



Pedagogická
fakulta
Faculty
of Education

Jihočeská univerzita
v Českých Budějovicích
University of South Bohemia
in České Budějovice

Jihočeská univerzita v Českých Budějovicích

Pedagogická fakulta

Katedra informatiky

**PostCSS a cssnext jako nástroje pro
transformaci a modularitu CSS JavaScriptem**

**PostCSS and cssnext, tools enabling
transformation and modularity of CSS using
JavaScript**

Bakalářská práce

Vypracoval: Jakub Jetleb

Vedoucí práce: PaedDr. Petr Pexa, Ph.D.

České Budějovice 2018

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH
Fakulta pedagogická
Akademický rok: 2016/2017

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Jakub JETLEB**
Osobní číslo: **P15664**
Studijní program: **B7507 Specializace v pedagogice**
Studijní obor: **Informační technologie a e-learning**
Název tématu: **PostCSS a CSSnext jako nástroje pro transformaci a modularitu CSS JavaScriptem**
Zadávající katedra: **Katedra informatiky**

Z á s a d y p r o v y p r a c o v á n í :

Cílem bakalářské práce je zpracovat problematiku nástrojů PostCSS a CSSnext, které jsou určeny pro transformaci a modularitu CSS JavaScriptem využitím proměnných a vkládaných importů. Tyto nástroje usnadňují práci při tvorbě front-end webových aplikací, díky standardizované syntaxi a modularitě např. rychleji kompilují. Autor se ve své práci zaměří na porovnání vývoje front-end webu s využitím těchto nových postprocesorových nástrojů s nástroji preprocesorovými (SASS, LESS) a také bez využití jakýchkoliv podpůrných nástrojů. Praktickou část bakalářské práce bude tvořit vlastní webová aplikace, realizovaná s využitím CSS postprocesorů, na níž bude technologie detailně prakticky představena. Dalším výstupem práce bude srovnání časové náročnosti vývoje, výsledná datová velikost kódu a obtížnost pro vývojáře z hlediska znalosti syntaxe.

Rozsah grafických prací: CD ROM

Rozsah pracovní zprávy: 40

Forma zpracování bakalářské práce: tištěná

Seznam odborné literatury:

1. MICHÁLEK, Martin. Vzhůru do CSS3 [online]. 2015 [cit. 2017-04-10]. ISBN 978-80-2650-8440-2.
2. POSTCSS. PostCSS - a tool for transforming CSS with JavaScript [online]. [cit. 2017-04-10]. Dostupné z: <http://postcss.org/>
3. THIROUIN , Maxime. Cssnext - Use tomorrow's CSS syntax, today. [online]. [cit. 2017-04-10]. Dostupné z: <http://cssnext.io/>
4. GRUNT. Grunt: The JavaScript Task Runner [online]. [cit. 2017-04-10]. Dostupné z: <https://gruntjs.com/>
5. Joyent, Inc. Node.js [online]. [cit. 2017-04-10]. Dostupné z: <https://nodejs.org/en/>

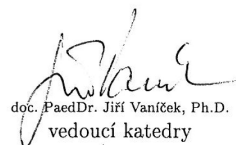
Vedoucí bakalářské práce: PaedDr. Petr Pexa, Ph.D.
Katedra informatiky

Datum zadání bakalářské práce: 24. dubna 2017

Termín odevzdání bakalářské práce: 30. dubna 2018



Mgr. Michal Vančura, Ph.D.
děkan



doc. PaedDr. Jiří Vaníček, Ph.D.
vedoucí katedry

V Českých Budějovicích dne 24. dubna 2017

Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne 8. července 2018.

Abstrakt / Anotace

Cílem bakalářské práce je zpracovat problematiku nástrojů PostCSS a CSSnext, které jsou určeny pro transformaci a modularitu CSS JavaScriptem využitím proměnných a vkládaných importů. Tyto nástroje usnadňují práci při tvorbě front-end webových aplikací, díky standardizované syntaxi a modularitě např. rychleji kompilují. Ve své práci se autor zaměří na porovnání vývoje front-end webu s využitím těchto nových postprocesorových nástrojů s nástroji preprocesorovými (SASS, LESS) a také bez využití jakýchkoliv podpůrných nástrojů. Praktickou část bakalářské práce bude tvořit vlastní webová aplikace, realizovaná s využitím CSS postprocesorů, na níž bude technologie detailně prakticky představena. Dalším výstupem práce bude srovnání časové náročnosti vývoje, výsledná datová velikost kódu a obtížnost pro vývojáře z hlediska znalosti syntaxe.

Klíčová slova

HTML, CSS, PostCSS, CSSnext, JavaScript

Abstract

The goal of bachelor thesis is to process problematic of PostCSS and cssnext tools, wich they are used for transformation and modularity of CSS by JavaScript with using inserted imports. These tools make easier work of front-end development and web applications, thanks to standardisation syntax and modularity they, for example, compile faster. The author of his work will focus on compare development front-end web with using these postprocessing tools with preprocessor tools (SASS, LESS) and also without using any helping tools. The practical part of bachelor thesis will create the own web application, done by using CSS postprocessors, on there will be this technology detailed and practically introduced. Next goal of work will be time difficulty of development, the final size of code and difficulty for developer terms of knowledge of syntax.

Keywords

HTML, CSS, PostCSS, CSSnext, JavaScript

Poděkování

Rád bych poděkoval PaedDr. Petru Pexovi, Ph.D., za odborné vedení při zpracování mé bakalářské práce a jeho cenné rady. Dále bych rád poděkoval své rodině, která mi umožnila studium na vysoké škole.

Obsah

1	Úvod	10
1.1	Východiska práce	10
1.2	Cíle práce	10
1.3	Metoda práce	11
2	Buildovací nástroje	12
2.1	Grunt	12
2.1.1	Instalace Gruntu	13
2.1.2	Soubory Gruntu	13
2.2	Gulp	13
2.2.1	Instace Gulpu	14
2.3	Prepros	14
2.3.1	Funkce a výhody	14
2.3.2	Založení projektu	16
3	PostCSS	17
3.1	Řešení problémů s PostCSS	18
3.2	Instalace	18
3.2.1	Instalace grunt-postcss	19
3.3	Konfigurace	20
3.4	Pluginy	21
3.4.1	Autoprefixer	21
3.4.2	postcss-easy-import	23
3.4.3	Font Magican	24
3.4.4	LostGrid	26
4	cssnext	29
4.1	Konfigurace cssnext	29
4.2	Funkce	29

4.2.1	Vlastní vlastnosti a proměnné	29
4.2.2	Vlastní vlastnosti a @apply	30
4.2.3	Upravená funkce calc()	31
4.2.4	Vlastní media queries	31
4.2.5	Rozsahy media queries	32
4.2.6	Vlastní selektory	32
4.2.7	Zanořování	33
4.2.8	Modifikace barev	35
5	Porovnání	36
5.1	Porovnání oproti CSS	36
5.1.1	Výhody PostCSS a cssnext	36
5.1.2	Nevýhody PostCSS a cssnext	36
5.2	Porovnání oproti preprocesorům	37
5.2.1	Časové porovnání kompilace	37
6	Praktická část	39
7	Závěr	43
	Seznam použité literatury a zdrojů	44
	Seznam obrázků	46
	Seznam ukázek kódu	47
8	Přílohy	48

1 Úvod

1.1 Východiska práce

V dnešní době jsou oblíbené CSS preprocesory, které rozšiřují možnosti psaní CSS stylů. Kód je díky nim čistější, srozumitelnější a znovupoužitelný. Jedny z nejpoužívanějších preprocesorů jsou SASS a LESS.

