Ćwiczenie 7

1. Cel ćwiczenia

Często zdarza się, że tworzone projekty muszą osiągnąć swoją użyteczność w możliwie najkrótszym terminie. W niniejszym ćwiczeniu zapoznamy się z tego typu technologią.

2. Teoria

Obecnie większość oprogramowania musi powstawać coraz szybciej. Na fakt ten wpływa wiele czynników – coraz większa konkurencja, potrzeba klienta na jak najszybsze wdrożenie rozwiązania (obniżanie kosztów własnych), prezentacja swoich produktów, skuteczna reklama. Ponadto istnieje spora grupa klientów potrzebująca rozbudowy aktualnie posiadanych programów czy portali, których chce dokonać w jak najszybszym możliwym terminie.

Jedną z technologii pozwalających szybko stworzyć gotowy projekt strony jest Bootstrap – gotowy pakiet (framework) zawierający w sobie elementy HTML stylizowane przez pliki CSS (pisane z LASS/SASS) oraz własnymi rozwiązaniami JavaScript (oparte o jQuery). Rozwiązanie tego typu pozwala twórcy strony czy aplikacji HTML szybko zbudować wizualną część projektu bez większego wysiłku; od razu dostępne są ramki, okna, ikony czy tła – wystarczy tylko podać ich nazwy. Rozwiązanie to ma dodatkową zaletę – wszystkie dostępne w nim elementy są automatycznie dostosowywane do różnych rozmiarów ekranów – nie musimy więc ręcznie dopasowywać treści strony np. do urządzeń przenośnych (dopasują się same).

Sam Bootstrap jest bardzo ciekawym rozwiązaniem. Główny nacisk położono w nim na przystosowanie projektów do ekranów urządzeń przenośnych. Zajmuje się wyłącznie obsługą interfejsu użytkownika, sferę logiczną projektu pozostawiając dla innych rozwiązań/samego programisty. Dzięki takiemu podejściu jest on lekki i szybki.

INFORMACJA: Opisywane rozwiązanie może być porównywane (jednak nie jest równoważne) do rozwiązań znanych z języków aplikacji systemowych, takich jak WPF (.NET), QtQuick (Qt) czy SWT/SWING/SwingX (Java). Użytkowanie Bootstrap przy pisaniu dużych projektów (w środowisku kilku/kilkunastu programistów) może znacznie ułatwić wymianę kodu, przepływ informacji oraz przyspieszyć tempo powstawania graficznej części projektu. Oczywiście w przypadku projektów o indywidualnym charakterze Bootstrap będzie mniej efektywny/nieefektywny gdyż każdy z jego elementów trzeba będzie odpowiednio modyfikować (znacznie prostsze może być stworzenie własnych komponentów).

Poniżej zostaną wymienione i krótko scharakteryzowane poszczególne części pakietu Bootstrap.

2.1 Instalacja i konfiguracja

Tak jak z poprzednio opisywane rozwiązania (jQuery), tak i Bootstrap wymaga podłączenia dodatkowych plików celem dodania pożądanej funkcjonalności. Pliki można podłączyć na dwa sposoby:

a) pobrać kod Bootstrap; w tym wypadku mamy możliwość pobrania pliku zip do dołączenia na stronę (wersja minified), z kodem źródłowym (plik zawierający pełne nazwy funkcji i komentarze) bądź przebudowaną wersję Bootstrap na SASS (o tym rozwiązaniu wspomniane zostanie nieco później). Inną możliwością jest pobranie kodu poprzez narzędzie npm (Node.js Package Manager) bądź bower (narzędzie śledzenia wersji pakietów gotowych rozwiązań sieciowych). Oba podejścia wymagają posiadania Node.js. Dla PHP istnieje możliwość instalacji przez composer.
b) wykorzystać serwery CDN – podobnie jak to ma miejsce z jQuery. W tym wypadku wymagane jest jednak dodanie następujących linii:

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap-theme.min.css" integrity="sha384-rHyoN1iRsVXV4nD0JutlnGaslCJuC7uwjduW9SVrLvRYooPp2bWYgmgJQIXwl/Sp" crossorigin="anonymous">

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js" integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa" crossorigin="anonymous"></script>

Z czego drugi odnośnik jest opcjonalny (zawiera motywy rozwiązań, które każdy może tworzyć wedle własnego uznania). Z kolei ostatnia linia wymaga by NAJPIERW DOŁĄCZYĆ kod jQuery (stosowne informacje jak to zrobić można znaleźć w poprzednich materiałach). Przy tym proszę pamiętać, że używanie dodatków JS w Bootstrap również jest nadobowiązkowe (w związku z tym i ostatnia linia może być opcjonalna).

