin. *Průvodce CSS preprocesory: jak vám vylepší pracovní postupy* [online]. 2014. [cit. 2017-12-29]. Dostupné z: https://www.vzhurudolu.cz/blog/14-css-preprocesory-3

[11] BROTON, Josh. *My SASS and CSS Best Practices* [online]. 2013. [cit. 2017-12-30]. Dostupné z: http://joshbroton.com/my-sass-less-css-practices-modularization-nesting-variables-mixins-etc/

[12] MAZZA, Marcelo. *Embracing Sass: Why You Should Stop Using Vanilla CSS* [online]. 2015. [cit. 2018-01-20]. Dostupné z: https://www.toptal.com/front-end/embracing-sass-why-you-should-stop-using-vanilla-css

[13] CIASTEK, Mirosław. *(S)CSS Best Practices That You Have Not Yet Known* [online]. 2017. [cit. 2018-01-24]. Dostupné z: https://medium.com/@mciastek/s-css-best-practices-that-you-have-not-yet-known-ba2f6329b5dd

[14] SAXENA, Rishabh. *Do I Need To Start Using CSS Preprocessors?* [online]. 2017. [cit. 2018-01-26]. Dostupné z: https://blog.zipboard.co/do-i-need-to-start-using-css-preprocessors-a72b525c180a

[15] BRADLEY, Steven. *Sass And LESS: An Introduction To CSS Preprocessors* [online]. 2012. [cit. 2018-02-06]. Dostupné z: http://vanseodesign.com/css/css-preprocessors/

[16] GUBE, Jacob. *6 Current Options for CSS Preprocessors* [online]. 2014. [cit. 2018-02-09]. Dostupné z: https://www.sitepoint.com/6-current-options-css-preprocessors/

[17] CHARBENEAU, Ed. *Why Bootstrap 4 Means Sass Has Won* [online]. 2015. [cit. 2018-02-14]. Dostupné z: https://developer.telerik.com/featured/why-bootstrap-4-means-sass-has-won/

[18] DEAN, Matthew. Less: *The World's Most Misunderstood CSS Pre-processor* [online]. 2015. [cit. 2018-02-19]. Dostupné z: https://getcrunch.co/2015/10/08/less-the-worlds-most-misunderstood-css-pre-processor/

[19] COYIER, Chris. *Sass vs. Less* [online]. 2012. [cit. 2018-02-23]. Dostupné z: https://css-tricks.com/sass-vs-less/

[20] HOLOWAYCHUK TJ. *Stylus* [online]. 2017. [cit. 2018-02-27]. Dostupné z: http://stylus-lang.com/

[21] SELLIER Alexis. *Less* [online]. 2017. [cit. 2018-03-13]. Dostupné z: http://lesscss.org/

[22] FIRDAUS, Thoriq. *CSS Preprocessors Compared: Sass vs. LESS* [online]. 2010. [cit. 2018-03-16]. Dostupné z: https://www.hongkiat.com/blog/sass-vs-less/

[23] JACKSON, BRIAN. *CSS Preprocessors – Sass vs LESS* [online]. 2017. [cit. 2018-03-20]. Dostupné z: https://www.keycdn.com/blog/sass-vs-less/

8 Seznam použitých obrázků

Obrázek 1: Zobrazení HTML elementů v prohlížeči bez použití CSS [vlastní zpracování] 8

Obrázek 2: Zobrazení HTML elementů v prohlížeči s použitím CSS [vlastní zpracování] 8

Obrázek 3: Popis částí syntaxe CSS pravidla [vlastní zpracování] 11

Obrázek 4: Proces zobrazení dokumentu prohlížečem [1] 13

9 Seznam použitých tabulek

Tabulka 1: Přehled dostupných CSS preprocesorů [vlastní zpracování] 28

Tabulka 2: Přehled základní informací o Sass, Less a Stylus [vlastní zpracování] .. 50

Tabulka 3: Souhrn bodování funkce proměnných [vlastní zpracování] 63

Tabulka 4: Souhrn bodování funkce hnízdění [vlastní zpracování] 68

Tabulka 5: Souhrn bodování funkce mixinů [vlastní zpracování] 75

Tabulka 6: Souhrn bodování funkce extend [vlastní zpracování] 81

Tabulka 7: Souhrn bodování funkce import [vlastní zpracování] 87

Tabulka 8: Souhrn bodování základních funkcí [vlastní zpracování] 89

Tabulka 9: Souhrn bodování popularity [vlastní zpracování] 96

Tabulka 10: Výsledné bodování porovnání základních funkcí [vlastní zpracování] 98

Tabulka 11: Výsledné bodování porovnání popularity [vlastní zpracování] 99

10 Seznam použitých příkladů

Příklad 1: Zápis HTML s použitím prezentačních prvků [vlastní zpracování] 10

Příklad 2: Zápis HTML v případě použití CSS [vlastní zpracování] 10

Příklad 3: Proměnná v CSS preprocesoru Sass [vlastní zpracování] 30

Příklad 4: Výsledné CSS po použití proměnné v CSS preprocesoru [vlastní zpracování] 31

Příklad 5: Hnízdění v CSS preprocesoru Sass [vlastní zpracování] 31

Příklad 6: Výsledné CSS po použití hnízdění v CSS preprocesoru [vlastní zpracování] 32

Příklad 7: Mixin v CSS preprocesoru Sass [vlastní zpracování] 33

Příklad 8: Výsledné CSS po použití mixinu v CSS preprocesoru [vlastní zpracování] 33

Příklad 9: Technika znovupoužití vlastností v CSS [vlastní zpracování] 34

Příklad 10: Extend v CSS preprocesoru Sass [vlastní zpracování] 35

Příklad 11: Soubor CSS s nulovými hodnotami vlastností [8] 36

Příklad 12: Import v CSS preprocesoru Sass [8] 36

Příklad 13: Výsledné CSS po použití funkce import v CSS preprocesoru [8] 36

Příklad 14: Operátory v CSS preprocesoru Sass [vlastní zpracování] 37

Příklad 15: Výsledné CSS po použití operátorů v CSS preprocesoru [vlastní zpracování] 38

Příklad 16: Způsob použití funkce import v CSS preprocesoru Sass [vlastní zpracování] 43

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2017/2018

Studijní program: Aplikovaná informatika
Forma: Prezenční
Obor/komb.: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Černý Václav	Pražská 1552, Náchod - Staré Město nad Metují	I1500348

TÉMA ČESKY:

CSS preprocesory jako efektivní nástroj pro tvorbu frontendu

TÉMA ANGLICKY:

CSS preprocessors as an efficient tool for creation of frontend

VEDOUCÍ PRÁCE:

doc. Ing. Filip Malý, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cílem bakalářské práce je zpracovat problematiku preprocesorů pro kaskádové styly (CSS), respektive tzv. CSS preprocesorů. Teoretická část je zaměřená na představení výhod klíčových vlastností této technologie. Část praktická obsahuje vlastní výzkum, zabývající se porovnáním tří nejpoužívanějších variant, jimiž jsou SASS, LESS a Stylus.

1. Úvod
2. Teoretická část
3. Vlastní výzkum
4. Závěr
5. Seznam použité literatury

SEZNAM DOPORUČENÉ LITERATURY:

Podpis studenta: ..... **Datum:** 3. 10. 2017

Podpis vedoucího práce: ..... **Datum:** 3. 10. 2017