PostCSS a cssnext je potenciální konkurent těchto CSS preprocesorů. Rozdíl je už v momentě zpracování. Preprocesor (SASS, LESS, Stylus) předpracovává, což v důsledku znamená, že vytváří nový jazyk, který se kompiluje do CSS. PostCSS je postprocessor. Více či méně předpokládá, že to, co píšete, je současná nebo budoucí verze CSS. [1]

Další rozdíl je v modularitě. Preprocesory jsou monolity, jejichž všechny vlastnosti nikdy nevyužijete. PostCSS je modulární. Samo o sobě nic neumí, pro každou vlastnost si musíte doinstalovat plugin. To je samozřejmě také trochu nevýhoda, protože to vyžaduje režii na učení a prozkoumávání. [1]

1.2 Cíle práce

Cílem bakalářské práce je zpracovat problematiku nástrojů PostCSS a cssnext, které jsou určeny pro transformaci a modularitu CSS JavaScriptem využitím proměnných a vkládaných importů. Tyto nástroje usnadňují práci při tvorbě front-end webových aplikací, díky standardizované syntaxi a modularitě např. rychleji kompilují. Ve své práci se autor zaměří na porovnání vývoje front-end webu s využitím těchto nových postprocessorových nástrojů s nástroji preprocesorovými (SASS, LESS) a také bez využití jakýchkoliv podpůrných nástrojů. Praktickou část bakalářské práce bude tvořit vlastní webová aplikace, realizovaná s využitím CSS postprocesorů, na niž bude technologie detailně prakticky představena. Dalším výstupem práce bude srovnání časové náročnosti vývoje, výsledná datová velikost kódu a obtížnost pro vývojáře z hlediska znalosti syntaxe.

1.3 Metoda práce

V úvodu popíši, co jsou CSS preprocesory a jak fungují. Dále se zaměřím, jakými nástroji lze preprocesory zpracovávat a jaký nástroj k zpracování použiji. Poté popíši nástroje PostCSS a cssnext. Dále porovnáám tyto nástroje mezi sebou v souvislosti časové náročnosti, velikosti kódu a z hlediska syntaxe. Porovnání těchto nástrojů docílím pomocí názorných ukázek, které zveřejním pomocí webové aplikace na internetu, kde bude projekt veřejně dostupný.

2 Buildovací nástroje

Buildovací nástroje pomáhají s automatizací. Je to sada úkolů, které běží na projektových souborech. Ulehčují práci při opakujících se úlohách, jako jsou minifikace CSS souborů a javascriptových souborů, kompilace nebo kontrolování javascriptových souborů. Tyto buildovací nástroje pracují s pluginy, které dané úlohy vykonávají. Pro správu těchto úloh se používají konfigurační soubory, které nastavují chování těchto úloh. Například kompilace SCSS souboru do CSS souboru, nám může obstarávat plugin WATCH, která zaznamenává změny v souboru a následně SCSS soubor zkompiluje do CSS souboru. [2]

Každá stránka by měla mít dvě verze, vývojářskou a produkční. Vývojářská obsahuje všechny HTML, CSS, JS a obrázkové soubory, které jsou čistě zformátované a dá se s nimi pracovat. Produkční verze tyto soubory minifikuje a spojuje a optimalizuje soubory, jako jsou obrázky nebo svg. [2]

Mezi populární nástroje k implementaci buildovacího procesu jsou Gulp a Grunt, oba nástroje jsou v příkazovém řádku. Existují nástroje, které mají grafické rozhraní, které mohou být lehčí uchopit, ale mohou být méně flexibilní. [2]

2.1 Grunt

Využívá rozhraní příkazové řádky a k spuštění úloh Grunt používá soubor pod názvem Gruntfile. Grunt je původně vytvořený Benem Almanem v roce 2012 jako efektivní alternativa jednoduchého psaní a udržování JavaScriptových buildovacích procesů v jednom velkém souboru. [3] Je navržen jako nástroj pro tvorbu úloh v příkazovém řádku pro JavaScriptové projekty. [4]

Gruntové rozhraní příkazové řádky lze nainstalovat skrz balíčkovací systém NPM. Napsáním příkazu "grunt" do příkazové řádky, načte a spustí verzi Gruntu lokálně nainstalovanou v aktuálním adresáři. Díky čemuž můžeme spravovat více verzí Gruntu v jiných složkách a řídit si je, jak si přejeme. [3]

2.1.1 Instalace Gruntu

Abychom jsme mohli začít používat nástroj Grunt, musíme ho nejprve nainstalovat pomocí příkazové řádky. Příkazovým řádkem: [3]

```
npm install -g grunt-cli
```

Tímto si zajistíme, že tento nástroj nám bude fungovat z jakéhokoliv adresáře.

2.1.2 Soubory Gruntu

Abychom mohli používat Grunt ve svém projektu, potřebujeme dvě specifické soubory vytvořené v našem kořenovém adresáři, jmenovitě package.json a Gruntfile. [3]

- package.json - obsahuje metadata pro konkrétní projekt včetně názvu, verze, popisu, autora, licence a nejdůležitější závislosti. Všechny závislosti jsou v zaznamenány v sekci "dependencies" nebo "devDependencies". [3]
- Gruntfile - Buď JavaScriptový soubor nebo CoffeScript soubor pojmenovaný "Gruntfile.js" nebo "Gruntfile.coffee", který obsahuje kód ke konfiguraci úloh, načtení pluginů nebo vytvoření vlastních úloh. [3]

2.2 Gulp

Gulp je konkurence nástroje Grunt. Autorem je Eric Schoffstall a vznikl ve společnosti Fractal a s pomocí GitHub komunity. Gulp je o něco mladší nástroj než Grunt, který vznikl v roce 2013. Opět se jedná o nástroj, který pomáhá s automatizací úloh.

2.2.1 Instace Gulpu

Instalace je podobná jako u programu Grunt. Musíme mít prvně nainstalovaný Node.js a poté pomocí příkazové řádky:

```
npm install gulp -g
```

