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

Tvorba vizuálních komponent webu s využitím Object Oriented CSS a metodiky BEM

Creating visual components of web using Object Oriented CSS and BEM methodology

Bakalářská práce

Vypracoval : Jan Smrž

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

České Budějovice 2020

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH
Pedagogická fakulta
Akademický rok: **2017/2018**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jan SMRŽ**
Osobní číslo:	**P16950**
Studijní program:	**B7507 Specializace v pedagogice**
Studijní obor:	**Informační technologie a e-learning**
Název tématu:	**Tvorba vizuálních komponent webu s využitím Object Oriented CSS a metodiky BEM**
Zadávající katedra:	**Katedra informatiky**

Zásady pro vypracování:

Cílem bakalářské práce je představit možnosti využití správných postupů při psaní objektově orientovaného CCS (OOCSS) kódu v HTML5 za pomocí metodiky BEM při vývoji front-endu webu. OOCSS je způsob, jak snadno psát srozumitelný kód, který je možné použít i opakovaně, je lehce spravovatelný a nezabírá v souboru příliš mnoho datového prostoru. Metodika BEM určuje postupy, které lze využít ke správnému pojmenování jednotlivých bloků, elementů nebo modifikátorů. Tyto prvky společně zjednodušují práci při psaní CSS kódu nejen pro autora, který ho píše, ale i pro další spolupracující členy týmu. V rámci práce bude porovnána tvorba objektově orientovaného CSS kódu s využitím metodiky BEM s klasickým postupem bez použití uvedených technologií. Výstupem práce bude webová prezentace, vytvořená s využitím OOCSS s BEM metodologií, bude otestována časová náročnost, datová velikost a obtížnost práce pro vývojáře.

Rozsah grafických prací: **CD ROM**

Rozsah pracovní zprávy: **40**

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

1. **BEM [online]. Moskva: Yandex, 2017 [cit. 2018-04-05]. Dostupné z: https://en.bem.info/**
2. **Get BEM [online]. BEM?- Block Element Modifier, 2018 [cit. 2018-04-05]. Dostupné z: http://getbem.com/**
3. **OOCSS: objektové psaní CSS. Vzhůru dolů [online]. Praha: Martin Michálek, 2017 [cit. 2018-04-05]. Dostupné z: https://www.vzhurudolu.cz/prirucka/oocss**
4. **BEM: Pojmenovávací konvence pro třídy v CSS. Vzhůru dolů [online]. Praha: Martin Michálek, 2017 [cit. 2018-04-05]. Dostupné z: https://www.vzhurudolu.cz/prirucka/bem**
5. **An Introduction To Object Oriented CSS (OOCSS). Smashing magazine [online]. Freiburg: Smashing Media, 2011 [cit. 2018-04-05]. Dostupné z: https://www.smashingmagazine.com/2011/12/an-introduction-to-object-oriented-css-oocss/**

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

Datum zadání bakalářské práce: **12. dubna 2018**

Termín odevzdání bakalářské práce: **30. dubna 2019**

Mgr. Michal Vančura, Ph.D.
děkan

L.S.

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

V Českých Budějovicích dne 12. dubna 2018

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 14. května 2020

Jan Smrž

Abstrakt/Anotace

Cílem bakalářské práce je využít správných postupů při psaní objektově orientovaného CCS (OOCSS) kódu v HTML5 za pomocí metodiky BEM při vývoje frontendu. OOCSS je způsob, jak snadno psát srozumitelný kód, který se může použít hned několikrát, lehce spravovatelný a nezabírá v souboru příliš mnoho místa. Metodika BEM nám určuje postupy, které využijeme na správné pojmenovaní jednotlivých bloků, elementů nebo modifikátorů. Dohromady mají celkově zjednodušit práci při psaní kódu CSS nejen pro člověka, který ho píše, ale i pro další členy týmu. V rámci práce budu porovnávat psaní kódu CSS jako objektově orientovaného, výhody používaní metodiky BEM a nepoužití ani jednoho z postupů. Výstupem práce bude webová prezentace obsahující použití OOCSS s BEM metodologií, časovou náročnost, datovou velikost a obtížnost pro vývojáře.

Keywords

CSS, OOCCS, BEM, frontend

Abstract

The aim of this thesis is to use the right approach when writing object-oriented CSS (OOCSS) code in HTML5 and using the methodology of BEM while developing frontend. OOSCC is a way to write easily understandable code which can be used multiple time is simple to manage and does not take much space. BEM methodology gives us an approach that is used to correctly name different blocks, elements or modifiers. Both of them together can be used to simplify writing CSS not only for the person writing it but also for everyone else on the team. In this thesis I will be comparing writing object-oriented CSS, benefits of using BEM methodology and not using either of these approaches. The outcome of this work will be a web presentation consisting of OOCSS with BEM methodology, time consumption, data size and difficulty of development for developers.

Keywords

CSS, OOCCS, BEM, frontend

Poděkování

Děkují vedoucímu práce za rady, které mi dával se zhotovením práce. Děkuji své rodinně za podporu, kterou mi dodávala při celém výkonu studia. V neposlední řadě, také své přítelkyni, která mi byla psychickou podporou a svým přátelům, kteří mi nezištně pomáhali.

Obsah

1 Úvod

1.1 Východiska práce

Webový design se s objektově orientovaným psaním kódu ještě nesblížil, jak by měl. Tento způsob lépe zviditelní jednotlivé komponenty a bude tím zjednodušené čtení kódu. Je tedy výhodné spojit jej i s metodikou BEM, která jasně nastavuje, jak elementy vizuální části stránky transformují svoje názvy podle pozic a využití na stránce. CSS psané za použití obou těchto technologií by se navíc dalo použít u více projektů. Pro OOCSS a BEM u nás neexistuje ucelená ukázka využití s CSS a popsání výhod nebo nevýhod.

1.2 Cíle práce

Cílem bakalářské práce je ukázat vývoj front-end webové stránky při použití OOCSS a metodiky BEM. OOCSS je rozšíření samotného CSS pravidly pro psaní kódu v jiných jazycích (Java, C++, C). Výhody, které by tento způsob vytváření kódu měl přinést, jsou jednodušší čtení kódu, možnost opakovaně použit takto napsaný kód a navíc i menší velikost souborů CSS. Metodologie BEM vylepšuje OOCSS o jasné a neměnné názvosloví jednotlivých elementů kódu. Teoretická část práce bude obsahovat samotné představení informací a pravidel o OOCSS a BEM pro jejich správné použití při vývoji front-endu. Praktickou část bude tvořit webová prezentace, která obě tyto technologie využije a porovná CSS bez nich.

1.3 Metoda práce

Úvod práce se zaměří na CSS samotné a způsoby jeho psaní pro front-end webové stránky. Poté přejdeme přímo na OOCSS a BEM a porovnám použití s nimi a bez nich. Všechno následně spojíme do webové prezentace, která prakticky představí rozdíly v CSS bez použití Objektově orientovaných pravidel s metodikou BEM a s nimi.

2 Technologie webu

Dnes se při tvorbě webových stránek dá použít nespočet technologií a postupů, aby stránka vypadala podle našich představ. Některé jsou zde s námi již od počátku věků, kdy na svět přišla první webová stránka a ostatní se začaly objevovat kvůli vylepšení nebo usnadnění práce vývojáře. Hlavními proudy vývoje jazyků pro web ať, značkovacích, nebo programovacích se zabývá organizace W3C. Schvaluje a sdružuje většinu dokumentace, pro projekty spjaté s vývojem webu, dostupné ze stránek organizace. Můžete zde také nalézt krátké kurzy pro seznámení s jednotlivými jazyky.[1]

Mezi už zmíněné technologie se dají samozřejmě zařadit samotné prohlížeče webových stránek, které se snaží podporovat pravidla jazyků udávaná W3C. Vývoj jde totiž vždy dopředu a ne všechno, co prohlížeče podporují, je schváleno nebo plně funkční. Existují proto předpony. Přikazují prohlížečům používat pro ně ještě nezavedené příkazy a obejít problém, kdy se na různých prohlížečích nezobrazí stránka ve stejném vzhledu. Tyto problémy se objevují u starších prohlížečů, kde jíž skončila jejich podpora, ale uživatelé nevědí nebo nechtějí přejít na aktuální verzi. Pokud některou z předpon použijeme, můžeme zachovat uniformitu vzhledu stránky.[1]

Patří sem frameworky fungující na principu knihoven, kdy kodér již nemusí vypisovat stále stejné příkazy na rozložení stránky a jejich parametry, ale stačí mu, aby si pamatoval pár, které mu pomůžou stránku složit podle libosti. Může jít o různé druhy frameworků například k vývoji webových aplikací nebo internetového obchodu s velkým počtem transakcí. [1]

JavaScript, PHP a Python neodmyslitelně také patří do této kategorie. Najdeme je dnes na každé webové stránce, protože se starají například o interaktivitu a správnost zadaných údajů v kontaktním formuláři. Jsou součástí každé větší stránky, Facebookem začínaje a Wikipedia konče. [1]

3 CSS

CSS, neboli kaskádové styly, se používají ke stylizaci webových stránek. Může se jednat o grafickou podobu stránky, barva pozadí, velikost písma, ale také způsob, jakým je stránka rozložena. Nebyl to jediný jazyk, který se zrodil pro tuto funkci, ale jako jediný bral v potaz to, že není jen na autorovi nebo čtenáři, jak bude stránka vypadat. Obě strany se musí spojit a přidat navíc i velmi důležité části, zařízení a prohlížeč, které budou zobrazovat obsah. CSS stejně jako většinu webových technologií má na starost konzorcium W3C. [1]

Před vynalezením tohoto jednotného jazyka měl každý prohlížeč vlastní pravidla pro stylizování. Některé prohlížeče měly svoje osobité verze dnešních kaskádových stylů. Například prohlížeč NeXT měl možnost editace webové stránky podle uživatele, ale tvůrce nepublikoval syntax. Myslel si, že je na každém prohlížeči, jak má zobrazovat obsah pro čtenáře. Postupně se u některých prohlížečů ukazovaly funkce, které dovolovaly měnit některé barvy a fonty, ale nebylo to dost. Tvůrci webových stránek si stále stěžovali, že nemají žádnou kontrolu nad tím, jak jejich stránky na webu vypadají. Jazyk HTML žádné z těchto funkci nepodporoval a jeho tvůrce to tak ani nechtěl. CSS byla tedy technologie, která měla všechny tyto problémy vyřešit a dát prohlížečům jasná pravidla pro stylizování webových stránek.[1]

3.1 Historie CSS

První ukázka CSS byla v listopadu roku 1994 při příležitosti Web konference Håkonem Wium Lie. Byl to teprve koncept balancování autora a čtenáře. Kritici vyzdvihovali, že není dostatečně flexibilní a bude potřeba plnohodnotného jazyka na stylizování dokumentů. CSS šlo přesně proti tomu. Bylo jednoduché a mělo jasný formát. Další rok se při prezentaci konečně objevila implementace v prohlížečích Argo a Arena, které byly uzpůsobeny pro využití kaskádových stylů. Znovu se objevili kritici. Tentokrát se debata proměnila v politickou, kdy jedna strana stála za autory dokumentů a druhá za čtenáři. Autoři například

namítali, že existují některé zákony ovlivňující velikost písma na varovných značeních. Čtenáři stáli za tím, že uživatelé jsou ti, kteří by měli rozhodovat, pokud vzniknou nějaké problémy, protože je na nich, aby se vyznali v prezentacích autorů.[2]