Od tego momentu możemy wykorzystywać Bootstrap na naszej stronie WWW.

2.1 Gotowe komponenty HTML.

Poniżej wymienione zostaną elementy HTML dodane przez bibliotekę. Ponieważ dodanie ich na własną stronę sprowadza się najczęściej do operacji kopiuj/wklej (ewentualnie nadaj własne kolory) niniejszy materiał ograniczy się jedynie do krótkiego opisu każdej grupy, natomiast celem rozwinięcia zagadnienia zostaną udostępnione odnośniki zawierające przykłady z kodem i dodatkowymi informacjami.
Dostępne elementy HTML (stylizowane przez CSS):
- listy rozwijalne – do dyspozycji projektanta oddano kilka wariantów list rozwijalnych. W przeciwieństwie do elementu <select> są one lepiej dopasowane do ogólnie przyjętej grafiki dla aplikacji oraz pozwalają na dodatkowe usprawnienia, jak wyświetlanie rozwijania u góry/dołu czy też dodawanie linii rozdzielających poszczególne opcje (jak w menu okna programów). Przykłady i szczegóły: http://getbootstrap.com/components/#dropdowns
- przyciski – elementy te tworzy się generalnie z dowolnego elementu: warstw(div) bądź przycisku (button). Prócz CSS stylizujących wygląd i dodających dodatkowe motywy graficzne przycisków (wedle ich funkcji – np. potwierdzenia, anulowania, akceptacji) możliwe jest tworzenie grup przycisków. Przyciski objęte klauzulą btn-group łączą się w jeden blok rozdzielony ramkami. Możliwe jest ustawianie przycisków zarówno w poziomie jak i pionie. Dodatkowo istnieje możliwość tworzenia przycisków rozwijalnych (zarówno z opcją szybkiego kliknięcia jak i klasycznej listy rozwijalnej). Przykłady i szczegóły: http://getbootstrap.com/components/#btn-groups, http://getbootstrap.com/components/#btn-dropdowns
- pola wprowadzania – największym usprawnieniem jest dodanie tzw. grupy wprowadzania (input-group). Pozwala ona na lepszą stylizację pól poprzez dodanie np. stałej wprowadzania (przedrostka bądź przyrostka nazwy użytkownika, protokołu strony WWW itp.) w podobny sposób jaki ma miejsce w tradycyjnych aplikacjach systemowych. Przykłady i szczegóły: http://getbootstrap.com/components/#input-groups
- nawigacja – udostępnione są dwa podejścia. Jedno z nich to klasyczne menu, pozwalające na dodawanie przycisków, pól wprowadzania (np. do przeszukiwania opcji), tworzenia przejść