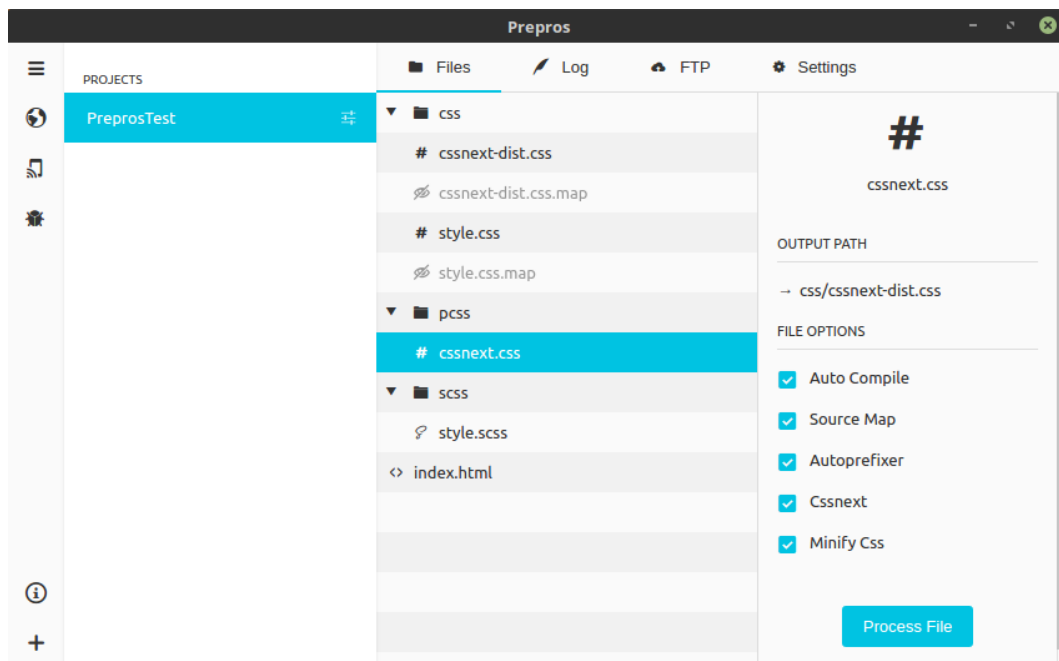
Tímto se nám Gulp nainstaluje globálně do našeho systému a můžeme ho spustit z jakéhokoliv adresáře.

2.3 Prepros

Prepros je vytvořen Subash Pathakem, je to nástroj, který dokáže automaticky řídit veškerý preprocessing, který potřebujeme. Je to otevřený software s grafickým rozhraním pro kompilaci. Dokáže kompilovat například LESS, SASS, Compass, Stylus, Markdown, CoffeScript apod. S Preprosem vícejazyčný preprocessing není problém. Prepros funguje na systému Windows a Mac. Pokud používáme pouze jeden preprocesor, nemusí to být tak obtížné, ale častěji se setkáme s tím, že používáme více preprocesorů najednou a věci se pak stávají komplikovanější. Preprocesory musíme nainstalovat a všechny jazyky musíme nastavit správně, aby kompilovaly naše soubory, což samozřejmě znamená více práce. [5]

2.3.1 Funkce a výhody

- Žádné závislosti. Stačí nainstalovat Prepros a tím máme vše hotové. Už nic nemusíme instalovat nebo konfigurovat. Nemusíme zapínat příkazový řádek. Stačí zapnout aplikaci a můžeme začít pracovat.
- Grafické rozhraní. Pokud nám nevyhovuje příkazová řádka.
- cssnext - Prepros nabízí možnost pracovat s nástrojem cssnext.



Obrázek 1: Ukázka grafického rozhraní Prepros

- Několik přizpůsobovacích úrovní. Můžeme si Prepros nakonfigurovat globálně pro všechny projekty, specifikovat pro každý projekt jednotlivě. Každý soubor může být manuálně nebo automaticky zkompilován. Záleží jen na našich potřebách.
- Minifikace a sjednocení JavaScriptu. Prepros dokáže zminifikovat a sjednotit naše JavaScriptové soubory v reálném čase, pokud změníme naše soubory v našem editoru.
- Optimalizace obrázků. Prepros dokáže optimalizovat obrázky formátu PNG, JPG a GIF jedním kliknutím, což nám může pomoci k rychlejšímu načítání stránky a celkově zlepšit stránku.
- Detekce a pozorování. Prepros sleduje změny v našich souborech a kompiluje je při tom. tom.

- Automatické obnovování prohlížeče. Prepros dokáže živě obnovovat náš prohlížeč kdykoliv nastane změna v souborech, ale musí se použít rozšíření pro prohlížeč. Tato funkce funguje ve Firefoxu, Chrome a Opeře.
- Automatické obnovování prohlížeče na více zařízení. Prepros podporuje testování naší webové stránky na několika zařízeních v naší síti.
- Notifikace úspěchu nebo neúspěchu kompilace. Prepros oznamuje hlášením, pokud se soubor zkompiloval v pořádku. Když se zkompile s chybou, oznámí chybové hlášení a záznam.

[5]

2.3.2 Založení projektu

1. Založení složky - Vytvoříme si složku s názvem projektu a vytvoříme si v ní vlastní projektovou strukturu.
2. Přidání projektu - Jednoduše přesuneme složku do Prepros okna.
3. Odfiltrování nepotřebných složek nebo souborů - Prepros ve výchozím nastavení sleduje všechny soubory. Proto můžeme po kliknutí pravým tlačítkem myši na složku nebo soubor zvolit filtrování, abychom zamezili sledování nepotřebných složek nebo souborů.
4. Nastavení kompilování souborů - Můžeme si manuálně nastavit kompilaci souborů. Možnost vybrat si výstupovou složku a název souboru, nebo jestli chceme generovat SourceMap.

[6]

3 PostCSS

Pokud řekneme PostCSS, mohou se tím myslet dvě věci.

1. Pluginový ekosystém poháněný tímto nástrojem.
2. Nástroj jako takový, to, co dostaneme, když napíšeme do příkazového řádku "npm install postcss"

[7]

PostCSS je nástroj pro transformaci CSS pomocí pluginů napsaných v Node.js. Na první pohled vlastně nic nedělá, je to ale nutná spodní vrstva pro fungování známějších a zajímavějších pluginů. Lze využít jako jednoduchý a rychlý preprocesor. [1]

Nástroj je Node.js modul, který rozebírá CSS na "Abstract syntax tree" zkráceně AST. AST poskytuje přímý API, který developeři mohou využít k psaní pluginů. To je vše, co PostCSS zatím dělá. Nemění CSS, ale plugin je toho schopný, ale také nemusí. PostCSS pluginy mohou v podstatě udělat cokoli s CSS. PostCSS může pohánět nekonečno druhů pluginů, které čtou a manipulují s našim CSS. Pluginy nemají žádnou sjednocující agendu. [7]

Rozdíl je už v momentě zpracování. Preprocesor (SASS, LESS, Stylus) před zpracovává, což v důsledku znamená, že vytváří nový jazyk, který se kompiluje do CSS. PostCSS je postprocesor. Více či méně předpokládá, že to, co píšete, je současná nebo budoucí verze CSS. Další rozdíl je v modularitě. Preprocesory jsou monolity, jejichž vlastnosti pravděpodobně nikdy všechny nevyužijeme. PostCSS je modulární. Samo o sobě nic neumí, pro každou vlastnost se musí doinstalovat plugin. To je samozřejmě také trochu nevýhoda, protože to vyžaduje režii na učení a prozkoumávání. [1]

K čemu takový PostCSS může sloužit? K řešení globálních CSS problémů, jako jsou automatické izolování selektorů v rámci komponent. Automatické globální CSS resety komponent, např. pro BEM zápis, automatické fallbacky pro obrázkové sety pro IE9. Dále pomáhá psát přehlednější CSS. Například s `precss`

pluginem je možné používat SASS funkce, jako jsou proměnné, zanořování a mixiny. Používání budoucího CSS dnes, kdy autoprefixer přidá předpony css vlastností nebo pluginy k analyzování kódu, které pomáhají detekovat chyby v CSS nebo pomáhají usměrnit zápis CSS, jak si nastavíme. [8]