Jeden z nejdůležitějších úkolů bylo donutit autory webových prohlížečů podporovat právě technologii CSS. Existovalo by totiž riziko, že by každý prohlížeč podporoval vlastní technologii a webové stránky by tedy vypadaly vždy nějak jinak a ne tak, jak to tvůrci chtěli. První, kdo se zavázal podporovat právě CSS, byl Microsoft se svým Internet Explorer verze 3, ale právě s konkurenční Netscape byl vůči kaskádovým stylům dost skeptický a jeho první podpora byla jen slabý pokus ukázat, že stejně jako Microsoft, podporují standarty. Jejich implementace brala části CSS pravidel a překládala je do JavaScript. Tak nechala tvůrce webu psát tento JavaScript (JSSS) přímo a obcházet CSS. Naštěstí se tento způsob kaskádových stylů neuchytil a zůstalo jenom CSS.[2]

Další gigant ve světě prohlížečů té doby byla Opera. Jako jediná měla čas otestovat implementaci CSS1 a podporovala tedy největší část této technologie. Testování bylo v té době dost unikátní a Håkon byl tak nadšen jejich přístupem, že se k vývojářům přidal v pozici CTO (Technický ředitel). Odpověď Netscape byla volně publikovat zdrojový kód jejich prohlížeče. Doufali, že tím otevřou dveře novým návrhům a vylepšením. Projekt „Mozila“ vzešel právě z jejich snahy. Hlavním úkolem bylo právě zaručit funkčnost CSS. Dobrovolníci strávili spoustu času, aby dosáhli toho, že se stránky budou zobrazovat právě podle specializací vývojářů webových stránek. [2]

Apple, který je považovaný za průkopníka nových technologií, se vývoji prohlížeče nebo webu příliš nevěnovali a nechali na Microsoftu, aby pro jejich zařízení vytvořil prohlížeč. Kupodivu byl právě Internet Explorer na Mac zařízení v CSS lepší než na platformě Windows. Safari vzniklo až poté, co Microsoft skončil s podporou Mac v roce 2003.[2]

CSS také přivedlo možnost webovým vývojářům přidávat vlastní fonty do webových stránek. Stačilo pouze přidat daný font jako součást kaskádo-

vého stylu. Tuto vlastnost podporoval prvních 10 let pouze Microsoft, Netscape používal alternativní způsob, který nebyl tak dobrý. Důvod, proč trvalo tak dlouho než tuto technologii začali používat i jiné prohlížeče je, že nebylo dostatek fontů bez autorských práv. Změna nastala v roce 2008. Objevilo se spoustu fontů jejichž tvůrci, je chtěli zadarmo rozšiřovat. Spustil se také tlak, aby Microsoft uvolnil svojí technologii EOT. Toto se nakonec povedlo a EOT se stala součást konzorcia W3C. Problémy však nekončily, velká většina vývojářů prohlížečů se rozhodla neimplementovat EOT a požadovala, aby W3C vytvořila nový formát fontů. Důvod byl, že preferovali algoritmus gzip pro kompresy, jak ho používalo HTTP. Nechtěli přidávat nový kód pro algoritmus, který využívá EOT. [2]

Webové fonty se staly velice oblíbenými a technologii EOT nebo od W3C využívají fonty málokdy. Nyní jsou online služby, které sdružují různé druhy fontů zadarmo a požadují pouze statistiku jejich využití.[2]

Vývoj CSS se ještě stále nezastavil. Existuje přes 60 modulů vysvětlující jak funguje, některé jsou stále ve vývoji, ostatní jsou již součástí standartu. Za pomocí CSS se například sázejí knihy, chybí sice pár funkcí pro komplexní rozvržení, ale je to jedna z cest, kterou se jazyk rozšiřuje. Využívají ho i elektronické publikace ve formátu EPUB. Poslední prohlížeči podporovanou verzí je CSS3 a to ne zcela. Připravovaná je CSS4, ale kdy přijde na řadu a jestli vůbec, je ve hvězdách.[2] [3]

3.2 Způsoby zápisu

Existuje hned několik způsobů, jak můžete CSS zapsat, aby mohl stylovat webovou stránku. Nejčastější způsob je psaní CSS v externím souboru a jejich následné aplikování v HTML po připojení souboru. Oddělíme tak od sebe oba kódy a dosáhneme větší přehlednosti při dělání změn, nebo přidávání dalších stylů. Zde můžeme vytvářet třídy a id, které sdružují jednotlivé elementy našeho stylu, nebo aplikovat styl přímo na jednotlivé tagy. Zároveň lze stylo-

vat i v návaznosti na strukturu, kdy ovlivňujeme tagy nadřazenou třídou, id nebo tagem. Soubory lze také úplně vynechat a psát CSS přímo do HTML souboru, a to buď ve speciálním style tagu nebo přímo do jednotlivých tagů tzv. in-line. Tato poslední možnost nám pevně přidá vlastnosti, které potřebujeme, ale těžce se pak mění, pokud nechme zasahovat do HTML kódu. Ukázky některých způsobů můžete vidět níže.

```
<html>
<head>
    <link rel="stylesheet" href="soubor.css">
</head>
<body>
    <button class="button">Button</button>
    <button id="btn">Button</button>
</body>
</html>
/*V CSS souboru*/
.button{
    ...
}
#btn{
    ...
}
```

Listing 1: CSS připojený pomocí externího souboru

```
1  <html>
2  <head>
3  <style>
4      .button {
5      ...
6      }
7  </style>
8  </head>
9  <body>
10     <button class="button">Button</button>
11 </body>
12 </html>
```

Listing 2: CSS ve style tagu v HTML za pomocí třídy

```
1  <button style="...;...;">Button</button>
```

Listing 3: CSS v in-line stylu

```
<div class="media">
    <p>...</p>
    <a href="#">Odkaz</a>
    <button>Button</button>
</div>
/*V CSS souboru*/
.media{
    p{
    ...
    }
    a{
    ...
    }
    button{
    ...
    }
}
```

Listing 4: CSS vázané na strukturu HTML

3.3 Preprocessoring

Hlavní myšlenka je zde ulehčení práce. Můžeme zde totiž využívat proměnné, kde si za značkou dolaru napíšeme cokoliv, co budeme používat na více než jednom místě v kódu. Zkracuje momenty neustálého psaní stejných vlastností. Přidává také hnízdění, s kterým můžeme jednoduše rozlišovat hierarchii podřazenosti a zlepšit čitelnost vazeb. Rizikem se stává přerůstání vazeb na opravdu velké nečitelné bloky, ale záleží samozřejmě na vývojáři. Mixins jsou sady definic využitelné pro celou stránku. Může jít např. pozadí, gradient a předponou značící ještě nepodporovaný příkaz. Extend nebo dědění je také důležitá součást. Omezuje psaní dlouhých tříd v HTML kódu, protože přidáte danou

vlastnost už třídě přímo v CSS souboru. Příkaz `<@extend >` totiž rozšíří vaší zvolenou třídu o všechen svůj obsah. Matematické operátory, jako +, -, /, * jsou také podporovány, třeba na počítaní šířky objektu a bez nutnosti přesné procentuální hodnoty. Další výhodou je možnost zpřehlednění naších souborů za pomocí funkce `@import`. Dovolí nám přidávat jednotlivé soubory do jednoho, který pak můžeme využít jako ten hlavní. Zamezíme tedy hledání určitého stylu v několika tisících řádek. Jednoduše najdeme soubor, který obsahuje například pouze styly pro menu a zde pak provádíme změny. Musíme ještě zmínit způsob, jakým nám pomáhá zmenšovat naše CSS soubory. Protože rychlost načítání stránek je dnes také velmi důležitá položka. Vytvoří totiž nejen normální CSS soubor, ale i jeho zmenšeninu. Výsledkem je celý kód napsaný bez zbytečných mezer v jednom řádku. Tímto výrazně zmenšíme velikost našeho souboru a zrychlíme tak načítaní stránky. [4][5]

V mojí práci budu používat preprocesor SASS v editoru Visual Studio. Populární jsou ještě LESS, Stylus a PostCSS. Pravidla jsou v nich většinou podobná a liší se od sebe syntaxí a způsobem kompilace.[4][5]

4 Metodologie CSS

Obecně si každý ve svém životě najde určitou metodu, jak svojí práci vykonávat co nejefektivněji nebo alespoň svým stylem. Protože ve většině případech nepracujeme sami, je vždy lepší se dohodnout, jaký daný způsob se vybere a bude se držet pro lepší domluvu v týmu. Právě z tohoto důvodu jsou tu metody s pevnými pravidly a jasným postupem. Přináší přístupné, flexibilní a lépe zvládnutelné programování, které se dá jednoduše používat znovu a znovu. Některé se dají používat spolu najednou. [6][7]

Patří mezi ně:

- OOCSS
- BEM a BEMIT
- ACSS
- SMACSS
- ITCSS

Práce se bude hlavně týkat prvních dvou zmíněných, ale popíšu stručně i zbývající. [6][7]

4.1 OOCSS - Objektově orientované CSS

V roce 2008 vymyslela tuto metodologii Nicole Sullivan. Popisuje vytváření abstraktních objektů, které by měli sloužit, jako základ kódu. Žádný z nich tedy není stylován ihned, ale jsou na ně aplikovány vlastnosti dalšími třídami. Vytvoříme si tedy například obecný styl pro nadpisy a posléze teprve vytvoříme třídu, která přidává velikost písma podle použití.[6][7][8][9]

Obrázek 1: Opakující se vizuální vlastnosti

Na výše viditelném obrázku je příklad opakujícího se objektu. Je vidět, že má společné vlastnosti, které jsou v tomto případě ohraničení a jeho barva. Liší se pouze ve způsobu jakým je aplikováno. Bez metodologie bychom nejspíše vytvořili pro každý jednu třídu a opakovali zbytečně části kódu. Metodika nám radí vytvořit si jednotlivé komponenty, které dokážou objekt vytvořit, pokud je dáme všechny dohromady. Ušetříme si tak čas při aplikování podobných stylů na další objekty, které se můžou na stránce vyskytnout. Navíc zmenšíme velikost souboru CSS, protože nebude obsahovat tolik zbytečných řádek. [6][7][8][9]

4.2 Metodika BEM

Blok, element a modifikátor vystupující pod zkratkou BEM vytvořené ruskou společností Yandex je přístup vývoje webových stránek závislý na těchto komponentech. Její hlavní využití je ve vytváření uživatelských rozhraní a zrychluje, zpřehledňuje a podporuje znuvupoužitelnost kódu bez nutnosti kopírování. Je zde podporována i modularita, kde se projekt rozdělí na jednotlivé částí, aby byl lepe zpracovatelný. V CSS je dědičnost nekonečná, takže pokud nazveme třídu stejně jako element, budou všechny části dokumentu změněny. Důležitá je tedy specifikace tříd a jejich součástí, už jen pro přehlednost. Oba zmíněné postupy práce se nedají snadno využít bez použití BEM nebo některé jiné metodiky. Problém, který BEM řeší. Využívá pro každou třídu vlastní strukturu a zbavuje se dědičnosti. Je možná lépe využitelná jako cesta a nástavba pro objektově orientované CSS a střední nebo velké projekty, ale může se využít všude. Minimálně zpřehlední práci a pomůže u dodatečných úprav, ale není nutné se jí držet při skriptem generovaným kódem a opravdu malých pro-

jektech. Kód se možná občas zdá nehezký kvůli, délce některých tříd jejich elementů a modifikátorů. Například `class="menu__item menu__item_visible menu__item_type_radio"` je opravdu hodně dlouhé, ale je z něj dobře poznat objekt, stavu a typ. [11][12][14][14]

