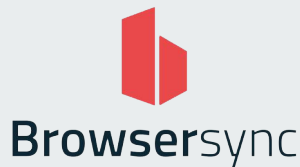




Błędy i problemy w CSS

Na podstawie serwisów tworzonych wewnątrz IT S / Z / G / Warsaw



BEM

BLOCK • ELEMENT • MODIFIER

<https://en.bem.info/methodology/>

✓ używane zasady

✗ "martwe" zasady

Nasze modyfikacje (i doprecyzowania) w BEM-ie

wybrane zasady →

<http://confluence.arsthanea.com/pages/viewpage.action?pageId=1933650>

- modyfikator jako oddzielna klasa,
czyli `.block__element -modifier`
zamiast `.block__element_modifier`
- ~~stosujemy `+takie-nazwy-klas`
- z myślnikami, zamiast `camelCase`~~
- ~~przejściowe stany, w odróżnieniu od
stałych modyfikatorów, oznaczamy klasą
`.is-costam` (np. `.is-active`)~~
- style responsywne dotyczące danego
modułu są zadeklarowane na jego końcu
- ~~unikamy stosowania czystych tagów html
jako selektorów~~
- ~~"style.scss - main SASS file gathering
all other files into one"~~

1

Błędy wynikające ze złego używania BEM-a

1.1. Używanie camelCase'ów w nazwach klas



CSS selector naming convention

- Names of BEM entities are written using **numbers** and **lowercase Latin characters**.
- Individual words within names are separated by a **hyphen** (-).
- Information about the names of blocks, elements & modifiers is stored using **CSS classes**.



1.2. Stylowanie bloków wewnątrz innych bloków

Jest to złamanie fundamentalnej idei BEM-a:

Block

*A logically and functionally independent page component, the equivalent of a component in Web Components. A **block** encapsulates behavior (JavaScript), templates, styles (CSS), and other implementation technologies.*

```
.products-category {
  &__button {
    position: absolute;
    right: 20px;
    top: 10px;

    .button {
      font-size: 12px;
      height: 30px;
      line-height: 30px;
      padding: 0 15px;
      svg {
        height: 30px;
        margin-left: 5px;
      }
    }
  }
}
```

<https://en.bem.info/methodology/key-concepts/#block>

1.2. Stylowanie bloków wewnątrz innych bloków

By zmodyfikować wygląd danego bloku - w zależności od jego kontekstu - należy użyć:

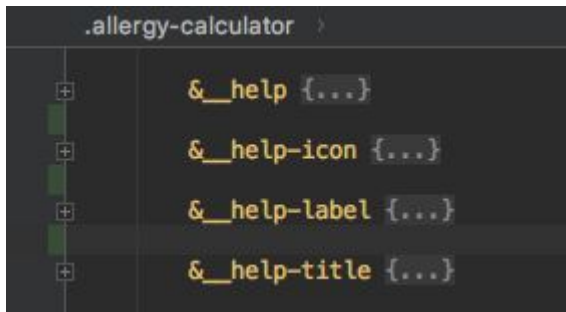
- **modyfikatora**
jeśli przewidujemy, że blok o takim wyglądzie może być użyty gdzieś indziej
- **mixa**
jeśli wygląd zależy od specyficznego kontekstu i nie będzie więcej używany

```
.button
107 &.-disabled {
108   cursor: default;
109   pointer-events: none;
110   opacity: .5;
111 }
112
```

```
12
13 &__button {
14   position: absolute;
15   bottom: 20px;
16 }
17
```

<https://en.bem.info/methodology/faq/#what-is-the-correct-way-to-modify-the-appearance-of-every-block-instance-on-a-page>

1.3. Nadużywanie elementów vs. tworzenie nowych bloków



Create an element

If a section of code can't be used separately without the parent entity (the block).

The exception is elements that must be divided into smaller parts – subelements – in order to simplify development.



W powyższym przykładzie powinien zostać utworzony nowy blok `.allergy-calculator-help` i w razie potrzeby (pozycjonowanie) zmiksowany z elementem `.allergy-calculator__help`

<https://en.bem.info/methodology/quick-start/#should-i-create-a-block-or-an-element>

1.4. Stylowanie po elementach HTML



Selectors

*BEM doesn't use tag and ID selectors.
The styles of blocks and elements
are defined by class selectors.*



<https://en.bem.info/methodology/css/#selectors>
<https://en.bem.info/methodology/faq/#can-i-use-a-css-reset>

```
.diet-calendar > &__week  
  
a {  
  display: inline-block;  
  position: relative;  
  padding: 10px 5px;  
  text-align: center;  
  color: $gray;  
  margin: auto;  
  
  strong {  
    display: block;  
    color: $blue;  
    text-transform: capitalize;  
  }  
  
  em {  
    color: $gray;  
  }  
  
  &.-disabled {  
    opacity: .4;  
    pointer-events: none;  
  }  
}
```

1.4. Stylowanie po elementach HTML

Specyficzne przypadki, gdy stylowanie po elemencie jest nieuniknione*:

m. in. includowanie do HTML-a grafik SVG czy elementy bez klas “wyplute” z CMS-a.