3.1 Řešení problémů s PostCSS

Práce s PostCSS nám může připomínat, že CSS existuje k řešení problémů, a tudíž většina problémů má více řešení. Máme na výběr mezi alternativními řešeními, nebo si můžeme vytvořit vlastní řešení problému. Pokud máme PostCSS a vypořádáváme se s CSS, potřebujeme prvně problém. Podobně jako s Javascriptem, místo toho abychom vybrali masivní knihovnu, aniž bychom věděli, co se děje, tak prvně přemýšlíme o konkrétním problému, který je potřeba vyřešit. Takže zvažujeme existující řešení, kdy použijeme již existující řešení, nebo začneme pracovat na vlastním řešení. [7]

3.2 Instalace

Abychom mohli nainstalovat a používat nástroj PostCSS, potřebujeme prvně knihovnu Node.js. Při instalaci se nám automaticky nainstaluje i balíkový ekosystém npm. Po instalaci si můžeme zkontrolovat, zda jsme nainstalovali systém správně. Otevřeme příkazový řádek a pomocí příkazu, zkontrolujeme verzi, zdali máme Node.js a npm nainstalováno. [9]

```
node -v
```

Poté podobný příkaz pro npm:

```
npm -v
```

Nyní jsi už můžeme nainstalovat PostCSS a máme na výběr z několika Task runnerů, pod kterými PostCSS poběží.

- Grunt: `grunt-postcss`
- HTML: `posthtml-postcss`
- Stylus: `poststylus`
- Rollup: `rollup-plugin-postcss`
- Brunch: `postcss-brunch`
- Broccoli: `broccoli-postcss`
- Meteor: `postcss`
- ENB: `enb-postcss`
- Taskr: `taskr-postcss`
- Connect/Express: `postcss-middleware`

Ve své práci budu používat TaskRunner neboli buildovací nástroj Grunt.

3.2.1 Instalace `grunt-postcss`

Než nainstalujeme plugin PostCSS pro Grunt, musíme mít prvně Grunt nainstalovaný. Toto provedeme v příkazové řádce pomocí příkazu:

```
npm install -g grunt-cli
```

Tento příkaz nám pouze nainstaluje Gruntové příkazové rozhraní (CLI). Poté si připravíme nový grunt projekt. Plugin do projektu nainstalujeme přes příkaz:

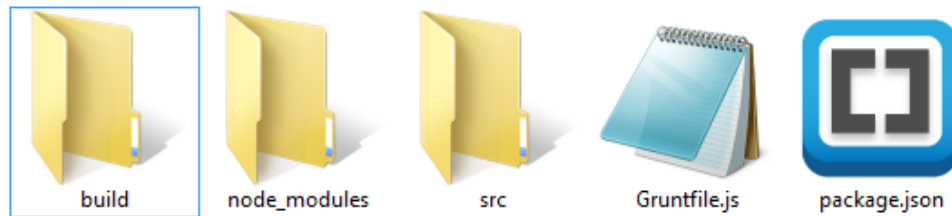
```
npm install grunt-postcss --save-dev
```

Poté přidáme do souboru Grunfile tento řádek kódu, aby se nám plugin načtl při spuštění Gruntu.

```
grunt.loadNpmTasks('grunt-postcss');
```

3.3 Konfigurace

Vytvoříme si složku a vytvoříme základní strukturu projektu.



Obrázek 2: Základní struktura projektu

Ve složce src budou veškeré soubory, se kterými budeme pracovat. Do složky build budeme kompilovat veškeré soubory, kromě HTML souborů. Pokud nebudeme používat žádný HTML preprocesor, jako je například Haml nebo Markdown. Otevřeme si složku v příkazovém řádku a inicializujeme si soubor package.json pomocí příkazu:

```
npm init
```

Tento příkaz se nás zeptá na několik otázek jako například název projektu, autor, licence, popis atd., a poté připraví soubor package.json.

```
1 {  
2   "name": "postcss-projekt",  
3   "version": "1.0.0",  
4   "description": "Inicializace postcss",  
5   "main": "Gruntfile.js",  
6   "dependencies": {},  
7   "devDependencies": {},  
8   "author": "Jakub Jetlebo"  
9 }
```

Listing 1: Inicializace souboru package.json

Poté přidáme postcss do našeho projektu grunt pomocí příkazu:

```
npm install grunt --save-dev
```

Tím se nám Grunt nainstaluje a přidá do souboru package.json jako závislost. Nyní už nainstalujeme samotné PostCSS pomocí příkazu:

```
npm install grunt-postcss --save-dev
```

v souboru package.json nám nyní přibyly závislosti.

```
1  "devDependencies": {  
2    "grunt": "^1.0.2",  
3    "grunt-postcss": "^0.9.0"  
4  },
```

Listing 2: Závislosti v souboru package.json

Připravíme si soubor Gruntfile.js, kde inicializujeme, jaké pluginy se budou spouštět a případně si je nastavíme podle potřeby.

3.4 Pluginy

3.4.1 Autoprefixer

Autoprefixer je plugin, který převádí CSS a přidává předpony experimentálních CSS vlastností od výrobců prohlížečů. Cíl toho pluginu je jednoduchý. Zapomenout, že nějaké předpony vlastností od výrobců prohlížečů existují. Nemusíme používat speciální jazyk jako SASS nebo si pamatovat, musíme použít mixin. [10]

```
1  ::placeholder {  
2    color: gray;  
3  }
```

Listing 3: Autoprefixer-01-vstup

Zkompiluje do:

```
1  ::-webkit-input-placeholder {
2    color: gray;
3  }
4  :-ms-input-placeholder {
5    color: gray;
6  }
7  ::-ms-input-placeholder {
8    color: gray;
9  }
10 ::placeholder {
11   color: gray;
12 }
```

Listing 4: Autoprefixer-01-výstup

Autoprefixer podporuje selektory jako jsou:

- `:fullscreen` - je CSS pseudo element, který reprezentuje element pokud je prohlížeč v režimu celé obrazovky. [11]
- `::selection` - CSS pseudo element, který reprezentuje styl označeného textu.
- početní funkci `calc()` - početní funkce, která nám umožní provádět výpočty na konkrétní CSS hodnoty. Může být využito kdykoliv - délku, frekvenci, úhel, čas, číslo, barvu. [12]

Jelikož je Autoprefixer postprocesor pro CSS, může se používat v kombinaci s preprocesory, jako jsou SASS, Stylus nebo LESS. Autoprefixer využívá nejnovější data ze stránky “Can I use” k přidání nezbytných předpon. Dále odstraňuje staré a nepotřebné předpony z našeho CSS. Například toto

```
1 a {
2     -webkit-border-radius: 5px;
3     border-radius: 5px;
4 }
```

Listing 5: Autoprefixer-02-vstup

Zkompiluje do:

```
1 a {
2     border-radius: 5px;
3 }
```

Listing 6: Autoprefixer-02-výstup

Autoprefixer používá “Browserlist” plugin, tudíž můžeme specifikovat, jaké prohlížeče chceme cílit v našem projektu. Například můžeme nastavit poslední dvě verze prohlížeče.