pomiędzy wynikami (lista wyników/stronicowanie) jak też nawigowanie po stronie w formie zakładek (stylizacja przycisków nawigacji). Do wyboru mamy możliwość wyświetlenia elementów w poziomie bądź pionie. Przykładu i szczegóły: http://getbootstrap.com/components/#nav, http://getbootstrap.com/components/#navbar, http://getbootstrap.com/components/#pagination
- ***lokalizacja –*** pozwala na dodanie informacji lokalizacyjnej na przeglądanej stronie WWW (na której podstronie się aktualnie znajdujemy). Przykład: http://getbootstrap.com/components/#breadcrumbs
- ***oznaczenia –*** dodaje dość popularne ostatnimi czasy pole do wskazywania użytkownikowi np. ilość nowych wiadomości, które napłynęły do niego w czasie korzystania z naszej strony/aplikacji. Przykład: http://getbootstrap.com/components/#badges
- ***miniatury –*** pozwala na tworzenie ładnych miniatur dla galerii. Miniatury same dostosowują swój rozmiar do aktualnego aspektu szerokość/wysokość. Istnienie możliwość dodawania opisu do zdjęć oraz pozostałych elementów do galerii (działa na podobnej zasadzie do FlexBox). Przykład: http://getbootstrap.com/components/#thumbnails
- ***elementy ramkowe –*** pozwalają na nadanie ładnego wyglądu dla prezentowanej treści. Jednym z takich elementów jest nagłówek, pozwalający na ładniejszą prezentację tytułów stron. Element 'well' pozwala na nadanie ładnego wyglądu elementom poprzez zaokrągloną ramkę i delikatny odcień szarości. Z kolei powiadomienia można wypisywać w specjalnie przygotowanych do tego elementach alerts (wyglądających jak element well z dodatkowym kolorem). Ostatnim tego typu elementem jest Jumbotron, pozwalający na tworzenie ramki tekstowej w stylu kafelka z nagłówkliem informacyjnym (który może służyć do przejścia na podstronę pełnego artykułu). Przykłady: http://getbootstrap.com/components/#jumbotron, http://getbootstrap.com/components/#page-header, http://getbootstrap.com/components/#wells, http://getbootstrap.com/components/#alerts
- ***etykiety –*** element typu inline pozwalający dodawać szeroko rozumiane etykiety (np. jako hasło opisujące dany temat). Przykład: http://getbootstrap.com/components/#labels
- ***paski postępu –*** gotowe do użycia paski postępu pozwalające użytkownikowi na kontrolę ładowania się danego elementu czy też całej strony (efektowne szczególnie przy stronach asynchronicznych). Przykłady i szczegóły: http://getbootstrap.com/components/#progress
- ***obiekty wyświetlające –*** obiekty pozwalające na wyświetlanie różnego rodzaju informacji (media object). Rozwiązanie szczególnie przydatne przy publikacji artykułów okraszanych zdjęciami, materiałami wideo bądź audio. Przykłady zastosowania: http://getbootstrap.com/components/#media
- ***okienka –*** jeżeli chcemy wyświetlić dodatkowe informacje, np. przez określony czas bądź jako niezależne informacje to możemy wykorzystać elementy okienkowe (panele). Elementy te mogą być także przydatne w przypadku dekorowania wyświetlania rozwijalnych pudełek (rollbox). Przykłady i szczegóły: http://getbootstrap.com/components/#panels
- ***elastyczne wbudowania –*** elementy te mają pozwalać na odpowiednie dynamieczne zmniejszanie i zwiększanie elementów, które dołączane są do strony poprzez zewnętrzne wtyczki bądź jako elementy ramek pływających (iframe). Przykład: http://getbootstrap.com/components/#responsive-embed
- ***listy –*** element grupujący i wyświetlający elementy list jako osobne pozycje. Dzięki zastosowaniu odpowiednich klas możliwe jest utworzenie list na kształt panelu bądź tabeli (w zależności od potrzeb). Przykład: http://getbootstrap.com/components/#list-group
- ***ikony obrazkowe –*** Boostrap dodatkowo posiada 250 ikon obrazkowych. Każdą z nich można wykorzystać w dowolnym momencie – wystarczy wkleić jej kod. Pełny zestaw dostępny pod adresem: http://getbootstrap.com/components/#glyphicons

Wykorzystując powyższe elementy należy pamiętać by umieścić w kodzie dokumentu znacznik meta informujący przeglądarki o kierunku skalowania oraz proporcji początkowej:

<meta name="viewport" content="width=device-width, initial-scale=1">

Bez powyższej linii przeglądarki na urządzeniach przenośnych nie zawsze mogą radzić sobie z powiększaniem/pomniejszaniem zawartości strony.

Poszczególne klasy CSS mają niekiedy przyrostki -lg, -md, -sm, czy -xs. Oznaczają one rozmiar danego elementu (od największego do najmniejszego).

Niektóre elementy mają swoje dodatkowe możliwości, np. z paneli można tworzyć okna modalne (użytkownik musi zamknąć je poprzez kliknięcie w przycisk – blokują całą stronę), przyklejać pasek nawigacji (tzw. Affix) czy też porządkować elementy HTML wedle siatki (GridLayout). Więcej informacji o poszczególnych elementach Bootstrap można znaleźć w odnośnikach do materiałów.