- zła praktyka:

```
24
25
26     svg {
27       width: 18px;
28       height: 18px;
29       path {
30         fill: $secondaryColor;
31       }
32     }
```

* lub nieproporcjonalnie czasochłonne

- lepsza praktyka:

```
.svg {
  svg {
    display: block;
    width: 100%;
    height: 100%;
    fill: currentColor;
  }
}
```

```
.button
  &__icon {
    display: inline-block;
    vertical-align: middle;
    width: 1em;
    height: 1em;
  }
```

```
<a class="button" href="#">
  Dowiedz się więcej
  <span class="button__icon svg">{% include '@svg/arrowRight.svg' %}</span>
</a>
```

<https://www.smashingmagazine.com/2016/06/battling-bem-extended-edition-common-problems-and-how-to-avoid-them/#comment-1289706>

1.5. Elementy wewnątrz elementów

Należy pamiętać, że BEM nie jest ściśle powiązany z drzewem DOM.

```
<span
  class="form__radio__extraLabel"
  ng-show="variant.extraRadioLabel.length"
  ng-cloak
>
  {{ '{{ variant.extraRadioLabel }}' }}
</span>
```

Nesting

- *Elements can be nested inside each other.*
- *You can have any number of nesting levels.*
- *An element is always part of a block, not another element. This means that element names can't define a hierarchy such as `block__elem1__elem2`.*

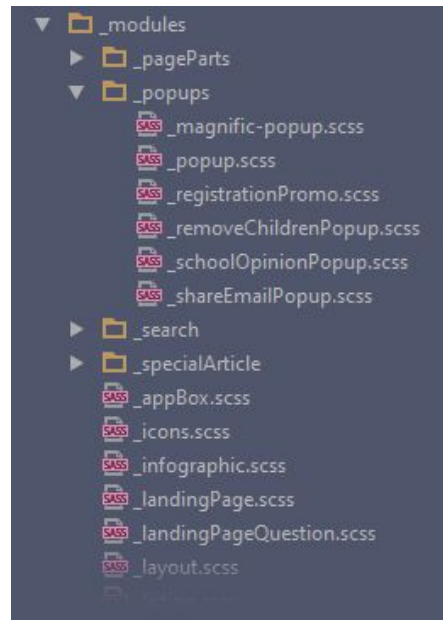
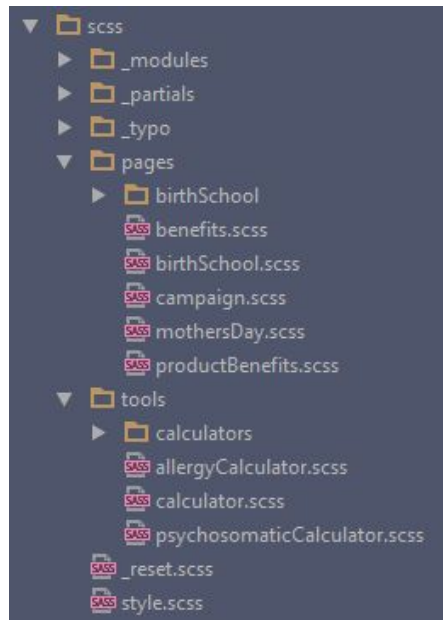
<https://en.bem.info/methodology/quick-start/#nesting-1>

1.6. Błędna struktura i nazwy plików

Tak, metodologia BEM mówi też o tym...

i pomimo, że dopuszcza wiele różnych podejść do struktury katalogów i plików to wszystkie podejścia łączą dwie wspólne cechy:

- hierarchia katalogów i plików musi być **jasno zdefiniowana**
- **nazwa pliku jest taka sama jak nazwa bloku**, który jest wewnątrz (np. blok `.menu` w pliku `_menu.scss`)



<https://en.bem.info/methodology/filestructure/>

2

Czy popsuliśmy BEM-a?
– **nasze modyfikacje**

2.1. Alternatywna konwencja nazewnictwa



No Namespace style

This style is characterized by the absence of a block or element name before a modifier. This scheme limits the use of mixes, because it makes it impossible to determine which block or element a modifier belongs to.