Ve výchozím nastavení Autoprefixer také odstraňuje zastaralé předpony. Toto chování můžeme zakázat.

```
remove: false
```

3.4.2 `postcss-easy-import`

Importy, které mohou být známé z preprocesorů, v čistém CSS neuděláme, proto užitečný plugin `postcss-easy-import`. Tento plugin nám umožní importovat jiné CSS do našeho CSS, tudíž můžeme naše CSS lépe třídit a organizovat. Plugin umí také automaticky rozpoznat cesty k do `/node_modules` nebo importovat všechny soubory v adresáři. [1] [13]

```
1 /* komponenta z node_modules: */
2 @import "normalize.css";
3 /* relativní cesta: */
4 @import "./theme.css";
5 /* import všech souborů v adresáři: */
6 @import "./components/*.css";
```

Listing 7: postcss-easy-import ukázka importu v css

Použití: vložíme do seznamu vyžádaných pluginů do souboru našeho taskruneru, v našem případně Gruntfile.js.

```
postcss([ require('postcss-easy-import') ])
```

3.4.3 Font Magican

Plugin, který generuje veškerá pravidla @font-face.

- Možnost použít veškeré Google Fonty
- Lokální kopie pro návštěvníky
- Hostování vlastních fontů. Stačí specifikovat, kde se nachází.
- Načítání fontů asynchronně
- Podpora Bootstrapové typografie

Vstup:

```
1 body {
2     font-family: "Montserrat";
3 }
```

Listing 8: Font Magican - vstup

Výstup:

```
1 @font-face{
2   font-family:Montserrat;
3   font-style:normal;
4   font-weight:400;
5   src:
6   url(//fonts.gstatic.com/s/montserrat/v12/
       zhcZ-_WihjSQC0oHJ9TCYPk_vArhqVIZ0nv9q090hN8.woff2)
       format("woff2"),
7   url(//fonts.gstatic.com/s/montserrat/v12/
       zhcZ-_WihjSQC0oHJ9TCYBsxEYwM7FgeyaSgU71cLGO.woff)
       format("woff"),
8   url(//fonts.gstatic.com/s/montserrat/v12/
       zhcZ-_WihjSQC0oHJ9TCYFQ1YEbsez9cZjKsNMjL0wM.eot?#)
       format("eot"),
9   url(//fonts.gstatic.com/s/montserrat/v12/
       zhcZ-_WihjSQC0oHJ9TCYC3USBnSvpkopQaUR-2r7iU.ttf)
       format("truetype"),
10  url(//fonts.gstatic.com/l/font?kit=
       zhcZ-_WihjSQC0oHJ9TCYKWUboTb-jS2tyCOQMtm97g&skey=7
       bc19f711c0de8f&v=v12#Montserrat) format("svg");
11 }
12
13 body {
14   font-family: "Montserrat";
15 }
```

Listing 9: Font Magican - výstup

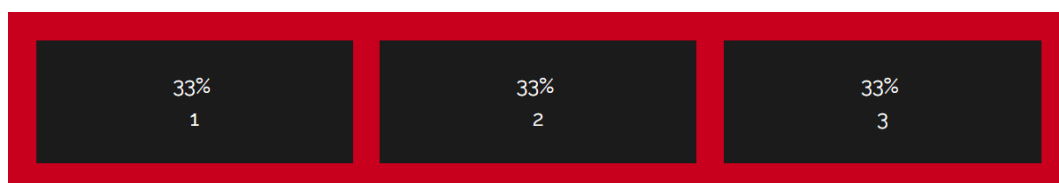
3.4.4 LostGrid

Je to nástroj na tvorbu mřížek nebo rozestavení elementů na stránce, který funguje s každým preprocesorem, a dokonce i s čistým CSS. LostGrid používá vlastní zjednodušenou syntaxi psaní mřížek a používá u toho funkci `calc()` [14]

Základní jednoduchá struktura

```
1 .row {
2     lost-center: 1200px;
3     lost-utility: clearfix;
4 }
5
6 .col-33 {
7     lost-column: 1/3;
8 }
9
10 @media (--mobile) {
11     .e-col-33 {
12         lost-column: 1/1;
13     }
14 }
```

Listing 10: LostGrid - základní struktura

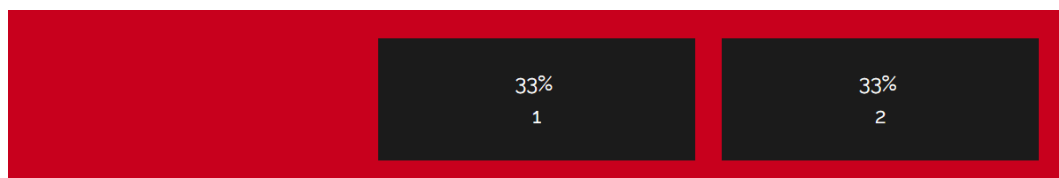


Obrázek 3: LostGrid - základní struktura gridu

Použití odsazení

```
1
2 .col-33 {
3     lost-column: 1/3;
4 }
5
6 .row__offset .col-33:first-child {
7     lost-offset: 1/3;
8 }
9
10 @media (--mobile) {
11     .row__offset .col-33:first-child {
12         lost-offset: clear-left;
13     }
14 }
```

Listing 11: LostGrid - použití odsazení

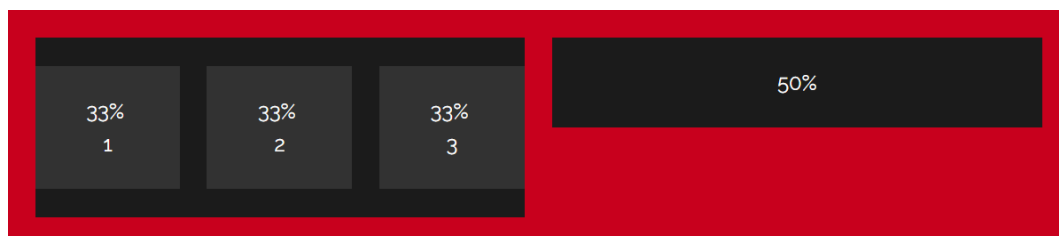


Obrázek 4: LostGrid - použití odsazení

Použití zanořování

```
1  
2 .col-33 {  
3     lost-column: 1/3;  
4 }  
5  
6 .col-50 {  
7     lost-column: 1/2;  
8 }
```

Listing 12: LostGrid - použití zanořování



Obrázek 5: LostGrid - použití odsazení

4 cssnext

Cssnext je plugin nástroje PostCSS, který nám pomáhá psát nejnovější CSS syntaxi. Transformuje nové CSS specifikace ve více kompatibilní CSS, takže nemusíme čekat na podporu prohlížečů. Jelikož psát čisté CSS může být frustrující, jelikož nemá specifikaci pro funkce, jako jsou například: proměnné, matematické funkce, manipulace s barvami a žádná přizpůsobení. Díky práci W3C se tyto funkce pro čisté CSS připravují, ale díky cssnext, můžeme tyto funkce používat již dnes. Cssnext je budoucnosti vzdorný, což znamená, že pokud prohlížeče budou implementovat nové CSS specifikace a cssnext odstraní ty, které již nebudou potřeba. [15]

4.1 Konfigurace cssnext

Abych mohli používat nástroj cssnext. Musíme prvně mít nainstalovaný Node.js a námi zvolený Task runner a nástroj PostCSS.