2.2 Użytkowanie CSS.

Ogólnie Bootstrap umożliwia użytkowanie natywnej implementacji CSS. Jednak przystosowany został do wykorzystywania dwóch wspomagających rozwiązań – LESS oraz, w nowszej wersji, SASS (SCSS).

Obie technologie wprowadzają wiele udogodnień do pierwotnej implementacji CSS. Przede wszystkim umożliwiają użytkowanie zmiennych (dostępne częściowo także w czystym CSS). Drugą ważną rzeczą jest wykorzystywanie pełnej implementacji warunków (CSS posiada jedynie ograniczoną funkcjonalność warunkową). Obie implementacje pozwalają także na tworzenie własnych funkcji CSS (brak implementacji w CSS). Nowością jest także możliwość zagnieżdżania definicji CSS w istniejącej (czytelniejsza forma dziedziczenia poprzez operatory).

Kolejną istotną rzeczą łączącą oba rozwiązania jest konieczność ich ‘kompilacji’. Kompilacją określa się tutaj proces zamiany utworzonych plików less/sass w pliki css – przeglądarki nie potrafią bowiem interpretować składni omawianych technologii.

Na tym podobieństwa pomiędzy omawianymi technologiami się kończą. Poniżej każda z nich zostanie krótko scharakteryzowana oraz przedstawione zostaną ich możliwości (poprzez przykłady kodu wraz z omówieniem).

2.2.1 LESS

Język LESS powstał w 2009. Pierwotnie napisana w Ruby do wykorzystania w framework Ruby on Rails. Wraz ze wzrostem popularności przepisana została do języka JavaScript. Pozwoliło to na umieszczenie parsera zarówno po stronie coraz popularniejszego serwera node.js jak i po stronie klienta – rozwiązanie to wymaga jednak dodania dodatkowej biblioteki do dokumentu strony WWW. Swój parser posiada również PHP.

Sama składnia podobna jest do oryginalnego CSS. Różnicą jest posiadanie wyżej wymienionych dodatków językowych, tworzących z CSS prawdziwy język programowania.

a) zmienne

LESS pozwala na tworzenie zmiennych. Dzięki nim poszczególne wartości można ze sobą łączyć czy porównywać, tworząc ściśle dopasowane elementy HTML.

Przykład wykorzystania zmiennych:

```
// Import biblioteki - innego pliku z kodem less
@import "library.less";
// Zmienne
@selektor: banner;    //nazwa selektora, która zostanie podstawiona poniżej
```

```
@images: "../img";   //zmienna tekstowa pozwalający na wstawienie się w dowolny fragment kodu poniżej
@kolor: green;         //nazwa koloru
@kolor2: darken(@kolor, 20%); //zmiana wartości koloru ze zmiennej (przyciemnienie o 10%)

@tekst: "Tekst ze zmiennej występujący przed banerem"; //zmienna tekstowa
@tekstZmiennej: "tekst"; //zmienna tekstowa, której wartość wykorzystana zostanie do wstawienia zawartości
                         //zmiennej o takiej nazwie
.@{selektor} {
  font-weight: bold;
  line-height: 40px;
  margin: 0 auto;
  color: @kolor;
  background: url("@{images}/white-sand.png");
}

.@{selektor}::before {
  content: @@tekstZmiennej;
}

body {
  color: @kolor2;
}
```

Wynik w CSS (po kompilacji):

```
.banner {
  font-weight: bold;
  line-height: 40px;
  margin: 0 auto;
  color: green;
  background: url("../img/white-sand.png");
}

.banner::before {
        content: "Tekst ze zmiennej występujący przed banerem";
}

body {
  color: #4d8a0f;
}
```

b) Rozszerzenia (dziedziczenie) – pozwalają dziedziczenie przez wskazane selektory właściwości z innych selektorów. Rozszerzenia mają postać pseudo-klasy CSS. Pozwalają na przypisywanie ich na kilka sposobów:

.a:extend(.b) {} - przypisanie wartości znajdujących się w klasie .b do klasy .a; jeżeli klasy będą zawierać te same właściwości (np. color) to końcowa wartość tego parametru w klasie .a będzie zależeć od położenia klasy .b (rozszerzenie przypisuje się do położenia klasy, po której dziedziczymy właściwości – przykład kodu poniżej wyjaśnień)

```
.a {
        &:extend(.b);
}
```
działanie powyższego kodu jest tożsame z działaniem poprzedniego przykładu. Różnica polega jedynie na stylu zapisu (dla niektórych czytelniejszy). Elementem spajającym jest znak &, który jest wskaźnikiem na obecnie tworzoną klasę (działa tak samo jak np. pseudozmienna this w JavaScript)

.a:extend(.b,.c) {} - pozwala na dziedziczenie z klas podanych jako parametry; dziedziczenie będzie odbywać się, tak jak w poprzednich przypadkach, wedle kolejności dodania selektorów klas w kodzie less

a:extend(tr .header) {} - dziedziczenie odbędzie się w przypadku wykorzystania w kodzie less konstrukcji dziedziczenia selektorów CSS (znacznika wiersza tabeli, dla którego przypisana zostanie klasa header)

a:extend(.header all) {} - ten typ rozszerzenia pozwala na odziedziczenie właściwości ze wszystkich klas, których konstrukcja będzie rozpoczynać się od nazwy header (.header.c, .header.row itp.)

```
@media screen and (max-width: 1000px) {
        .a:extend(.b) {}

        .b {color: green;}
}

@media screen and (max-width: 500px) {
        .b { color: yellow;}
}

.b { color: red;}
```

Powyższy przykład zawiera kod rozszerzenia i obrazując jego zasięg; klasa a przyjmie wartość koloru zielonego dla zawartości docelowego elementu. Jednak działanie rozszerzenia obejmuje jedynie **pierwszą klauzulę media**. Druga klauzula oraz klasa spoza zakresu @media nie będą miały wpływu na klasę z pierwszego @media (przykład kodu na końcu opisu).

Niestety rozszerzenia nie działają poprawnie z dopasowaniami, pseudoselektorami i pseudoklasami. Nie działa również łączenie po zmiennych less. Extends nie posiadają również mechanizmu wykrywającego wielokrotne deklaracje (zdublowane).

Przykład kodu LESS:

```
.tag {
        &:extend(.panel);
        width: 50%;
        height: 200vh;
        background: url('./img/cloud.jpg');
}

.panel {
        height: 100px;
```

```
        width: 100%;
}

@media screen and (max-width: 1000px) {
        .a:extend(.b) {}

        .b {color: green;}
}

@media screen and (max-width: 500px) {
        .b { color: yellow;}
}

.b { color: red;}
```

Uzyskany kod CSS z powyższego przykładu:

```
.tag {
        width: 50%;
        height: 200vh;
        background: url('./img/cloud.jpg');
}

.tag,
.panel {
        height: 100px;
        width: 100%;
}

@media screen and (max-width: 1000px) {
        .a,
        .b {
                color: green;
        }
}

@media screen and (max-width: 500px {
        .b {
                color: yellow;
        }
}

.b {
        color: red;
}
```

Jak można zauważyć, klasa tag początkowo jest w całości przeniesiona do kodu CSS (za wyjątkiem linii extend). Z kolei przy deklaracji klasy panel po przecinku również wymieniona została klada tag. Tym samym przeglądarka NADPISZE wartości tag o nowe wartości, które pierwotnie zarezerwowane były dla klasy panel.
Podobnie jest w przypadku przytaczanego elementu @media. W pierwszym z nich klasa a jest zadeklarowana w tym samym miejscu co klasa b (klasa a nie posiadała żadnej deklaracji

właściwość → wartość). W drugim bloku @media klasa b nie wpływa na zawartość a – należą do innych przedziałów. To samo ma miejsce na końcu kodu CSS – klasa b nie wpływa na zawartość a w przestrzeni media.

c) Mixins (wartości łączone) – cecha ta pozwala na budowanie selektorów poprzez nadawanie im właściwości innych selektorów; nadawanie właściwości odbywa się poprzez podanie nazwy innego selektora w ciele selektora docelowego. Łączenia pozwalają ponadto na przekazywanie do nich parametrów (zachowują się jak funkcje) – zarówno ściśle określoną ilość parametrów jak i dowolną (zaznaczaną jako wielokropek). Ponadto łączenia mogą pełnić rolę przestrzeni nazw.