“No namespace style” czyli `block__element -modifier`

<https://en.bem.info/methodology/naming-convention/#no-namespace-style>

2.1. Alternatywna konwencja nazewnictwa

Zalety:

- krótsze nazwy klas
- większa czytelność kodu (?)

```
<a class="button -primary -small">
  click me! <span class="button__icon -large -right">i</span>
</a>

<a class="button button--primary button--small">
  click me! <span class="button__icon button__icon--large button__icon--right">i</span>
</a>
```

<https://en.bem.info/methodology/naming-convention/#no-namespace-style>

2.1. Alternatywna konwencja nazewnictwa

Wady:

- ograniczona możliwość używania mix-ów poprzez **zwiększenie szczegółowości selektorów**

```
.packshot-box
  &__title.title {
    margin: 0;
    padding: 0;
    color: inherit;
    line-height: 1.5;
  }
```

```
.packshot-box
  &__button.button {
    transform: scale(1.5);
    transform-origin: center top;
  }
```

```
<h2 class="packshot-box__title title -1">{{ title | sierotki }}</h2>
```


2.2. Prefiksowanie modyfikatorów oznaczających stan

Do którego i tak się nie stosujemy...



Przejściowe stany, w odróżnieniu od stałych modyfikatorów, oznaczamy klasą `.is-costam` (np. `.is-active`)



Zalety:

- odróżnienie przejściowego stanu od “zwykłego” modyfikatora

Wady:

- brak (?)

<http://confluence.arsthanea.com/pages/viewpage.action?pageId=1933650>

<https://www.smashingmagazine.com/2016/06/battling-bem-extended-edition-common-problems-and-how-to-avoid-them/#6-how-to-handle-states>

INNE

WYSTĘPUJĄCE PROBLEMY Z CSS



3

Dbanie o jakość kodu

3.1. Wyciągnięcie więcej z Sass'a (SCSS-a)

Sass daje nam wiele dobrodziejstw takich jak **funkcje** oraz **mixiny** których **nie wykorzystujemy w pełni**.

- **zła praktyka:**

```
.button
  &.-blue {...}
  &.-green {...}
  &.-gray {...}
  &.-gray.-dark {
    background: $gray-medium;
    color: white;
  }
  &.-white {
    background: white;
    color: $blue;
  }
```

<http://sass-lang.com/>

- **dobra praktyka:**

```
// bg, text, hover bg, hover text
$colors-buttons: (
  "default": ($color-primary-dark, white, $color-primary-dark, $color-secondary),
  "inverted": (white, $color-primary, darken(white, 5%), $color-secondary),
  "disabled": (#dedede, $color-text-light, #dedede, $color-text-light),
);

@mixin button($color-bg, $color-text, $color-hover-bg, $color-hover-text) {
  background: $color-bg;
  color: $color-text;

  &:hover, &:focus {
    background: $color-hover-bg;
    color: $color-hover-text;
  }
}

@each $type, $colors in $colors-buttons {
  @if $type == 'default' {
    @include button(nth($colors, 1), nth($colors, 2), nth($colors, 3), nth($colors, 4));
  } @else {
    // ...
  }
}
```

3.2. Nazywanie klas na podstawie roli, nie wyglądu

- źle

```
&.-blue {  
  background: $blue;  
  color: white;  
  
  &:hover, &:focus {  
    background: lighten($blue, 10%);  
  }  
}
```

- dobrze

```
&.-primary {  
  background: $blue;  
  color: white;  
  
  &:hover, &:focus {  
    background: lighten($blue, 10%);  
  }  
}
```



Good names don't change

*Think about why you want something to look a certain way,
and not really about how it should look.*

***Looks can always change, but the reasons for giving
something a look stay the same.***



↑ Rada od QA W3C z 2004 roku... nadal aktualna.

<https://www.w3.org/QA/Tips/goodclassnames>

3.3. Nazewnictwo zmiennych

Mimo, że BEM nie opisuje nazewnictwa zmiennych to powstało wiele konwencji zarządzania zmiennymi w Sass'ie, które pomagają w ich zarządzaniu i pomagają utrzymać większy porządek w kodzie.