Nástroj cssnext nainstalujeme pomocí příkazu v příkazové řádce:

```
npm install postcss-cssnext --save-dev
```

Poté zařadíme cssnext mezi procesory v nastavení Postcss v našem konfiguračním souboru task runneru:

```
require('postcss-cssnext')({option: value})
```

4.2 Funkce

4.2.1 Vlastní vlastnosti a proměnné

Zaměřuje se poskytovat budoucnosti vzdorný způsob, omezující se na :root selektor přirozeně poskytující CSS vlastnosti. [15] Tato funkce nám dovolí použít vlastní vlastnosti v CSS

```
1 :root {
2   --mainColor: red;
3 }
4
5 a {
6   color: var(--mainColor);
7 }
```

Listing 13: cssnext deklarace css vlastnosti jako proměnné

4.2.2 Vlastní vlastnosti a @apply

Dovolí nám uschovat vlastnosti pojmenované vlastní proměnné a poté je odkázat v někdy jinde ve stylech. [15]

Vstup:

```
1 :root {
2   --pos-absolute-center: {
3     position: absolute;
4     top: 50%;
5     left: 50%;
6     transform: translation(-50%, -50%);
7   };
8 }
9
10 .muj-element {
11   @apply --pos-absolute-center;
12 }
```

Listing 14: cssnext deklarace elementu s vlastnostmi - vstup

Výstup:

```
1 .muj-element {  
2     position: absolute;  
3     top: 50%;  
4     left: 50%;  
5     transform: translate(-50%, -50%);  
6 }
```

Listing 15: cssnext vlastní element - výstup

4.2.3 Upravená funkce calc()

Dovolí nám použít funkci calc() i s proměnnými.

```
1 :root {  
2     --fontSize: 1rem;  
3 }  
4  
5 h1 {  
6     font-size: calc(var(--fontSize) * 2);  
7 }
```

Listing 16: cssnext - použití funkce calc() s proměnnou

4.2.4 Vlastní media queries

Umožní nám si vytvářet vlastní pojmenované media queries.

```
1 @custom-media --small-viewport (max-width: 30em);
2
3
4 @media (--small-viewport) {
5     /* styly pro malý viewport */
6 }
```

Listing 17: cssnext - vlastní pojmenované media queries

4.2.5 Rozsahy media queries

Povolí nám nahradit min a max za \leq a \geq pro lepší syntaxi a čitelnost.

```
1 @media (width >= 500px) and (width <= 1200px) {
2     /* naše styly*/
3 }
4
5 /* Nebo s kombinací vlastních media queries */
6 @custom-media --only-medium-screen (width >= 500px) and
7     (width <= 1200px);
8
9 @media (--only-medium-screen) {
10     /* naše styly */
11 }
```

Listing 18: cssnext - vlastní pojmenované media queries s cssnext rozsahy

4.2.6 Vlastní selektory

Funkce, která nám umožní vytvářet si vlastní selektory. K definování vlastního selektoru musíme použít “:-”

Vstup:

```
1 @custom-selector :--any-link :link, :visited;
2
3 a:--any-link {
4   color: blue;
5 }
```

Listing 19: cssnext - vlastní selektory - vstup

Výstup:

```
1 a:link,
2 a:visited {
3   color: blue;
4 }
```

Listing 20: cssnext - vlastní selektory - výstup

4.2.7 Zanořování

Funkce, které nám umožní zanořovat styly uvnitř každého stylu.

Vstup:

```
1 .main-nav {
2     margin-bottom: 2.5rem;
3     & ul {
4         display: flex;
5         justify-content: flex-end;
6         list-style-type: none;
7     }
8     & a {
9         display: block;
10    }
11 }
```

Listing 21: cssnext - zanořování - vstup

Výstup:

```
1 .main-nav {
2     margin-bottom: 2.5rem;
3 }
4 .main-nav ul {
5     display: flex;
6     -webkit-box-pack: end;
7     list-style-type: none;
8 }
9 .main-nav a {
10    display: block;
11 }
```

Listing 22: cssnext - zanořování - výstup

4.2.8 Modifikace barev

Funkce umožňující modifikovat barvy, které poté překompiluje do rgba formátu. Nabízí několik možností modifikace barev.

Vstup:

```
1 color( barva [modifikace-barvy( [+/-/%] ] ) )
```

Listing 23: cssnext - rozhraní modifikace barev

Možnosti Modifikace barev:

- tint - světlejší odstín barvy
- shade - tmavší odstín barvy
- a/alpha - průhlednost barvy
- s/saturation - saturace barvy

Průhlednost barvy

Vstup:

```
1 body {  
2   background: color(red a(90%))  
3 }
```

Listing 24: cssnext - modifikace barev - vstup

Výstup:

```
1 body {  
2   background: rgba(255, 0, 0, 0.9)  
3 }
```

Listing 25: cssnext - modifikace barev - výstup

5 Porovnání

5.1 Porovnání oproti CSS

Zásadní výhoda oproti ostatním nástrojům na transformaci CSS je ta že, nepotřebujeme žádné programy. Instalovat jiné nástroje a poté je složitě konfigurovat. Jednoduše vytvoříme soubor s koncovkou “.css a můžeme psát. Nemusíme pracovat v příkazovém řádku a sestavovat složité automatizační úlohy. Bohužel zde výhody končí a nastává několik problémů, které se začnou objevovat při psaní čistého CSS. Začnou se objevovat magické konstanty a kód začne být nepřehledný. Psaní kódu začne být zdlouhavé a frustrující.

5.1.1 Výhody PostCSS a cssnext

- Proměnné
- Zanořování
- Matematické výrazy
- Zlepšuje přehlednost kódu
- Zvyšuje efektivnost psaní kódu

5.1.2 Nevýhody PostCSS a cssnext

- Počáteční časová investice do vytvoření vlastního workflow
- Průzkum dostupných pluginů
- Nutnost kompilace pomocí automatizačních nástrojů
- Podpora pluginů nemusí být zaručena

Z počátku se může zdát časová investice vysoká, ale tato investice se vrací, jelikož začneme psát kód mnohem rychleji a stane se mnohem přehlednější,

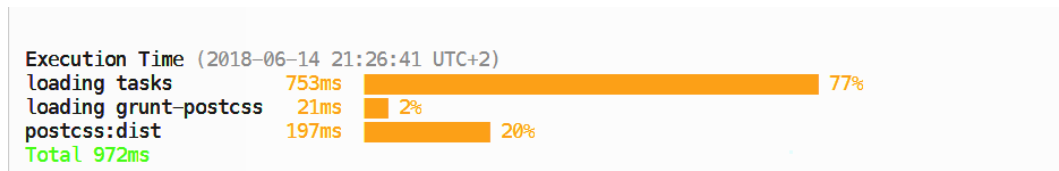
například pomocí proměnných. Naší workflow můžeme přenášet na další projekty a můžeme ji dále rozšiřovat a zdokonalovat. Zlepší organizaci kódu. Díky vlastnosti “@import” můžeme zapomenout na organizaci kódu orientovanou na stránky nebo na celostránkové breakpoints v media queries. PostCSS nám kód umožní organizovat přes moduly malé kousky kódu vztažené ke konkrétní komponentě uživatelského rozhraní, jako je třeba navigace, článek nebo karusel. [1]

5.2 Porovnání oproti preprocesorům

PostCSS může rychleji kompilovat, ale také nemusí. Záleží na počtu pluginů a jaké pluginy budeme používat. PostCSS lze použít s kombinací s preprocesory, aby doplnil funkce, které preprocesory neumí.