Przykład kodu:

```
.bielZielen() {
        color: white;
        background: green;
}

#bielCzerwien {
        color: white;
        background: red;
}

#krojePisma {
  //ta funkcja bedzie ukryta - posiada przy deklaracji nawias
        .pochyle(@wielkosc: 12px, @grubosc: normal) {
                font-height: @wielkosc;
                font-width: @grubosc;
                color: black;
        }
//ta bedzie jawnie deklarowana w CSS
        .pogrubione {
                font-weight: bold;
        }

        .duzeLitery(@wielkosc: 10px; @kolor: black) {
      font-size: @wielkosc;
                text-transform: uppercase;
                color: @kolor;
        }
}
//niejawnie deklarowana - posiada nawias
.ramka(@x: 0; @y: 0; @sz: 200px; @w: 100px; @tlo...) {
        position: absolute;
        left: @x;
        top: @y;
        width: @sz;
        height: @w;
        background: @tlo;
}

.funkcjaNazwana() {
        @back: yellow url('./obrazek.png') no-repeat center fixed;
```

```
}

@funkcjaAnonimowa: {
        font-family: Arial, serif;
        font-variant: small-caps;
};

//tutaj zaczyna sie wlasciwa czesc CSS, mozna
//rozdzielic ten kod na dwa pliki!

body {
        //nawias wymagany - inaczej nastapi blad kompilatora
   @funkcjaAnonimowa();
        .bielZielen; // <==> .bielZielen();
}

.okno {
        .ramka(30px;0; 200px; 300px; white; fixed);
}

.okno p {
        #krojePisma.pogrubione;
}

h2 {
  #krojePisma.duzeLitery(20px; black);
}
```

Kod po parsowaniu do CSS:

```
#bielCzerwien {
  color: white;
  background: red;
}
#krojePisma .pogrubione {
  font-weight: bold;
}
body {
  font-family: Arial, serif;
  font-variant: small-caps;
  color: white;
  background: green;
}
.okno {
  position: absolute;
  left: 30px;
  top: 0;
  width: 200px;
  height: 300px;
  background: white fixed;
}
.okno p {
```

```
  font-weight: bold;
}
h2 {
  font-size: 20px;
  text-transform: uppercase;
  color: black;
}
```

d) import plików – rozszerza oryginalną dyrektywę importu o możliwości dołączania niejawnie treści plików; dostępne dyrektywy:

- reference: użyj pliku LESS lecz nie wyświetlaj go
- inline: umieść plik źródłowy w wyjściowym lecz nie przetwarzaj go
- less: traktuj plik jako LESS (bez względu na rozszerzenie)
- css: traktuj plik jako CSS (bez względu na rozszerzenie)
- once: dołącz wskazany plik tylko jeden raz (domyślnie)
- multiple: dołączaj plik wielokrotnie (przy każdym imporcie)
- optional: plik nie jest wymagany przy kompilacji (jego brak nie spowoduje błędu)

Przykład użycia:

```
@import "foo.less";
@import "plik.css";
@import (optional, reference) "foo2.less";
@import (reference) "foo3.less";
```

e) strażnicy (instrukcje warunkowe) – LESS wprowadza funkcje warunkowe, nazywane w nim jako strażnicy (guards). Pozwalają one na pozostawienie decyzji parserowi czy dana deklaracja powinna zostać dołączona do pliku docelowego czy nie. Warunki można stosować bezpośrednio do samych deklaracji jak i do zmiennych/funkcji LESS.

Przykład kodu LESS (dla mixins):

```
.mixin (@a) when (lightness(@a) >= 50%) {
  background-color: black;
}
.mixin (@a) when (lightness(@a) < 50%) {
  background-color: white;
}
.mixin (@a) {
  color: @a;
}

body {
  .mixin(#707070);
}

article {
  .mixin(#f0f);
}
```

Przykład wyjścia CSS:

```
body {
  background-color: white;
  color: #707070;
}
article {
  background-color: black;
  color: #f0f;
}
```