Jedna z dalších používaných metodik je BEMIT využívané jako součást ITCSS. Pravidla jsou skoro totožná s BEM, až na používání předpon a přípon. Každá třída má předponu značící, jestli se jedná o komponent, objekt, utilitu, stav, kód Javascriptu a v neposlední řadě hack. Přípony se zde využívají pro naznačení změny vzhledu způsobené responzivním designem nebo přímo označí třídu, které se například stará o přípravu na tisk stránky. Můžete také rychle označit Vámi žádanou příponu a hned na stránce najít, kde se nachází a kolik jich existuje.[11][12][14][14]

Jedno z důležitých pravidel je nepoužívat selektory `#id` pro CSS, protože by se na stránce měly vyskytovat pouze jednou. Můžou se samozřejmě použít jako součást Javascriptu nebo ukotvením stránky. [11][12][14][14]

Blok označuje oddělený komponent označující určitý objekt stránky. Měl by být co nejvíce obecný, aby se mohla zaručit jeho znovu použitelnost. Můžeme například použít `.button`, ale `.TopLeftButton` nebo `.top-left-button` je zakázané. Dva poslední příklady by se totiž nemohly použit nikde v kódu nebo by jejich pojmenování nebylo přesné a další vývojář by nemusel poznat, kde se daná třída využívá. [11][12][14][14]

4.3 ACCS

Atomické CSS je psaní všech opakujících se vlastností jako samostatné třídy. Může se například jednat o posunutí textu v odstavci o danou vzdálenost. V atomickém myšlení si vytvoříme jednoduše třídu, která pouze pohne objektem a můžeme jí libovolně používat podle potřeby. Těchto tříd může být v každém objektu nespočetné množství, takže může být HTML kód plný zkratek jednotlivých parametrů.[15]

4.4 SMACSS

Škálovatelná modulární architektura je průvodce rozdělující CSS kód do 5 kategorií. Každá z nich má potom vlastní soubor. Dělá to tedy kód přehlednější, když potřebujete najít a změnit daný parametr nebo řádkek. Kategorie jsou base, layout, module, state a theme.[13]

- Base: skládá se z hodnot vyskytujících se na celé stránce ve všech elementech, jako výplň, okraj, ohraničení a písmo.
- Layout: rozdělí stránku na sekce s elementy hlavička, patička, hlavní stránka atd. Obsahují většinou jeden nebo více modulů a často se zde využívá předpony značící tuto třídu.
- Module: obsahuje znovu použitelné části stránky jako navigaci, galerii, a opakující se elementy.
- State: popisuje jak jednotlivé moduly a sekce budou vypadat v různých stavech, aktivní, neaktivní nebo skrytý. Mají většinou předponu `is-` například `is-active`
- Theme: podobné jako state, ale použitelné hlavně pro rozsáhlé stránky se společnými elementy, které vypadají různě v rámci stránky.

Uvádím zde jen příklad. Použití jmen a pořadí je na Vás. Stačí jen přidat dokumentaci popisující cestu, ke které jste se odhodlali. [13]

4.5 ITCSS

Myšlenka Invertovaného troúhelníku CSS se zakladá na principech lepší organizace kódu v rozdílných velikostech jednotlivých projektů. Vychází z postupů zavedených v OOCSS, ale klade větší důraz na individuální rozdělení. Můžeme současně s ním použít i jiné metodologie podle vlastních potřeb. Klíčové je rozdělení CSS stránky na vrstvy.[16] [17]

- Settings: měl by obsahovat preprocessované písmo a použité barvy, ale žádné CSS.
- Tools: mixins a funkce bez samotného CSS kódu.
- Generic: protože různé prohlížeče stylují některé elementy samy, obsahuje reset CSS pro jednotnou základnu. https://meyerweb.com/eric/tools/css/reset/
- Element: stylujeme samotné HTML elementy a to bez tříd.
- Objekt: CSS třídy podle pravidel z OOCSS.
- Components: třídy stylující veškerou estetickou část webové stránky. V celé struktuře by měla být nejobsáhlejší
- Utilities: velmi specifické třídy přepisující některé styly.

Takto podobně by mohla struktura vypadat, ale protože nejsou žádná pevná pravidla, je zcela na vás, jak si strukturu rozvrhnete. Preprocessory nejsou nuceny a můžete tím pádem úplně přeskočit první z dvou. Možností je zde opravdu nespočet. Dobré společné využití můžeme najít třeba s BEMIT, která může být adoptována pro ještě větší přehlednost. Metodologii chybí otevřená dokumentace. Je tedy vzácné najít stránky, které jí používají.[16] [17]

5 Frameworky

S front endem souvisí i Frameworky. Jedná se o knihovny obsahující celé řady funkcionalit pro různé využití. Předchází se jimi opakováním kódu a zbytečných chyb nebo řešení jíž dříve vyřešených problémů. Tímto vším zkracují čas, který je potřebný k vývoji. Můžou se například starat o řízení webové aplikace a všech prvků s ní spojených. Stačí ji tedy vizuálně rozvrhnout a zbytek je už na implementaci frameworku.

Rozdíly jsou většinou ve flexibilitě využití. Některé mají pouze základní kostru a další již hotové aplikace, když se držíme pravidly framework popisující. Záleží na vyvážení zmíněné flexibility a funkcionality.

Výhodou je, že používáme kód, který prošel přes tisíce rukou. Bezpečnost je tedy zaručena a krátí se tak čas strávený testováním. Aktualizace se také může stát jednoduší. Nemusíme zbytečně předělávat projekt při změně šifrování hesel. Přidání nově podporované platební brány může také proběhnout bez zásahu. Vedou nás také k dobrým programovacím praktikám. Třeba podle modelu MVC, rozdělením uživatelského prostředí na oddělené moduly měnící se podle potřeby. [18]

5.1 Foundation 6

V tomto projektu se bude využívat framework Foundation 6. Přesněji verzi 6.5.3. Pomůže nám při vytvoření stránky, která bude sloužit pro porovnání s metodologiemi. Odlišíme tak od sebe vytvořené styly a postavím metodologiím populárního soupeře. Cíl zde bude najít rozdíly, které mezi sebou mají a porovnat využití. Samotný Foundation je totiž dost flexibilní a dovoluje velice rychle vytvářet prototypy stránek nebo aplikací.

Knihovnu můžete využívat i bez instalování. Pomůže webový repositář.

```
1 <link rel="stylesheet" href="https://cdnjs.cloudflare.com/
2 ajax/libs/foundation/6.5.3/css/foundation-float.css">
```

Příklad 5: Repozitář Foundation 6

Nyní máme přístup k celé knihovně. Foundation se v této verzi rozlišuje hlavně v rozvržení stránky podle sloupců. Zde je stránka rozdělená do 12 bloků. Pokud použijeme všech 12 bloků, zaberou celou šířku stránky.

6 Specificita

Jedná se o velmi důležité téma z pohledu prohlížečů na CSS kód, protože se dá zapsat hned několika způsoby. Každý z nich se váže k imaginární hodnotě, kterou mu přidělí samotný prohlížeč a upřednostní tak styl, který je popsán. Hodnoty zde představují řady vypadající následovně: 0,0,0,1 0,0,1,0 0,1,0,0 1,0,0,0. Můžeme si je představit jako čísla od 1 do 1000, ale nefungují dekadicky, pokud máte řadu 0,0,1,2 není to 12. Vždy je nejdůležitější nejvyšší hodnota, která je nejvíce vlevo. Například 0,0,2,0 má vyšší prioritu než 0,0,1,0.[19]

Pod hodnotou 1 se skrývají jednotlivé tagy HMTL, když je stylujeme v CSS kódu. Příkladem třeba tag `p`. Specificitu 10 mají všechny třídy, pseudo-třídy a atributy. Mezi ně patří všechno, co začíná v CSS . nebo :. Pokračují id selektory, které mají 100. Poslední a ty, které přebijí všechno jsou styly zapsané přímo v tagu HMTL. Výjimkou je zde `!important` ta bude mít přednost v jakémkoliv případě. Pokud se sejdou dva CSS příkazy se stejnou hodnotou, prohlížeč vybere ten, který je v souboru zapsán jako poslední.[19]

```
 ul li label{
  display: block;
  text-align: center;
  padding: 10px 60px;
  text-decoration: none;
  color: #000;
  line-height: 2.8;
}
li ul li label{
  display: block;
  text-align: left;
  padding: 20px 30px;
  color: red;
  line-height: 2.8;
}
```

Příklad 6: Specificita ukázka 1.

Příklad ukazuje situace, kdy prohlížeč vybere možnost, která je více specifická v tomto případě to bude `li ul li label`. Pokud totiž spočítáme hodnoty, vyjde nám u prvního specificita 0,0,0,3 a u druhého 0,0,0,4.

```
1 #green div{
2     color:green;
3 }
4 .blue{
5     color:blue;
6 }
7 <div id="green" class="blue">...</div>
```

Příklad 7: Specificita ukázka 2.

Barva textu bude zelená, protože selektor id má vyšší hodnotu specificity.

```
1 #green div{
2     color:green;
3 }
4 .blue{
5     color:blue;
6 }
7 <div id="green" class="blue" style="color: red">...</div>
```

Příklad 8: Specificita ukázka 3.

Zde by prohlížeč vybral červenou barvu textu, protože je napsaná nejblíže. Jediná možnost, jak zde změnit barvu na jinou, je použít s třídou `!important`. Měla by přijít na řadu jako poslední, protože má nejvyšší prioritu.[19]

Abychom se vyvarovali všem těmto problémům s počítáním priority našich příkazů, je jednoduší držet se vždy pouze jednoho řádu. Nejlépe si vybrat normální CSS třídy a psát příkazy jen do nich, pokud to jde. [19]

Specificita se zmiňuje, protože se přímo váže k metodologiím, které jsou tématem této práce. Obě nám přesně určují, jak máme náš kód psát, a tím zamezují jakýmkoliv problémům, které můžou nastat, když píšeme kód. Pokud bychom používali jen OOCSS, musíme psát každý styl do samostatné třídy. Neměly by se nikde v kódu vyskytovat in-line nebo id styly. Zároveň je také doporučeno, abychom je pojmenovávali co nejpřesněji. Vůbec zde nevadí, když je název dlouhý a doslovně vysvětluje funkci daného stylu. Zvyšujeme tak nejen specificitu pro prohlížeč, ale pomáháme komukoliv dalšímu, aby se v kódu vyznal. Když bychom totiž psali styly vázané na strukturu našeho HTML, ušetříme tím nejspíš pár řádek, ale vyznat se v závorkách je někdy větší problém. BEM v tomto ohledu ještě zpřísňuje pravidla. Také podle ní musíme psát styly do jednotlivých tříd, ale přidává k tomu ještě svoje pravidla bloků, elementů a modifikátorů. Kód tím navyšuje mnohonásobně svoji specificitu. Jsme zde sice svázáni ohledně pojmenovávání, kdy nemůžeme doslovně vyjmenovat všechny jeho funkce, ale to neznamená, že ztrácíme přehlednost. [8][11]

Je tedy důležité specificitu pochopit a vědět, jak jí využít ve svůj prospěch a kdy nám může komplikovat práci, když se snažíme změnit určitá nastavení. Například jinak zarovnávat text, když máme přímo v HTML elementu nastavené něco jiného. Třídou v CSS bychom to až na jednu, už zmíněnou výjimku. nedokázali.

7 OOCSS

Pár základních informací bylo zmíněno a teď se ponoříme do hloubky. Vznikla po vzoru objektových programovacích jazyků. Stejně jako v nich je i tady důležitá znovu použitelnost. Vlastně je to ten nejdůležitější princip. Není potřeba nic instalovat, jen se naučit pravidla.