5.2.1 Časové porovnání kompilace

Samotná kompilace PostCSS je velice rychlá, pouhých 21ms. Bohužel načtení tohoto nástroje je o něco delší, necelá vteřina.



Obrázek 6: Rychlost kompilace PostCSS

```

C:\Programy\EasyPHP-Devserver-17\eds-www\PostCSS-speedTest\LESS>grunt
Running "less:dist" (less) task
>> 1 stylesheet created.

Done.

Execution Time (2018-06-18 18:15:13 UTC+2)
loading grunt-contrib-less 134ms 9%
less:dist 1.4s 91%
Total 1.6s

```

Obrázek 7: Rychlost kompilace LESS preprocesoru

```

C:\Programy\EasyPHP-Devserver-17\eds-www\PostCSS-speedTest\SCSS>grunt
Running "sass:dist" (sass) task

Done.

Execution Time (2018-06-23 15:06:28 UTC+2)
loading grunt-contrib-sass 19ms 1%
sass:dist 1.5s 98%
Total 1.6s

```

Obrázek 8: Rychlost kompilace SASS preprocesoru

Oba preprocesory LESS a SASS se načítají velmi rychle, ale jejich kompilace je pomalejší. LESS se načte za 134ms a jeho kompilace trvá 1.4s. SASS je na tom podobně, který se načte za 19ms a zkompiluje se za 1.5s.

Samozřejmě rychlost kompilace ovlivňuje počet souborů a velikost kódu. Snažil jsem se o co nejpodobnější nároky na nástroje a množství kódu tak, aby daly výsledky porovnat.

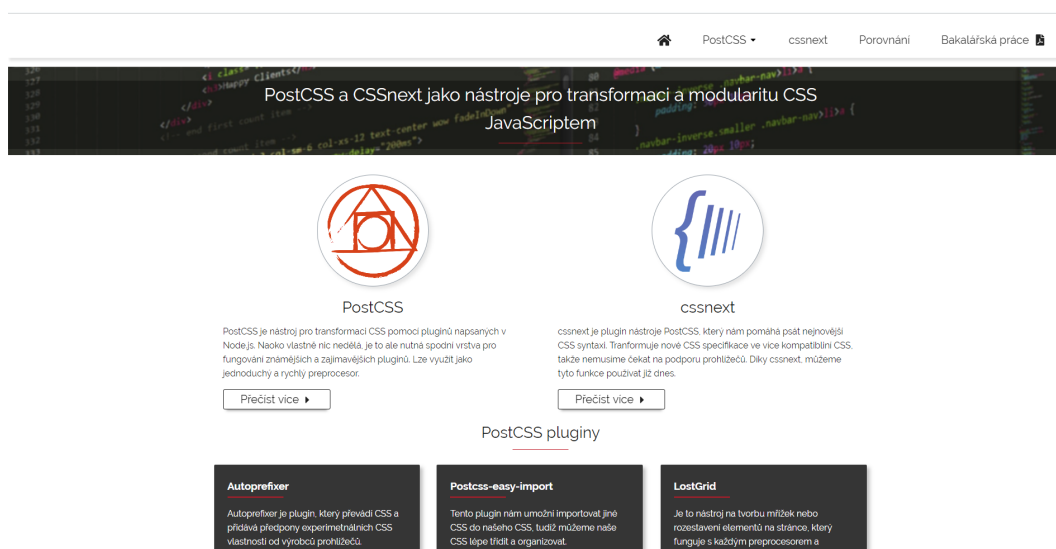
Nazev nástroje	Rychlost kompilace	Rychlost načtení	Celková rychlost
PostCSS a cssnext	197ms	753ms	972ms
SASS	1.5s	19ms	1.6s
LESS	1.4s	134ms	1.6s

Tabulka 1: Časové porovnání kompilace

6 Praktická část

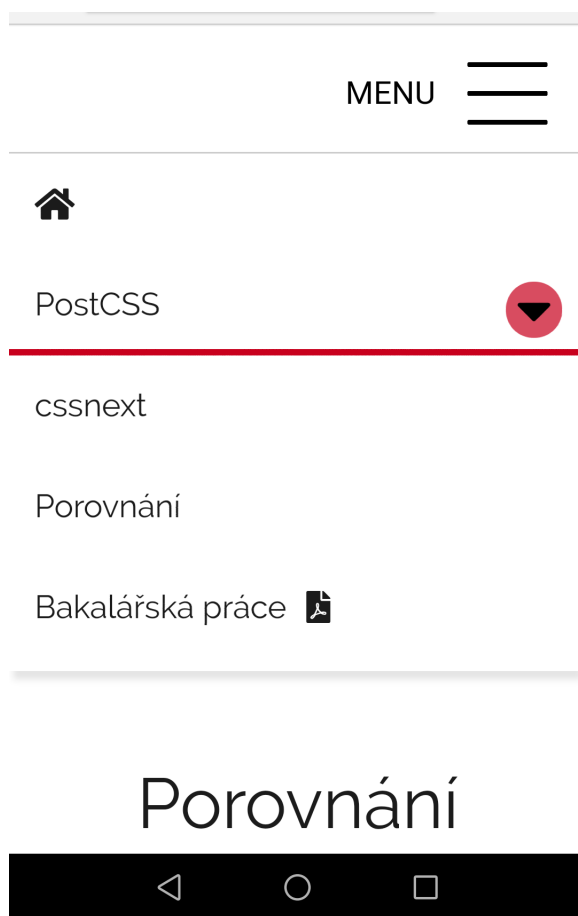
Praktická část je zpracovaná ve formě webové aplikace, která je příručkou pro nástroje PostCSS a cssnext. Stránka je na adrese `<http://jakubjetleb.cz/baka/>`. Webová aplikace je vytvořena pomocí těchto nástrojů, které jsou na webu detailně popsány a rovněž popis porovnání těchto nástrojů oproti čistému CSS a preprocesorům. Dále je zde popsáno, jak tyto nástroje nainstalovat a konfigurovat. Několik užitečných pluginů pro PostCSS jsou na webu představeny, jaké mají funkce a jak se používají. V sekci s cssnext jsou popsány funkce, které tento nástroj nabízí.

Pro vývoj této webové aplikace jsem použil vývojové prostředí Brackets, se kterým se mi pracuje dobře a mám s ním mnoho zkušeností. Brackets je vytvořen společností Adobe Systems a je šířen pod MIT licenci. Je to open-source textový editor primárně zaměřený na HTML, CSS a JavaScript. Pro programování ostatních jazyků jako PHP není docela vhodný. Brackets má spoustu užitečných doplňků, které ulehčují práci, například nástroj Emmet, který usnadňuje psaní kódu pomocí zjednodušené syntaxe, nebo pro organizaci a formátování kódu doplněk Beautify, který formátuje HTML CSS a JavaScriptové soubory do úhlednější a čitelnější podoby, takže se nemusíme obávat o nerovnoměrně zformátovaný kód.





Obrázek 9: Úvodní stránka webové stránky

Web se skládá z čtyř hlavních stránek a pěti podstránek pro PostCSS pluginy a odkaz na bakalářskou práci ke stažení. V hlavičce webu je jednoduché navigační menu. Stránka je plně responzivní a je optimalizovaná pro mobily a tablety. Hlavní menu při mobilním zobrazení se skládá pod sebe s tlačítkem na vyjždění menu neboli "hamburger".