- lepiej:

```
65 $font-size: 14px;  
66 $font-family: 'Intro', Arial, sans-serif;  
67  
68 $color-primary: #006420;  
69 $color-secondary: #43b649;  
70  
71 $alert-colors: (  
72   "success": ($color-success, white),  
73   "error": ($color-error, white),  
74 );
```

<http://thesassway.com/beginner/variable-naming>
<https://danielfurze.co.uk/blog/bem-up-your-variables/>

- bardzo źle:

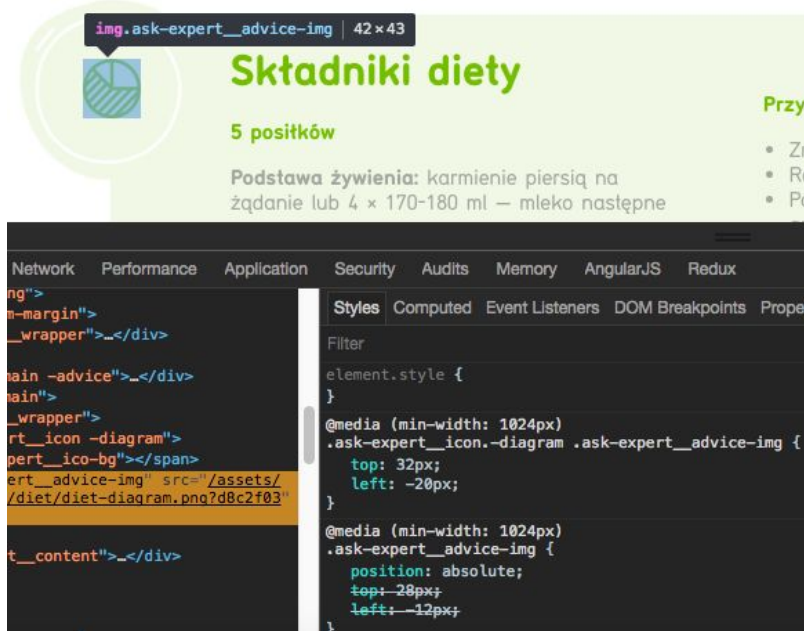
```
18  
19 $gray: #a2a7a0; // fonts  
20 $gray-light: #ebebeb; // borders  
21 $gray-medium: #d3d3d3; // icons  
22 $gray-lighter: #f6f5f1;  
23  
24 $green: #74be00; // fonts, dark backgrounds  
25 $green-medium: #89c826; // borders  
26 $green-second: #5e9a00; // "kontra dla podstawowej zieleni"  
27 $green-light: #f1f8e5; // light backgrounds  
28 $green-dark: #416b00; // dark fonts  
29 $green-dark2: #4e8000;  
30 $green-border: #d7eac6;  
31 $green-border2: #bad1a5;  
32 $green-lime: #d5ebb2;  
33 $brown: #64422f;  
34  
35 $blue: #054ea5; // fonts, button backgrounds  
36 $blue-medium: #106ddd; // hover links  
37 $blue-dark: #004495;  
38 $blue-light: #6fc6f5;  
39  
40 $red: #df4b32;  
41 $red-light: #e79287;  
42 $pink: #f68497;  
43 $orange-light: #f9b991;  
44 $orange: #fd843e;  
45 $white: #fff;  
46
```

3.4. Nadużywanie CSS-a

Wiele rzeczy można napisać znacznie bardziej generycznie - **nie zawsze potrzebny jest modyfikator**, często wystarczy dobry kod.

Dużym problemem jest także, powielanie **niemalże identycznie wyglądających bloków** - problem leży zarówno po stronie designu strony jak i samego wdrożenia.

- źle:



4

**Co więcej
możemy zrobić?**

4.1. Dodanie przestrzeni nazw do bloków BEM-owych

Czyli **BEMIT** (*BEM* + *ITCSS*).

Dodanie przestrzeni nazw eliminuje konflikty nazewnictwa w kodzie oraz poprawia jego czytelność poprzez **określenie roli jaką pełni dany blok**.

```
<div class="o-media c-user c-user—premium">  
  <img class="o-media__img c-user__photo c-avatar" src="" alt="" />  
  <p class="o-media__body c-user__bio">...</p>  
</div>
```

*Adding information onto the beginning and end of the standard Block, Element, or Modifier classes to **give us more knowledge about how the classes behave in a non-relative sense.***

<https://csswizardry.com/2015/03/more-transparent-ui-code-with-namespaces/>

<https://csswizardry.com/2015/08/bemit-taking-the-bem-naming-convention-a-step-further/>

<https://www.youtube.com/watch?v=1OKZOV-iLj4>

<https://www.smashingmagazine.com/2016/06/battling-bem-extended-edition-common-problems-and-how-to-avoid-them/#2-should-i-be-namespacing>

Koniec.

i mam nadzieję, że początek
lepszego jakości kodu ;)

autor:

Marcin Gościcki - <http://goscicki.eu>