7.1 První pravidlo

Příkladem může být klasické tlačítko, třeba u formuláře, abychom mohli naše údaje poslat dál. Normálním přístupem bychom vytvořili třídu `.button` nastavili jí velikost, barvu pozadí a pak využili styl na stránce. Tlačítko je tedy vytvořeno a všechno je v pořádku. Nastane ale případ, kdy potřebujeme další tlačítko. Pro stejnorodost vizuální stránky má mít stejné dimenze, ale jinou barvu pozadí označující funkci. Máme zde tedy několik možností. Vytvoříme úplně stejnou třídu, pouze s jiným pozadím. Tímto zbytečně plýtváme místem. Můžeme také vytvořit styl, který bude měnit pozadí podle toho kde ve struktuře HTML se tlačítko nachází. Zde můžeme zaručit, že kdokoliv, kdo po nás přebere kód, bude dlouho zkoumat, jak funguje, vázat se na strukturu je tedy také nežádoucí. Když vytvoříme obecný styl pro tlačítko a další třídy, které pak můžeme přidávat, kde je potřeba, budeme mít sice delší zápis v HTML, ale můžeme pak mít libovolný počet tlačítek, kterým přidání třídy navíc změníme důležitý styl. Toto je první pravidlo, kterým bychom se měli řídit, pokud se metodologii rozhodneme použít. Najít společné vizuální vlastnosti a oddělit je od těch které se můžou v rámci projektu měnit. Může to vypadat následovně.[8][9][10][20]

```
.button{
    width: 100px;
    height:50px;
    border: 1px solid transparent;
    background-color: blue;
}
.button-2{
    width: 100px;
    height:50px;
    border: 1px solid transparent;
    background-color: red;
}
```

Příklad 9: Tlačítko bez pravidla OOCSS

Pokud pravidlo aplikujeme s předpokladem, že tlačítko bude modré, když budeme něco posílat a červené, když rušit, zbytečně duplikujeme kód. Když se budeme řídit metodikou, dostaneme následující.

```
.button{
    width: 100px;
    height:50px;
    border: 1px solid transparent;
}
.button-send{
    border-color:blue;
}
.button-cancel{
    border-color:red;
}
```

Příklad 10: Tlačítko podle OOCSS

Dostaneme tak třídu navíc, ale naše tlačítko se muže vyskytovat na stránce

hned několikrát. Stačí pouze přidat třídu měnící jeho pozadí. Naše CSS se tak stává přehlednějším a znovupoužitelnost je zaručena.

7.2 Druhé pravidlo

Druhé pravidlo již bylo zmíněno v předchozím odstavci. To je oddělení od struktury. Nelze tedy vázat styl na jeho lokaci v HTML. Tím jsme odkázaní na vytváření tříd. Stejně jako předchozí pravidlo, vylepšuje čitelnost našeho kódu. Radí tedy vytvářet unikátní třídy pro unikátní elementy tzn. nepsat CSS následovně.[8][9][10][20]

```
1 .menu ul li label {
2   display: block;
3   text-align: center;
4   padding: 10px 60px;
5   text-decoration: none;
6   color: #000;
7   line-height: 2.8;
8 }
```

Příklad 11: CSS vázaný na HMTl strukturu

Jde zde sice poznat, že bude styl využíván v menu, ale těžko se bude měnit. Nejde ho použit nikde, kde nejsou splněny podmínky struktury. V případě, že půjdeme podle pravidla, vznikne nám následující.

```
.menu-item{
  display: block;
  text-align: center;
  padding: 10px 60px;
  text-decoration: none;
  color: #000;
  line-height: 2.8;
}
```

Příklad 12: CSS nevázaný na HTML

Výstupem bude vždy třída. Přidá to možná práci v HTML, kde se nám ke každému elementu musí vázat i třída, ale je lépe vidět, co ovlivňuje.

7.3 Knihovny

Metodologie radí si vytvořit svojí vlastní knihovnu abstraktních objektů, které se na stránce můžou objevit abychom nemuseli u každého nového projektu opakovat a vytvářet stejné věci. Postupem času si tak vytvoříme spoustu objektů, ke který můžeme vždy použít a vzít si zrovna to, co potřebujeme tím, že připojíme tento soubor k naší stránce. Celá metodologie je tedy postavená na znovupoužitelnosti a vytvářením této knihovny si vytvoříme vlastní framework pro další projekty. [20]

7.4 Kód podle pravidel

Když teď aplikujeme obě pravidla získáme kód oddělený od opakujících vizuálních informací.

```
.menu-item{
  display: block;
  padding: 10px 60px;
}
```

```
5  .menu-text{
6    text-decoration: none;
7    color: #000;
8    line-height: 2.8;
9    text-align: center;
10 }
```

Příklad 13: CSS podle obou pravidel OOCSS

Příkladem číslo 13 jsme náš kód připravili na situace, kdy můžeme měnit viditelné části kódu podle potřeby. Je tím myšlena barva písma, zarovnání apod. Zde je nutné vždy přemýšlet, jak moc se může v průběhu času náš kód měnit. Nikdy nezaručíme, že toto rozdělení bude nejideálnější, ale dává to alespoň možnost lepšího přístupu k jednotlivým stylům.

Když postupujeme podle zmíněných pravidel získáváme nejen výhody objektového programování, ale i jednoznačnou strukturu. Dá se na ní jednoduše navazovat. Zamezíme i opakováním stejného CSS, a tím zmenšíme velikost souboru. Dojde tedy i ke zrychlení načítaní stránky. Oddělením od struktury nám také nemusí záležet na kontextu, kde se dané CSS nachází a můžeme jej použít, kde je potřeba.

I tato metodologie má své nevýhody. Rozhodně je jedna z nich psaní více tříd do elementů v HTML. Nemusí tedy být úplně přehledná pro někoho, kdo nezná aspoň základní principy. Zároveň je možná zbytečně komplikovaná při využití pro malé projekty. Nikdy není psáno, že jí musíme využívat pro celý dokument. Stačí tedy, abychom postupovali podle pravidel a můžeme s její pomocí měnit jen styl třeba již zmíněného tlačítka.

8 BEM

Vychází ze stejných pravidel, jako OOCSS a je to spíš nástavba, která nám určuje, jak pojmenovávat jednotlivé části našeho kódu. Tyto pravidla jsou velmi důležitá, protože nám umožní rozpoznávat vazby našeho kódu, jak v CSS tak v HTML. Na první pohled bude možné poznat, že se třída A určitým způsobem váže ke třídě B a co by měla představovat. Rozděluje jednotlivé objekty na bloky, elementy a modifikátory podle jejich pojmenování. Bloky by měly představovat rodiče, elementy jejich potomky a modifikátory stavy, nebo styly, které nijak zásadně nemění vizuální podobu. Z těchto důvodů existují pravidla, jak jednotlivé části pojmenovat. Způsobů tohoto pojmenování je hned několik a podílí se na nich hlavně podtržítka a pomlčky, které nám jednotlivé vazby představují. Zásadní je zde vybrat si pouze jeden způsob, jak metodiku v projektu využívat, protože by pak bylo zbytečné se jí řídit, když vybereme hned několik způsobů pojmenování najednou. Dokumentace, jako první příklad představuje rozdělení vazeb za pomocí podtržítek.[11][21][22][23]

```
1 .jmeno-bloku__jmeno-elementu_jmeno-modifikatoru_hodnota
```

Příklad 14: Pojmenování tříd v BEM

Všechno jsou to malá písmena latinkou, kde první je název bloku poté následuje dvěma podtržítky oddělený element a pak jedním podtržítkem oddělený modifikátor. V případě, že chceme víceslovný název kterékoliv části, musíme využit pomlčku, která odděluje jednotlivá slova. Modifikátory se můžou také vázat přímo na bloky a to znovu za pomocí jednoho podtržítka. Je zde důležité říct, že v případě, že potřebujeme k modifikátoru přidat hodnotu znovu použijeme jedno podtržítko.[23]

Dokumentace zmiňuje několik dalších možností, jakým stylem můžeme pojmenovávat třídy za pomocí metodologie BEM, ty najdete níže.[23]

```
1 .jmenoBloku__jmenoElementu_jmenoModifikatoru_hodnota
```

Příklad 15: Pojmenování tříd v BEM v s CamelCase stylem

```
.JmenoBloku_JmenoElementu_JmenoModifikatoru_Hodnota
```

Příklad 16: Pojmenování tříd v BEM s REACT stylem

```
.jmeno-bloku__jmeno-elementu--jmeno-modifikatoru--hodnota
```

Příklad 17: Pojmenování tříd v BEM s pomlčkou

Tímto BEM ale rozhodně nekončí, jak můžete vidět stále se zde opakují slova, jako blok, element a modifikátor, jsou to vlastně komponenty s jejichž pomocí skládáme jednotlivé objekty v CSS.[23]

8.1 Blok

Blok můžeme označit jako entitu, která není nijak závislá na struktuře HTML a obsahuje styl nebo vlastnosti objektů. Právě kvůli nezávislosti podporují zásadní znovupoužitelnost, takže jich na stránce můžeme využít hned několik stejných, pokud je potřeba. Můžou se libovolně vkládat do sebe, nebo využívat i v jiných projektech a usnadňovat si tím práci.[11][21][22][23]

```
.header{
    ...
}
.menu{
    ...
}
.grid{
    ...
}

<div class="menu">...</div>
```

Příklad 18: Blok v BEM

Vidíme zde zápisy tříd jednotlivých bloků, které jsou v tomto případě `.header`, `.menu` a `.grid`. Zápis je zde jen pro ukázku různých pojmenování bloků v jednom souboru, ale BEM radí dávat každý blok do odděleného souboru nebo složky. K souborové struktuře se v práci ještě dostaneme.

8.2 Element

Elementy jsou, komponenty kódu, které můžeme využívat pouze v případě, že jsou uvnitř bloku, neměli by tedy mít mimo blok žádný smysl. [11][21][22][23]

```
1  .menu__item{
2      ...
3  }
4  .menu__logo{
5      ...
6  }
7  <div class="menu">
8      <div class="menu__logo">...</div>
9      <a class="menu__item">...</a>
10 </div>
```

Příklad 19: Elementy v BEM

Na ukázce vidíme 2 elementy bloku `menu` a to `menu__logo` a `menu__item`, které v tomto případě označují logo stránky a jednotlivé položky menu. Musí být vždy součástí nadřazeného bloku, nesmí se tedy stát, že bychom použily tyto elementy mimo blok `menu`.

8.3 Modifikátor

Modifikátory by se vždy měly vázat přímo na blok nebo element bloku a používáme je pro rozšíření vlastností nebo nijak zásadní změně stylu.[11][21][22][23]

```
.header_theme-dark{
background: black;
}
.button_send{
    background-color: #1779ba;
    color: fefefe;
    font-family: inherit;
    font-size: 0.9rem;
    line-height: 1;
    text-align: center;
}
<button class="button button_send">...</button>
```

Příklad 20: Elementy v BEM

Na ukázce je vidět, jak můžeme rozšířit některé bloky o nové vlastnosti, ale stále mít možnost je používat na více místech. První příklad pouze ukazuje, že pomocí modifikátoru dokážeme přidat styl pozadí pro záhlaví tedy blok `header`, které můžeme kdykoliv vyměnit za některý jiný modifikátor, pokud bude potřeba. Dále je ukázka rozšíření bloku `button`, který tímto změníme na styl tlačítka využívaného pro odesílaní. Získáme tak možnost vyměnit tento modifikátor, v případě, že stejné tlačítko použijeme jako potvrzovací, za jiný. Zvyšuje se nám takto znovupoužitelnost našeho kódu.