Obrázek 10: Mobilní zobrazení webové stránky

Velikost CSS se mi podařilo dostat v minifikované verzi na 24kB oproti tomu, kdybych použil například masivní framework Bootstrap 4.1, jehož CSS v minifikované verzi má 140kB. Nebo například podobný framework Foundation, který je na tom s velikostí minifikovaného CSS o něco líp, a tedy 117kB ve verzi 6.42.

Název	Typ	Velikost
 style-min.css	Soubor CSS	24 kB
 style.css	Soubor CSS	55 kB

Obrázek 11: Velikost CSS webové stránky

7 Závěr

Pokud používáme pouze čisté CSS, nástroje PostCSS s kombinací cssnext, jsou skvělé doplňky pro zaostávající CSS. Pomůže nám lépe organizovat kód a zrychlí psaní kódu, například pomocí proměnných nebo vkládaných importů.

Použití těchto nástrojů zrychluje vývoj webových aplikací, ačkoliv počáteční časová investice do znalosti syntaxe těchto nástrojů se může zdát vysoká, tak se nám následně produktivita vývoje zvýší, tudíž určitě se vyplatí tyto nástroje používat.

Porovnání s preprocesory je o něco obtížnější, jelikož PostCSS a cssnext lze použít jako doplněk pro naši workflow s preprocesory, doplnit některé funkce, které preprocesory nenabízejí, nebo můžeme PostCSS sestavit tak, že budou pracovat jako preprocesor. PostCSS se určitě hodí na menší projekty, kdy nemáme složitě navržené stránky. Pokud bychom měli potřebu používat ve stylech cykly, funkce, mixiny a složité grid systémy, je lepší použít preprocesor, kde již využijeme veškerých funkcí, které preprocesory nabízí. Rychlostně jsem porovnal PostCSS s doplňkem cssnext, který celkově kompiluje rychleji. Ovšem dá se předpokládat, že s více doplňky PostCSS se kompilace může postupně zpomalovat.

Na své webové stránce jsem podrobně popsal nástroje PostCSS, cssnext a další zajímavé doplňky pro PostCSS. Dále jsem porovnal tyto nástroje s čistým CSS a preprocesory, jako jsou SASS a LESS, kde je rovněž i časové porovnání rychlosti kompilace těchto nástrojů.

Seznam použité literatury a zdrojů

- [1] MICHÁLEK, Martin. Vzhůru dolů — webový frontend ze všech stran [online]. [cit. 2017-06-22]. Dostupné z: <<http://www.vzhurudolu.cz/>>
- [2] KEARNEY, Meggin a Matt GAUNT. Set Up Your Build Tools. In: Google Developers [online]. 2018 [cit. 2018-04-08]. Dostupné z: <<https://developers.google.com/web/tools/setup/setup-buildtools>>
- [3] GRUNT. Grunt: The JavaScript Task Runner [online]. [cit. 2017-04-10]. Dostupné z:<<https://gruntjs.com/>>
- [4] ALMAN, Ben. INTRODUCING GRUNT. In: Bocoup [online]. 2012 [cit. 2018-04-08]. Dostupné z: <<https://bocoup.com/blog/introducing-grunt>>
- [5] GERCHEV, Ivaylo. Multilingual Preprocessing with Prepros. In: Sitepoint [online]. 2013 [cit. 2018-04-08]. Dostupné z: <<https://www.sitepoint.com/multilingual-preprocessing-with-prepros/>>
- [6] , Prepros. Prepros [online]. 2018 [cit. 2018-06-24]. Dostupné z: <<https://prepros.io/>>
- [7] CLARK, David. It's Time for Everyone to Learn About PostCSS: What It Really Is; What It Really Does. In: Davidtheclark [online]. 2015 [cit. 2018-04-08]. Dostupné z: <<https://davidtheclark.com/its-time-for-everyone-to-learn-about-postcss/>>
- [8] Evil Martians a Sitnik the Developer. PostCSS. Github [online]. [cit. 2018-04-08]. Dostupné z: <<https://github.com/postcss/postcss>>
- [9] Npm [online]. Schlueter, 2011 [cit. 2018-04-08]. Dostupné z: <<https://www.npmjs.com/>>
- [10] Autoprefixer. In: GitHub [online]. 2008, 2016 [cit. 2018-04-08]. Dostupné z: <<https://github.com/postcss/autoprefixer>>

- [11] :fullscreen. In: MDN [online]. 2005, 2018 [cit. 2018-04-08]. Dostupné z: <<https://developer.mozilla.org/en-US/docs/Web/CSS/:fullscreen>>
- [12] Calc(). In: MDN [online]. 2015, 2018 [cit. 2018-04-08]. Dostupné z: <<https://developer.mozilla.org/en-US/docs/Web/CSS/calc>>
- [13] simonsmith. Postcss-easy-import. In: GitHub [online]. 2008, 2016 [cit. 2018-04-08]. Dostupné z: <<https://github.com/TrySound/postcss-easy-import>>
- [14] Documentation - LostGrid. Documentation - LostGrid [online]. [cit. 2018-06-26]. Dostupné z: <<http://lostgrid.org/docs.html>>
- [15] THIROUIN , Maxime. cssnext - Use tomorrow's CSS syntax, today. [online]. [cit. 2017-04-10]. Dostupné z: <<http://cssnext.io/>>
- [55] Joyent, Inc. Node.js [online]. [cit. 2017-04-10]. Dostupné z: <<https://nodejs.org/en/>>

Seznam obrázků

1	Ukázka grafického rozhraní Prepros	15
2	Základní struktura projektu	20
3	LostGrid - základní struktura gridu	26
4	LostGrid - použití odsazení	27
5	LostGrid - použití odsazení	28
6	Rychlost kompilace PostCSS	37
7	Rychlost kompilace LESS preprocesoru	38
8	Rychlost kompilace SASS preprocesoru	38
9	Uvodní stránka webové stránky	40
10	Mobilní zobrazení webové stránky	41
11	Velikost CSS webové stránky	42

Seznam ukázek kódu

1	Inicializace souboru package.json	20
2	Závislosti v souboru package.json	21
3	Autoprefixer-01-vstup	21
4	Autoprefixer-01-výstup	22
5	Autoprefixer-02-vstup	23
6	Autoprefixer-02-výstup	23
7	postcss-easy-import ukázka importu v css	24
8	Font Magican - vstup	24
9	Font Magican - výstup	25
10	LostGrid - základní struktura	26
11	LostGrid - použití odsazení	27
12	LostGrid - použití zanořování	28
13	cssnext deklarace css vlastnosti jako proměnné	30
14	cssnext deklarace elementu s vlastnostmi - vstup	30
15	cssnext vlastní element - výstup	31
16	cssnext - použití funkce calc() s proměnnou	31
17	cssnext - vlastní pojmenované media queries	32
18	cssnext - vlastní pojmenované media queries s cssnext rozsahy .	32
19	cssnext - vlastní selektory - vstup	33
20	cssnext - vlastní selektory - výstup	33
21	cssnext - zanořování - vstup	34
22	cssnext - zanořování - výstup	34
23	cssnext - rozhraní modifikace barev	35
24	cssnext - modifikace barev - vstup	35
25	cssnext - modifikace barev - výstup	35

8 Přílohy

1. CD se zdrojovými kódy celé aplikace a plné znění BP v PDF
2. Webová stránka: <http://jakubjetleb.cz/baka/>