8.4 Souborová struktura

Dokumentace má i svojí vlastní doporučenou souborovou strukturu, která radí jak jednotlivé bloky, elementy a modifikátory ukládat.[23]

```
projekt                                         složka projektu
    bloky                                       složka všech bloků
        header                                  složka bloku header
            __elementy                          složka elementů bloku header
                _modifikatory                   složka modifikatorů elementu v bloku header
                    header__elementy_modifikator    css implementace modifikátoru elementu v bloku header
                header__element.css             css implementace elementů bloku header
            _modifikatory                       složka modifikátorů bloku header
                header_modifikator.css          css implementace modifikátorů bloku header
            header.css                          css implementace bloku header
```

Obrázek 2: Doporučená struktura podle BEM

Na obrázku můžete vidět, jakou strukturu bychom měli vytvářet pro správné využívání BEM. V projektu je ale využita pouze základní myšlenka, která nám říká, že jednotlivé bloky máme mít v oddělených souborech. Důvod byl, že pokud se řídíme dokumentací, vytvoříme i v malém projektu obrovské množství složek pouze s jedním souborem a jedním řádkem a bude komplikované se ve struktuře vyznat. Používáme tedy vždy osamocené soubory obsahující veškerý kód, který se váže k bloku. Souborů bude stejně velký počet, ale vylepší se takto možnost jednotlivé položky kódu hledat.

9 Porovnání a příklady

9.1 Porovnání metodik

Nyní víme pravidla obou metodik a způsoby jakým se používají. Víme, že obě metodiky přistupují k CSS objektově a BEM přidává jednotlivým objektům jména podle vazeb. Jak tedy vypadá kód, když ho napíšeme pouze v OOCSS a poté s metodologií BEM?

```
/*OOCSS*/
.header{
    position: relative;
    width: 100%;
    top: 0;
    display: block;
    padding: 7px 10px 5px 10px;
    font-size: 2em;;
    text-align: center;;
    color: white;;
}
.dark-background{
    background: black;;
}
/*BEM*/
.header{
    position: relative;
    width: 100%;
    top: 0;
    display: block;
    padding: 7px 10px 5px 10px;
    font-size: 2em;;
    text-align: center;;
    color: white;;
```

```
25 }
26 .header_theme-dark{
27     background: black;;
28 }
```

Příklad 21: Porovnání OOCSS proti BEM

Vidíme funkčně zcela stejný kód, který má hlavní rozdíl v pojmenování tříd. V OOCSS nemáme nijak nařízeno, jak se mají jednotlivé třídy pojmenovávat, nebo čím indikovat vazbu k třídě jiné. Proto se může třída `dark-background` použít kdekoliv v kódu a nijak tím neporušíme žádné pravidlo, ale také nemusíme vědět kde třídu najit v CSS souborech v případě, že jich používáme v projektu více. Naproti tomu máme BEM, který se právě vyznačuje tím, že nám přímo v pojmenování jednotlivých tříd řekne, ke kterému objektu se třída vztahuje a kde jí najít. V tomto případě bychom třídu měli najít napsanou společně s blokem `header` a je také vidět, že může ovlivňovat pouze tento blok.

9.2 Příklady a porovnání s normálním CSS

Nyní uvidíte porovnání s různými styly napsaní CSS bez použití metodik a poté s nimi.

```
<div class="row">
  <h3>Ikony ve Foundation</h3>
  <div class="large-4 columns text-center">
    <div class="round">
      <i class="fa fa-university" aria-hidden="true">
    </div>
    <hr>
    <p class="text-left">
    ...
    </p>
  </div>

<div class="grid">
  <h3 class="heading heading_3">Ikony v BEM a OOCSS</h3>
  <div class="grid__line grid_size_33">
    <div class="media">
      <div class="icon icon_size_s icon__border icon__border_size_s icon__border_round media__icon_center">
        <i class="fa fa-university" aria-hidden="true"> </i >
      </div>
      <hr class="indentation indentation_line">
      <div class="media__content">
        <p class="paragraph  paragraph_align_right">
        ...
        </p >
      </div>
    </div>
  </div>
```

Obrázek 3: Ikony - zápis v HTML

Soustřeďme se u obou ukázek na červeně podtrženou řádku. První část obrázku nám ukazuje způsob, jakým bychom mohli napsat třídu, která ovlivňuje způsob ohraničení ikony bez použití metodik. Takto napsaná třída nám nic neříká o vlastnostech, jako je velikost nebo šířka ohraničení, můžeme jen předpokládat, že bude podle jména něco kulatého. Druhá část obrázku vyobrazuje, jak tyto stejné vlastnosti napsat pomocí metodik a vidíme tedy jednotlivé komponenty, ze kterých se toto ohraničení skládá. Je zde rozhodně veliké mínus, že celý zápis v HTML zabírá mnohem více místa než když metodiky nebude používat, ale získáme tím lepší čitelnost našeho kódu a představu, které třídy ovlivňují ikonu.

```css
.icon {
  text-rendering:auto;
  -webkit-font-smoothing:antialiased;
}

.media__icon_center {
  text-align:center;
}

.icon_size_s {
  font-size:40px;
}

.icon__border {
  border: 2px #000 solid;
  text-align:center;
  padding:20px 0 0 0;
  margin:0  auto;
}

.icon__border_size_s {
  height:100px;
  width:100px;
}

.icon__border_round {
  border-radius:50%;
}
.icon__border_bold {
  border: 10px #000 solid;
}

.icon_rotate {
  transform: rotate(180deg);
  padding: 0 0 0 0;
}
```

```css
.round {
  border: 2px solid;
  border-radius: 50%;
  height: 100px;
  width: 100px;
  text-align: center;
  margin: 0 auto;
  font-size: 40px;
  padding: 20px 0 0 0;
}
.blocky {
  border: 2px solid;
  height: 100px;
  width: 100px;
  text-align: center;
  margin: 0 auto;
  font-size: 40px;
  padding: 20px 0 0 0;
}
.blockyBoldRt {
  border: 10px solid;
  height: 100px;
  width: 100px;
  text-align: center;
  margin: 0 auto;
  font-size: 50px;
  transform: rotate(180deg);
}
```

Obrázek 4: Ikony - ukázka CSS

Zde vidíte jednotlivé třídy vytvářející vlastnosti z obrázku č. 5. Vlevo je CSS napsané za pomocí metodik OOCSS a BEM a vpravo jak bychom ho mohli napsat bez nich. Na první pohled je vidět že tříd, jako takových je mnohem více u metodik, ale zároveň se bez nich v kódu zbytečně opakujeme, a tím plýtváme svůj čas. Metodiky vytvářejí jednotlivé komponenty, které nám v tomto usnadňují práci a dovolují je používat znovu a znovu. Pokud bychom chtěli vytvořit novou třídu, s jinými vlastnostmi, v normálním CSS jistě musíme zopakovat některou část kódu již z vytvořených tříd.

```
  <div class="large-12">
    <div class="iconBottom">
      <p>
        ...
      </p>
      <i class="fa fa-lock">
    </div>
  </div>
</div>


.iconBottom {
  i{
    font-size: 50px;
    padding: 5px 5px;
    margin: 0
  }
}

<div class="grid">
  <div class="grid__line grid_size_50">
    <div class="media">
      <div class="media__content">
        <p class=""paragraph  paragraph_align_center">
          ...
        </p >
        <i class="icon icon_spacing icon_size_b before fa fa-university" aria-hidden="true"> <span class="icon_size_b"> Locksmith </span></i>
      </div>
    </div>
  </div>
</div>
.icon_size_b_before:before {
  font-size: 50px;
}
```

Obrázek 5: Ikony jinak - ukázka CSS podle struktury

Znovu se zde soustřeďme na oba podtržené řádky kódu. Horní část obrázku zobrazuje, jakým způsobem by mohl vypadat kód, když bychom vlastnosti přidávali v návaznosti na strukturu HTML. Vidíme pouze jeden řádek, který právě určuje, jak se budou chovat jednotlivé elementy uvnitř tagu `div`. Níže pak lze vidět, jak tato třída ovlivňuje své potomky. Druhá část obrázku ukazuje, jak vytvořit stejné vlastnosti podle metodologií, jak je vidět, mnohonásobně se nám prodlouží zápis, a ještě musíme individuálně řešit velikost písma, protože se ikony vykreslují pomocí pseudo-třídy `before`. Znamená to, že musíme vytvořit novou třídu, která tuto skutečnost zohledňuje a řeší. Zde se může vyplatit napsat kód podle struktury, protože získáme mnohem kratší zápis, ale ztratíme znovupoužitelnost, takže záleží na preferenci.

```
<div class="row">
  <h3>Galerie ve Foundation</h3>
  <div class="large-12">
    <div class="gallery">

      <div class="slides fade">
        <div class="number"> 1 / 4 </div>
          <img src="1.jfif"class="images-tall" alt="clouds">
        <div class="caption"> Dark clouds </div>
      </div>

      <div class="slides fade">
        <div class="number"> 2 / 4 </div>
          <img src="2.jfif"class="images-wide" alt="city">
        <div class="caption"> Night city </div>
      </div>

      <div class="slides fade">
        <div class="number"> 3 / 4 </div>
          <img src="3.jfif"class="images-wide" alt="table">
        <div class="caption"> Table decorations </div>
      </div>

      <div class="slides fade">
        <div class="number"> 4 / 4 </div>
          <img src="4.jfif"class="images-tall" alt="tree">
        <div class="caption"> Christmas tree </div>
      </div>

      <a class="prev" onclick="plusSlides(-1)"> &#10094; </a>
      <a class="next" onclick="plusSlides(1)"> &#10095; </a>

      <div style="text-align:center">
        <span class="dot" currentSlide="plusSlides(1)">  </span>
        <span class="dot" currentSlide="plusSlides(2)">  </span>
        <span class="dot" currentSlide="plusSlides(3)">  </span>
        <span class="dot" currentSlide="plusSlides(4)">  </span>
      </div>
```

```
<div class="grid">
  <h3 class="heading heading_3">Galerie v BEM a OOCSS</h3>
  <div class="grid__line grid_size_100">
    <div class="gallery">

      <div class="gallery__images gallery_fade">
        <div class="gallery__text gallery__text_number "> 1 / 4 </div>
          <img src="1.jfif"class="gallery__images_positon gallery__images_size_tall" alt="city">
        <div class="gallery__text gallery__text_caption "> Dark clouds </div>
      </div>

      <div class="gallery__images gallery_fade">
        <div class="gallery__text gallery__text_number "> 2 / 4 </div>
          <img src="2.jfif"class="gallery__images_positon gallery__images_size_wide" alt="clouds">
        <div class="gallery__text gallery__text_caption "> Night city </div>
      </div>

      <div class="gallery__images gallery_fade">
        <div class="gallery__text gallery__text_number "> 3 / 4 </div>
          <img src="3.jfif"class="gallery__images_positon gallery__images_size_wide" alt="table">
        <div class="gallery__text gallery__text_caption "> Table decorations </div>
      </div>

      <div class="gallery__images gallery_fade">
        <div class="gallery__text gallery__text_number "> 4 / 4 </div>
          <img src="4.jfif"class="gallery__images_positon gallery__images_size_tall" alt="tree">
        <div class="gallery__text gallery__text_caption "> Christmas tree </div>
      </div>

      <a class="gallery__prev gallery__prev_appearance" onclick="plusSlides(-1)"> &#10094; </a>
      <a class="gallery__next gallery__next_appearance" onclick="plusSlides(1)"> &#10095; </a>

      <div class="gallery_positon" >
        <span class="gallery__dot" currentSlide="plusSlides(1)">  </span>
        <span class="gallery__dot" currentSlide="plusSlides(2)">  </span>
        <span class="gallery__dot" currentSlide="plusSlides(3)">  </span>
        <span class="gallery__dot" currentSlide="plusSlides(4)">  </span>
      </div>
```

Obrázek 6: Galerie - zápis v HTML

Na obrázku vlevo vidíme galerii vytvořenou bez pomocí metodik a vpravo s nimi. Znovu si můžeme všimnout, že metodiky způsobují zapisování mnohem většího počtu tříd, ale i delších názvů jednotlivých komponent, kterými galerii vytváříme. Když je nepoužíváme, ušetříme si trochu místa, ale přijdeme o důležité vlastnosti, jako jsou údaje o vazbách jednotlivých tříd a znovupoužítelnost. Zde mi přijde lepší používat pravidla metodik, protože se nemusíme v kódu neustále opakovat, jak je vidět na obrázcích č. 8 a 9.

```css
.gallery {
  max-width: 700px;
  position: relative;
  margin: auto;
  z-index: 1;
}

.slides {
  display: none;
}

.prev, .next {
  cursor: pointer;
  position: abslolute;
  top: 50%;
  width: auto;
  margin-top: -22px;
  padding: 16px;
  color: red;
  font-weight: bold;
  font-size: 18px;
  transition: 0.6s ease;
}

.next {
  right: 0;
}

.prev:hover, .next:hover {
  background-color: black;
}

.caption{
  color: white;
  font-size: 15px;
  padding: 8px 12px;
  position: absolute;
  bottom: 8px;
  width: 100%;
  text-align: center;
}

.number{
  color: white;
  font-size: 12px;
  padding: 8px 12px;
  position: absolute;
  top: 0;
}

.dot{
  cursor: pointer;
  height: 15px;
  width: 15px;
  margin: 0 2px;
  background-color: #bbb;
  border-radius: 50%;
  display: inline-block;
  transition: background-color 0.6s ease;
}
```

```css
.gallery {
  max-width: 700px;
  position: relative;
  margin: auto;
  z-index: 1;
}

.gallery__images {
   display: none;
}

.gallery__images_positon {
  display: block;
  margin-left: auto;
  margin-right: auto;
 }

.gallery__images_size_tall {
  width: 45%;
}

.gallery__images_size_wide {
  width: 100%;
}

.gallery__prev, .gallery__next {
  cursor: pointer;
  position: absolute;
  top: 50%;
  width: auto;
  margin-top: -22px;
  padding: 16px;
  transition: 0.6s ease;
}

.gallery__prev_text, .gallery__next_text {
  color: red;
  font-weight: bold;
  font-size: 18px;
}

.gallery__next {
  right: 0;
}

.gallery__prev:hover>, .gallery__next:hover {
  background-color: black;
  color: #1779ba;
}

.gallery__text {
  padding: 8px 12px;
  position: absolute;
}

.gallery__text_caption {
  bottom: 8px;
  width: 100%;
  text-align: center;
  font-size: 15px;
}
```

Obrázek 7: Galerie - zápis v CSS první část

Vlevo klasické CSS a vpravo CSS s BEM a OOCSS.

```css
.active, .dot:hover {
  background-color: #717171;
}

.fade{
  -webkit-animation-name: fade;
  -webkit-animation-duration: 1.4s;
  animation-name: fade;
  animation-duration: 1.4s;
}

@-webkit-keyframes fade {
  from {
    opacity: .5;
  }
  to {
     opacity: 1;
  }
}

@keyframes fade {
  from {
    opacity: .5;
  }
  to {
    opacity: 1;
  }
}
```

```css
.gallery__text_number {
  top: 0;
  font-size: 12px;
  background-color: grey;
}

.gallery_positon {
  text-align: center;
}

.gallery__dot {
  cursor: pointer;
  height: 15px;
  width: 15px;
  margin: 0 2px;
  background-color: #bbb;
  border-radius: 50%;
  display: inline-block;
  transition: background-color 0.6s ease;
}
.gallery_active, .gallery__dot:hover {
  background-color: >#717171;
}

.gallery_fade {
  -webkit-animation-name: fade;
  -webkit-animation-duration: 1.4s;
  animation-name: fade;
  animation-duration: 1.4s;
}

@-webkit-keyframes fade {
  from {
    opacity: .5;
  }
  to {
     opacity: 1;
  }
}

@keyframes fade {
  from {
    opacity: .5;
  }
  to {
    opacity: 1;
  }
}
```

Obrázek 8: Galerie - zápis v CSS druhá část

Vlevo klasické CSS a vpravo CSS s BEM a OOCSS.

```
<div class="row">
  <h3>Formulář ve Foundation </h3>
  <div class="large-12">
    <form method="post" action="#" onsubmit="return validateForm(this,['jmeno','email','text','vyber','aktiv','vyber1']))">
      <p id="errormsg" class="errormsg"> Pole označena červeně musí být vyplněna </p>
      <div class="row-form">
        <label for="jmeno"> Jmeno: </label>
        <input type="text" name="jmeno" id="in-jmeno" value="">
      </div>

      <div class="row-form">
        <label for="in-email"> Email: </label>
        <input type="email" name="email" id="in-email" value="">
      </div>

      <div class="row-form">
        <label for="in-vyber"> Vyber: </label>
        <select name="vyber" id="in-vyber" > </select>
          <option value="1"> První </option>
          <option value="2"> Druhý </option>
          <option value="3"> Třetí </option>
      </div>

      <div class="row-form">
        <label id="in-aktiv"> Aktivita: </label>
        <input name="aktiv" type="checkbox" value="1" >
      </div>

      <div class="row-form">
        <label id="in-vyber1"> Vyber: </label>
        <input name="vyber1" type="radio value="1" >
        <input name="vyber1" type="radio value="2" >
        <input name="vyber1" type="radio value="3" >
      </div>

      <div class="row-form">
        <input type="submit" class="button" value="odeslat" >
      </div>
    </form>
  </div>
</div>
```

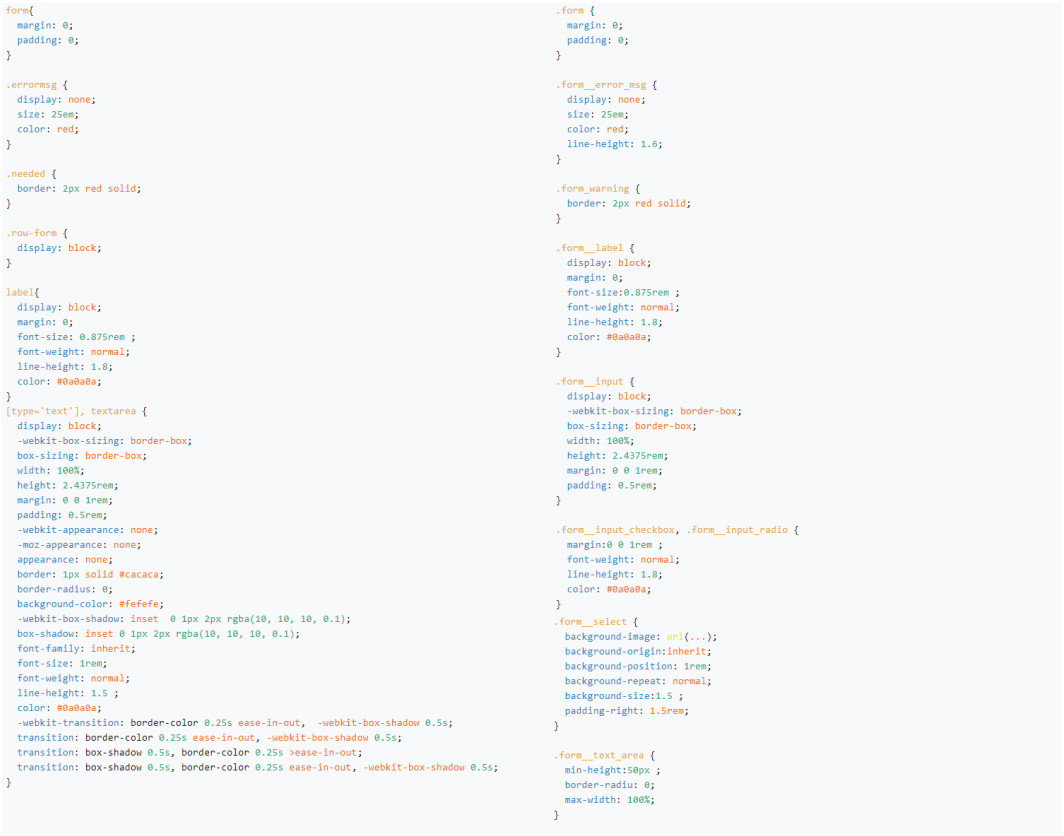
Obrázek 9: Formulář - zápis v HTML s Foundation

Ukázka zobrazuje HTML kód formuláře vytvořený za pomocí Foundation, jak je na první pohled vidět, nenajdeme zde skoro žádné CSS třídy, které označují jednotlivé styly využité pro formulář. Skoro všechny jsou totiž napsané pro typ použitého tagu nebo tag samotný a nemůžeme k nim tedy jednoduše přistupovat, pokud bychom chtěli provádět změny, ale toto je cena za to, že používáme framework, který má všechny vlastnosti nastavené a připravené k použití. Najdeme zde pouze třídy označující řádky formuláře, zobrazující varovnou zprávu a nastavující vlastnosti tlačítka odesílajícího formulář.

```
<div class="grid">
  <h3 class="heading heading_3">Formulář v BEM a OOCSS</h3>
  <div class="grid__line grid_size_100">
    <form class="form" method="post" action="#" onsubmit="return validateForm(this,['jmeno','email','text','vyber','aktiv','vyber1']))">
      <p id="form__error_msg" class="form__error_msg"> Pole označena červeně musí být vyplněna </p>
      <div class="form__row">
        <label class="form__label " for="jmeno"> Jmeno: </label>
        <input class="form__input form_appearance form_focus" type="text" name="jmeno" id="in-jmeno" value="">
      </div>

      <div class="form__row">
        <label class="form__label " for="in-email"> Email: </label>
        <input class="form__input form_appearance form_focus" type="email" name="email" id="in-email" value="">
      </div>

      <div class="form__row">
        <label class="form__label " for="in-vyber"> Vyber: </label>
        <select class="form__input form__select form_appearance form_focus" name="vyber" id="in-vyber" > </select>
          <option value="1" > První </option>
          <option value="2" > Druhý </option>
          <option value="3" > Třetí </option>
        </div>

      <div class="form__row">
        <label class="form__label " for="in-text"> Text: </label>
        <textarea class="form__input form__text_area form_appearance form_focus" name="text" id="in-text" > </textarea>
      </div>

      <div class="form__row">
        <label class="form__label " id="in-aktiv"> Aktivita: </label>
        <input class="form__input_checkbox" name="aktiv" type="checkbox" value="1" >
      </div>

      <div class="form__row">
        <label class="form__label " id="in-vyber1"> Vyber: </label>
        <input class="form__input_radio" name="vyber1" type="radio value="1" >
        <input class="form__input_radio" name="vyber1" type="radio value="2" >
        <input class="form__input_radio" name="vyber1" type="radio value="3" >
      </div>

      <div class="form__row">
        <input type="submit" class="button button_send" value="odeslat" >
      </div>
    </form>
  </div>
</div>
```

Obrázek 10: Formulář - zápis v HTML s BEM a OOCSS

Zde vidíme stejný zápis v HTML rozšířený o pravidla metodik BEM a OOCSS, kde je na první pohled vidět, že skoro každý tag má svojí vlastní třídu, která mu udává vizuální vlastnosti. Můžeme tedy velice rychle rozpoznat jednotlivé třídy a co ovlivňují a snadno poté najít, kde máme provádět změny a úpravy. Nevýhodou zde může být právě větší množství jednotlivých tříd, ale i delší názvy vytvářející delší zápis. Je zde rozhodně lépe poznat vztah jednotlivých tříd, jak se k sobě vážou a jakými komponenty vlastně formulář skládáme. Na následujících obrázkách číslo 12 a 13 je vidět, že i celý zápis v CSS je kratší, protože zbytečně neopakujeme některé vlastnosti.



```css
form{
  margin: 0;
  padding: 0;
}

.errormsg {
  display: none;
  size: 25em;
  color: red;
}

.needed {
  border: 2px red solid;
}

.row-form {
  display: block;
}

label{
  display: block;
  margin: 0;
  font-size: 0.875rem ;
  font-weight: normal;
  line-height: 1.8;
  color: #0a0a0a;
}
[type='text'], textarea {
  display: block;
  -webkit-box-sizing: border-box;
  box-sizing: border-box;
  width: 100%;
  height: 2.4375rem;
  margin: 0 0 1rem;
  padding: 0.5rem;
  -webkit-appearance: none;
  -moz-appearance: none;
  appearance: none;
  border: 1px solid #cacaca;
  border-radius: 0;
  background-color: #fefefe;
  -webkit-box-shadow: inset  0 1px 2px rgba(10, 10, 10, 0.1);
  box-shadow: inset 0 1px 2px rgba(10, 10, 10, 0.1);
  font-family: inherit;
  font-size: 1rem;
  font-weight: normal;
  line-height: 1.5 ;
  color: #0a0a0a;
  -webkit-transition: border-color 0.25s ease-in-out,  -webkit-box-shadow 0.5s;
  transition: border-color 0.25s ease-in-out, -webkit-box-shadow 0.5s;
  transition: box-shadow 0.5s, border-color 0.25s >ease-in-out;
  transition: box-shadow 0.5s, border-color 0.25s ease-in-out, -webkit-box-shadow 0.5s;
}
```

```css
.form {
  margin: 0;
  padding: 0;
}

.form__error_msg {
  display: none;
  size: 25em;
  color: red;
  line-height: 1.6;
}

.form_warning {
  border: 2px red solid;
}

.form__label {
  display: block;
  margin: 0;
  font-size:0.875rem ;
  font-weight: normal;
  line-height: 1.8;
  color: #0a0a0a;
}

.form__input {
  display: block;
  -webkit-box-sizing: border-box;
  box-sizing: border-box;
  width: 100%;
  height: 2.4375rem;
  margin: 0 0 1rem;
  padding: 0.5rem;
}

.form__input_checkbox, .form__input_radio {
  margin:0 0 1rem ;
  font-weight: normal;
  line-height: 1.8;
  color: #0a0a0a;
}
.form__select {
  background-image: url(...);
  background-origin:inherit;
  background-position: 1rem;
  background-repeat: normal;
  background-size:1.5 ;
  padding-right: 1.5rem;
}

.form__text_area {
  min-height:50px ;
  border-radiu: 0;
  max-width: 100%;
}
```

Obrázek 11: Formulář - zápis v CSS první část

Vlevo CSS z Foundation a vpravo CSS s BEM a OOCSS.

```css
[type='text'], textarea: focus{
  outline: none;
  border: 1px solid #8a8a8a;
  background-color: #fefefe;
  box-shadow: 0 0 5px #cacaca;
  transition: box-shadow 0.5s, border-color 0.25s ease-in-out;
}
select{
  height: 2.4375rem;
  margin: 0 0 1rem;
  padding: 0.5rem;
  -webkit-appearance: none;
  -moz-appearance: none;
  appearance: none;
  border: 1px solid #cacaca;
  border-radius: 0;
  background-color: #fefefe;
  font-family: inherit;
  font-size: 1rem;
  font-weight: normal;
  line-height: #1.5;
  color: #0a0a0a;
  background-image: url(...);
  background-origin: inherit;
  background-position: 1rem;
  background-repeat: normal;
  background-size: 1.5 ;
  padding-right: #0a0a0a;
  -webkit-transition: border-color 0.25s ease-in-out,  -webkit-box-shadow 0.5s;
  transition: border-color 0.25s ease-in-out, -webkit-box-shadow 0.5s;
  transition: box-shadow 0.5s, border-color 0.25s >ease-in-out;
  transition: box-shadow 0.5s, border-color 0.25s ease-in-out, -webkit-box-shadow 0.5s;
}

select{
  width: 100%;
}

select: focus{
  outline: none;
  border: 1px solid #8a8a8a;
  background-color: #fefefe;
  box-shadow: 0 0 5px #cacaca;
  transition: box-shadow 0.5s, border-color 0.25s ease-in-out;
}
textarea{
  resize: vertical;
  max-width: 100%;
  min-height: 50px;
  overflow: auto;
}

[type='checkbox'], [type='radio']{
  border: 0 0 1rem;
}

.button {
   display: inline-block;
  vertical-align: middle;
  margin: 0 0 1rem 0;
  padding: 0.85em 1em;
  border: 1px solid transparent;
  border-radius: 0;
  -webkit-transition: background-color 0.25s >ease-out, color 0.25s ease-out;
  transition: background-color 0.25s  ease-out,  color 0.25s ease-out;
  font-family: inherit ;
  font-size: 0.9rem;
  -webkit-appearance: none;
  line-height: 1;
  text-align: center;
  cursor: pointer;
}

button: hover, button: focus{
  background-color: #14679e;
  color: #fefefe;
}
```

```css
.form_appearance {
  -webkit-appearance: none;
  -moz-appearance: none;
  appearance: none;
  border: 1px solid #cacaca;
  border-radius: 0;
  background-color: #fefefe;
  -webkit-box-shadow:inset  0 1px 2px rgba(10, 10, 10, 0.1);
  box-shadow: inset 0 1px 2px rgba(10, 10, 10, 0.1);
  font-family:inherit;
  font-size: 1rem;
  font-weight: normal;
  line-height:1.5 ;
  color: #0a0a0a;
  -webkit-transition: border-color 0.25s ease-in-out,  -webkit-box-shadow 0.5s;
  transition: border-color 0.25s ease-in-out, -webkit-box-shadow 0.5s;
  transition: box-shadow 0.5s, border-color 0.25s >ease-in-out;
  transition: box-shadow 0.5s, border-color 0.25s ease-in-out, -webkit-box-shadow 0.5s;
}
.form_focus:focus {
  outline: none ;
  border: 1px solid #8a8a8a;
  background-color: #fefefe;
  box-shadow: 0 0 5px #cacaca;
  transition: box-shadow 0.5s, border-color 0.25s >ease-in-out;
}

.button {
  display: inline-block;
  vertical-align: middle;
  margin: 0 0 1rem 0;
  padding: 0.85em 1em;
  border: 1px solid transparent;
  border-radius: 0;
  -webkit-transition: background-color 0.25s >ease-out, color 0.25s ease-out;
  transition: background-color 0.25s  ease-out,  color 0.25s ease-out;
  -webkit-appearance: none;
  cursor: pointer;
}

.button_send {
  background-color: #1779ba;
  color: #fefefe;
  font-family:inherit ;
  font-size: 0.9rem;
  line-height: 1;
  text-align: center;
}

.button_send:hover {
  background-color: #14679f;
}
```

Obrázek 12: Formulář - zápis v CSS druhá část

Vlevo CSS z Foundation a vpravo CSS s BEM a OOCSS.

9.3 Preprocesory

Určitě zde musíme zmínit, jakým způsobem nám můžou preprocesory usnadnit práci při vytváření tříd, když používáme metodiku BEM. Za pomocí `&` můžeme totiž navazovat jména jednotlivých tříd, a tím celkem jednoduše rozšiřovat naše bloky o modifikace a elementy. Cenou za toto zjednodušení je velký počet ne zcela srozumitelných závorek, když náš blok obsahuje velký počet komponent a zhoršujeme tím srozumitelnost našeho preprocesorového souboru. Po kompilaci si pak musíme ověřit, že se jednotlivé třídy jmenují, jak mají a jejich vazby jsou správně podle pravidle BEM. Je zde na povážení jestli tuto možností využít, ale pokud dáváme pozor určitě si usnadníme práci, když nebudeme muset neustále opisovat jméno bloku a jeho elementu při vytvoření modifikátoru.

```
.header{
    position: relative;
    width: 100%;
    top: 0;
    display: block;
    padding: 7px 10px 5px 10px;
    font-size: 2em;;
    text-align: center;;
    color: white;;

    &_theme-dark{
        background: black;;
    }
}
```

```
.header{
    position: relative;
    width: 100%;
    top: 0;
    display: block;
    padding: 7px 10px 5px 10px;
    font-size: 2em;;
    text-align: center;;
    color: white;;
}

.header_theme-dark{
    background: black;;
}
```

Obrázek 13: Preprocesory - vlevo vstup a vpravo výstup

9.4 Zhodnocení

Pokud bychom chtěli vytvářet web za pomoci obou metodologií musíme se seznámit s jejich dobrými a špatnými stránkami. Výhodou může být, že po jejich naučení se velice těžko zapomíná, jak fungují a mění náš přístup k psaní CSS. Dále pak můžeme bez velkých obav projekt předat i někomu dalšímu, který se samozřejmě bude muset naučit, v jaké struktuře jsme metodiky používali, ale bude to určitě jednoduší než normální postup, kdy kód nebo třídy nemusí mít žádný společný základ. Hlavně si vytváříme jednotlivé bloky kódu, které se dají používat všude, jako komponenty pro vytváření dalších objektů. Nevýhodou je samozřejmě čas, který musíme vynaložit na pochopení obou metodologií. Na první pohled se zdá jednoduché je pochopit, ale problém může být rozvrhnout si správně jména jednotlivých bloků a jak do sebe budou zapadat. Může tedy nastat situace, že budete pouze přemýšlet, jak všechno srozumitelně pojmenovat. Dále se dostáváme do situace, kde pro každou maličkost musíme vytvořit třídu, nesmí se stávat, že stylujeme pomocí id nebo přímo tag. Zakázané je i kaskádování, kde stylujeme podle struktury našeho HTML, když je to potřebné. Pokud používáme souborovou strukturu podle dokumentace dostaneme výsledně obrovské množství souborů obsahující jednu až dvě řádky a tím také spoustu složek.

9.4.1 Výhody

- Srozumitelnost kódu jak v CSS, tak v HTML
- Zvýšena znovupoužitelnost
- Těžko se vrátíte do původních kolejí

9.4.2 Nevýhody

- Velké množství tříd
- Dlouhé zápisy tříd v HTML
- Spoustu složek a souborů, ale záleží na vybrané struktuře
- Časová náročnost na naučení postupů

9.4.3 Velikosti

Poslední může být porovnání velikostí souborů s frameworkem. Foudantion používám skrz webový repositář a poslední položka tedy není zcela objektivní, protože můj soubor neobsahuje ani zlomek nastavení a tříd jako celý Foudantion.

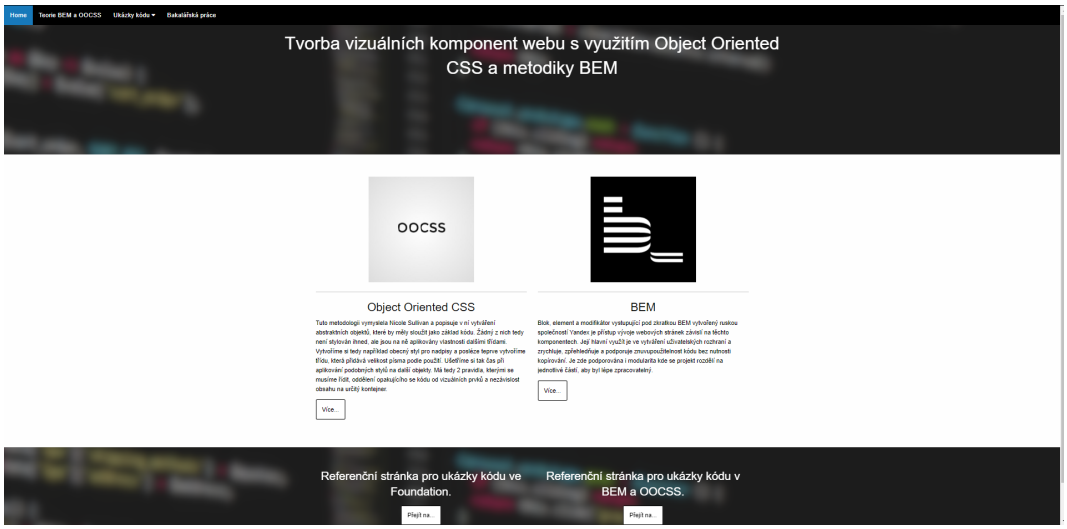
Technologie	Velikost bez minifikace preprocesorem s webovým repozitářem	Velikost s minifikací preprocesorem a webovým repozitářem	Velikost bez webového repozitáře
BEM a OOCSS	11,4Kb	9,18Kb	9,18Kb
Foundation	6,64Kb	5,11Kb	158,11Kb

Obrázek 14: Velikosti CSS souborů

10 Praktická část

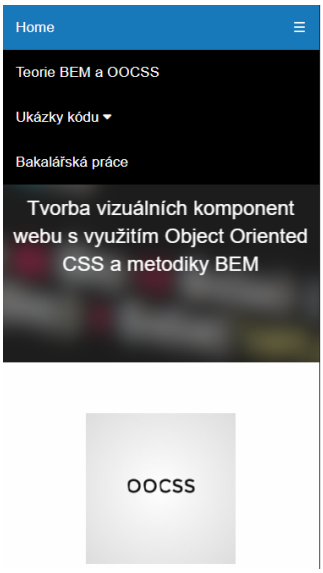
Praktická část bakalářské práce je koncipována jako webová prezentace popisující pravidla a doporučení pro správné používání metodik OOCSS a BEM a najdete ji na adrese `home.pf.jcu.cz/~smrzja02`. Stránka je vytvořena za pomocí frameworku Foundation 6 a obou zmíněných metodik. Najdete na ní všechny potřebné informace, ale i ukázky kódu s popsáním některých rozdílů, které se objeví, pokud používáme metodiky, nebo samotné CSS.

S vývojem této stránky mi pomáhalo prostředí Visual Studio Code od společnosti Microsoft. Vybral jsem si ho, protože v něm již mám zkušenost s vývojem a připadá mi, jako nejlepší a nejjednodušší pro práci, kterou jsem s ním zatím vykonával. Ke stažení je zcela zdarma pod licencí MIT a vyznačuje se podporou různých programátorských jazyku, jako je JAVA, nebo C++ a samozřejmě podporou HTML, CSS JavaScript a dalších. Můžeme si také stáhnout spoustu různých pluginů, které můžou rozšířit aplikaci o další funkce nebo dokonce přidat podporu dalších programovacích jazyků. Plugin Easy Sass třeba dokáže kompilovat všechny SASS a SCSS dokumenty přímo v aplikaci na minifikované verze bez nutnosti externích programů.



Obrázek 15: Úvodní stránka webu

Hlavička webu obsahuje jednoduchou navigaci na jednu ze čtyř hlavních stránek a odkaz na stažení bakalářské práce. Celý web je plně rezponzivní a má všechny funkce plně připravené pro prohlížení na mobilních zařízeních, kdy se horní navigační panel otevírá pomocí „hambuger“ ikony.



Obrázek 16: Úvodní stránka webu - mobilní verze

11 Závěr

Když píšeme CSS stránky normálním způsobem, budeme se dostávat do situací, kdy náš kód nebude mít žádný pevný řád a budeme se nejspíše snažit využívat ty nejjednodušší způsoby zápisu. Znamená to, že kdokoliv, kdo po nás kód převezme, bude muset dlouho tápat, jak vše funguje a která třída, id, nebo řádka dělá co. Pokud ale využijeme metodiky BEM a potažmo OOCSS zmíněných problému se ve větší míře vyhneme, protože nám dají jasný řád, jak s kódem pracovat. Navíc si budeme postupně vytvářet vlastní knihovny objektů, které se můžou používat v dalších projektech kde bude potřeba.

Problém nastává, když bychom si metodiky vybrali jako způsob naučení CSS, protože tím ztratíme rozhled, jak se dají psát různé části kódu. Rozhodně neradím metodikami začít, ale využívat je až v momentě, kdy CSS zvládáme a známe různé způsoby, jak řešit problémy pomocí CSS. Protože metodiky nevyužívají všechny funkce, které CSS podporuje a některé problémy jsou pak řešeny zbytečně složitě. Jasnou výhodou je, že nemusíme nic instalovat a pouze stačí naučit se pravidla a můžeme dle nich začít psát kód, ale určitě bude chvíli trvat, než plně pochopíme, jak do sebe všechno zapadá. Zde můžou pomoct preprocesory, hlavně u metodiky BEM, kdy nám svými funkcemi můžou ušetřit práci s opakováním některých názvů tříd.

Musí zde být také zmíněno, že právě kvůli zapisování delších názvů tříd, buď v CSS souborech, anebo přímo v HTML Vám nejspíš naroste velikost obou souborů, ale čím větší bude projekt, tím menší by měl být rozdíl při porovnání kódu s nimi a bez nich. Bude za tím stát znovupoužitelnost komponent kódu bez neustálého opakování.

Pokud jde o to, jakou metodiku si vybrat, je to zcela na Vás, ale BEM je vlastně nástavba na OOCSS, protože obě základní pravidla u ní platí také a přidává právě důležité znázornění jednotlivých vazeb kódu. S podporou v prohlížečích by snad nikdy neměl být problém, už jen kvůli tomu, že nic zásadně nemění, jsou to pouze pravidla, jak zapisovat CSS.

Na webové stránce jsou popsána pravidla pro psaní obou těchto metodik s několika radami, jak při jejich používání zefektivnit práci ještě o trochu více. Naleznete i ukázky, jak může kód vypadat s nimi a bez nich s krátkým komentářem ke každé ukázce. V neposlední řadě jsou popsány výhody a nevýhody psaní kódu s metodikami a porovnání velikostí CSS souborů.

Seznam použité literatury a zdrojů

[1] *9 Web Technologies Every Web Developer Must Know in 2020.* Tms-outsource [online]. Bělehrad (Srbsko):Milos Timotic, 2018 [cit. 2020-02-25]. Dostupné z: https://tms-outsource.com/blog/posts/web-technologies/

[2] *A brief history of CSS until 2016.* W3 [online]. Cambridge (Massachuttes): Bert Bos, 2016 [cit. 2019-04-08]. Dostupné z: https://www.w3.org/Style/CSS20/history.html

[3] *CSS4.* Css-tricks [online]. Bend (Oregon): Chris Coyier, 2020 [cit. 2020-02-25]. Dostupné z: https://css-tricks.com/css4/

[4] *An Introduction to CSS Pre-Processors: SASS, LESS and Stylus.* Htmlmag [online]. Norimberk (Německo): Bilal Cinarli, 2014 [cit. 2020-02-25]. Dostupné z: https://htmlmag.com/article/an-introduction-to-css-preprocessors-sass-less-stylus

[5] *Sass Basics.* Sass [online]. sass-lang [cit. 2020-02-25]. Dostupné z: https://sass-lang.com/guide

[6] *Understanding CSS Writing Methodologies.* Hongkiat [online]. Delhi (India): Preethi Ranjit, 2018 [cit. 2020-02-25]. Dostupné z: https://www.hongkiat.com/blog/css-writing-methodologies/

[7] *A Look at Some CSS Methodologies.* Webfx [online]. Harrisburg (Pensylvánie): William Craig [cit. 2020-02-25]. Dostupné z: https://www.webfx.com/blog/web-design/css-methodologies/

[8] *OOCSS: objektové psaní CSS.* Vzhůru dolů [online]. Praha: Martin Michálek, 2017 [cit. 2018-04-05]. Dostupné z: https://www.vzhurudolu.cz/prirucka/oocss

[9] *An Introduction To Object Oriented CSS (OOCSS).* Smashing magazine [online]. Freiburg: Smashing Media, 2011 [cit. 2018-04-05]. Dostupné z: https://www.smashingmagazine.com/2011/12/an-introduction-to-object-oriented-css-oocss/

[10] *OOCSS - The Future of Writing CSS.* Keycdn [online]. Švýcarsko: Cody Arsenault, 2019 [cit. 2020-02-25]. Dostupné z: https://www.keycdn.com/blog/oocss

[11] *'Why BEM?' in a nutshell.* Decaf [online]. Berlín (Německo): Dirk, 2019 [cit. 2020-02-25]. Dostupné z: https://blog.decaf.de/2015/06/24/why-bem-in-a-nutshell/

[12] *BEMIT: Taking the BEM Naming Convention a Step Further.* Csswizardry [online]. Leeds (Anglie): Harry Roberts, 2015 [cit. 2020-02-25]. Dostupné z: https://csswizardry.com/2015/08/bemit-taking-the-bem-naming-convention-a-step-further/

[13] *BEM and SMACSS: Advice From Developers Who've Been There.* Sitepoint [online]. Melbourne (Austrálie): Patrick Catanzariti, 2015 [cit. 2020-02-25]. Dostupné z: https://www.sitepoint.com/bem-smacss-advice-from-developers/

[14] *Making CSS/HTML readable — BEMIT it?* Medium [online]. ACGoff, 2019 [cit. 2020-02-25]. Dostupné z: https://medium.com/@acgoff/making-css-html-readable-bemit-it-2f0c88b38f16

[15] *In defense of Functional CSS.* Critter[online]. Greenville (Jižní Karolína): Mike Crittenden, 2018, 2018 [cit. 2020-02-25]. Dostupné z: https://critter.blog/2018/06/08/in-defense-of-functional-css/

[16] *Intro to ITCSS for Web Developers.* Hongkiat [online]. Massachusetts: Jake Rocheleau, 2018 [cit. 2020-02-25]. Dostupné z: https://www.hongkiat.com/blog/inverted-triangle-css-web-development/

[17] *ITCSS: Scalable and Maintainable CSS Architecture.* Xfive [online]. San Francisco (Kalifornie): Lubos Kmetko, 2016 [cit. 2020-02-25]. Dostupné z: https://www.xfive.co/blog/itcss-scalable-maintainable-css-architecture/

[18] *10 Best CSS Frameworks for Front-End Developers.* Geekflare [online]. London: Ankush Thakur, 2020 [cit. 2020-04-26]. Dostupné z: https://geekflare.com/best-css-frameworks/

[19] *CSS Specificity.* W3schools [online]. Sandnes, Norsko: w3schools, c1999-2020 [cit. 2020-04-26]. Dostupné z: https://www.w3schools.com/css/css_specificity.asp/

[20] *OOCSS - The Future of Writing CSS.* Keycdn [online]. Switzerland: Cody Arsenault, 2019 [cit. 2020-04-26]. Dostupné z: https://www.keycdn.com/blog/oocss

[21] *BEM 101.* Css-tricks [online]. San Francisco (Kalifornie): Robin Rendle, 2016 [cit. 2020-04-26]. Dostupné z: https://css-tricks.com/bem-101/

[22] *Get BEM* [online]. Yekaterinburg: Strukchinsky, 2016 [cit. 2020-04-26]. Dostupné z: http://getbem.com/

[23] *BEM* [online]. Moskva: Yandex, 2018 [cit. 2020-04-26]. Dostupné z: https://en.bem.info/methodology/

Seznam obrázků

Seznam příkladů

A Příloha

1. CD se zdojovými kódy celé aplikace a plné znění BP v PDF

2. Webová stránka: **home.pf.jcu.cz/~smrzja02**