Wykorzystując strażników można też stworzyć pętlę (LESS nie posiada jako takich pętli). Do chwili, kiedy wskazany warunek będzie zwracał prawdę, będzie generowany kod CSS. Przykład kodu LESS:

```
.generate-rows(4);

.generate-rows(@n, @i: 1) when (@i =< @n) {
  .row-@{i} {
    width: (@i * 100% / @n);
  }
  .generate-rows(@n, (@i + 1));
}
```

Wynik w kodzie CSS:

```
.row-1 {
  width: 25%;
}
.row-2 {
  width: 50%;
}
.row-3 {
  width: 75%;
}
.row-4 {
  width: 100%;
}
```

W dokumentacji LESS można jeszcze znaleźć opis złączeń czy dziedziczenia. Możliwości te jednak pośrednio korzystają z rozwiązań opisanych powyżej (istnieje pełna dokumentacja w języku polskim, z której można dowiedzieć się szczegółów). Celem przetestowania kodu LESS można wykorzystać parser na stronie WWW (adres dostępny w materiałach).

2.2.2 Używanie SASS/SCSS

SASS (Syntactically Awesome Style Sheets – tłum. np. świetne składniowo arkusze stylu) pierwotnie został stworzony w 2006 roku w języku Ruby. Głównym celem implementacji było stworzenie czystych składniowo plików stylów, w których podmiana poszczególnych wartości (np. kolorów, wielkości fontów itp.) nie wiązałaby się z przepisywaniem od nowa plików CSS bądź

byłaby po prostu żmudna (pliki te, w przypadku bardzo rozbudowanych serwisów, mogą mieć po kilka tysięcy linii).

Pierwotnie SASS posiadał bardzo podobną składnię do Ruby (i tym samym Pythona) – definicje wartości poszczególnych selektorów nie są obejmowane w nawiasy klamrowe (decydują o tym wcięcia), tylko jedna właściwość na linię (brak średników) itp. To z kolei decydowało, iż wielu programistów, przyzwyczajonych do języków C/C++, PHP, JavaScript czy Java czuło się z SASS nieswojo i nie korzystało z niego. Mało z tego – spowodowało to powstanie języka LESS, który bardziej przypominał składnię pierwotną CSS. Wszystko to sprawiło, że od 3 wersji SASS otrzymał nowy model składni nazwany SCSS (Sassy CSS). Generalnie działa ona tak samo jak pierwotna składnia SASS, jednak przywrócono w niej nawiasy, możliwość pisania w jednej linii (średniki) oraz wyeliminowano separatory w postaci tabulacji i spacji. Zmiana ta, w połączeniu z bardziej intuicyjnymi rozwiązaniami np. pętli czy instrukcji warunkowych powoduje, że SASS jest znacznie lepszym wyborem niż LESS (efektywność oba parsery posiadają podobną).

UWAGA! W dalszej części materiału zostanie przedstawiona składnia SCSS (chociaż zdania są podzielone to jednak nowa nomenklatura pozwala na większe upakowanie kodu).

SASS, podobnie jak opisany wcześniej LESS, posiada podobne elementy językowe. Poniżej przedstawione zostaną ich przykłady (i ewentualny opis zmian względem LESS):

a) zmienne – zmienne działają tak samo jak w przypadku LESS, jednak rozpoczynają się od znaku dolara (jak w PHP):

Kod w SCSS:

```
$font-stack: 'Times New Roman', Arial;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

Kod w CSS:

```
body {
  font: 100% "Times New Roman", Arial;
  color: #333;
}
```

b) zagnieżdżenia – uproszczenie wizualne składnie oryginalnego CSS co do dziedziczenia właściwości selektorów i zagnieżdżenia ich w kodzie HTML.

Kod w SCSS:

```
nav {
  ul {
    margin: 0;
    padding: 0;
    list-style: none;
  }
```

```
  li { display: inline-block; }

  a {
    display: block;
    padding: 6px 12px;
    text-decoration: none;
  }
}
```

Kod w CSS:

```
nav ul {
  margin: 0;
  padding: 0;
  list-style: none;
}
nav li {
  display: inline-block;
}
nav a {
  display: block;
  padding: 6px 12px;
  text-decoration: none;
}
```

c) dołączanie plików SASS – podobnie jak to miało miejsce w LESS, SASS pozwala na dołączanie plików z kodem CSS/SASS z innych plików. Ważnym elementem nazwy plików, które pragniemy dołączać, jest znak podkreślenia _, który należy dołączyć przed nazwą pliku scss (np. **_fragment.scss**). W pliku, do którego dołączamy plik należy wstawić dyrektywę:

```
@import 'fragment';
```

czyli bez znaku podkreślenia oraz rozszerzenia pliku (ważne by plik był w tym samym katalogu; jeżeli znajduje się on w innym katalogu trzeba poprzedzić dodawaną nazwę stosowną ścieżką!).

d) mieszacze (mixins) – działanie takie jak w LESS, jednak minimalne różnice w składni (przykład poniżej).

Kod w SCSS:

```
@mixin border-radius($radius) {
  -webkit-border-radius: $radius;
     -moz-border-radius: $radius;
      -ms-border-radius: $radius;
          border-radius: $radius;
}

.box { @include border-radius(10px); }

#ramka { @include border-radius(25px); }
```

Kod w CSS:

```
.box {
  -webkit-border-radius: 10px;
  -moz-border-radius: 10px;
  -ms-border-radius: 10px;
  border-radius: 10px;
}

#ramka {
  -webkit-border-radius: 25px;
  -moz-border-radius: 25px;
  -ms-border-radius: 25px;
  border-radius: 25px;
}
```

Proszę zauważyć, że nazwa elementu mixin jest taka sama jak właściwości CSS (nie koliduje z nią). Nazwa może być dowolna (nie wpłynie to na ostateczny kod CSS).

e) roszerzenia – możliwość wykorzystania jednej deklaracji selektora w innym (tak jak w LESS).

Kod w SCSS:

```
.message {
  @extend #commonColor;
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
  border-color: black;
}

.success {
  @extend .message;
  @extend #commonColor;
  border-color: green;
}

.error {
  @extend .message;
  @extend #commonColor;
  border-color: red;
}

.warning {
  @extend .message;
  @extend #commonColor;
  border-color: yellow;
}

#commonColor {
  border-color: orange;
}
```

Kod w CSS:

```
.message, .success, .error, .warning {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
  border-color: black;
}

.success {
  border-color: green;
}

.error {
  border-color: red;
}

.warning {
  border-color: yellow;
}

#commonColor, .message, .success, .error, .warning {
  border-color: orange;
}
```

Proszę zwrócić uwagę na istotną rolę umiejscowienia selektora, który jest dołączany do pozostałych. W przypadku klasy .message kolor ramki zostanie nadpisany przez kolejne selektory podczas czy identyfikator #commonColor nadpisze wszystkie dotychczasowe opcje swoją własną (ramki będą pomarańczowe!).

f) operatory – SASS w pełni obsługuje operatory matematyczne w związku z czym wszystkie wielkości możemy obliczać względem nadanych parametrów.

Przykład kody SCSS:

```
$width: 100%;

.container {
  width: $width;
}


article {
  float: left;
  width: 600px / 960px * $width;
}

aside {
  float: right;
  width: 300px / 960px * $width;
}
```

Kod CSS:

```
.container {
  width: 100%;
}

article {
  float: left;
  width: 62.5%;
}

aside {
  float: right;
  width: 31.25%;
}
```

g) instrukcje warunkowe – SASS posiada o wiele bardziej intuicyjne instrukcje warunkowe niż LESS. Spotkamy tutaj dyrektywę:
- if (warunek, jeżeli_prawda, jeżeli_fałsz) – najprostsza z instrukcji działająca podobnie jak warunek trójstanowy w C++/PHP/JavaScript
- @if(warunek) {/*kod SASS */} - dyrektywa pozwalająca na przypisanie serii właściwości w ramach pojedynczego warunku
- @if (warunek) {/*kod SASS*/} @else if (warunek2) {/*kod SASS*/} @else {/*kod SASS*/} - rozbudowana postać dyrektywy @if, pozwalająca na sekwencyjną eliminację kolejnych warunków.

Przykład kodu w SCSS:

```
$test: 12;
body {
  border-width: if(52 > $test, 2px, 3px);
}

p {
  @if (1 + $test) == 13 {
    border: 1px solid;
    background: yellow;
  }
  @if 5 > 3 {
    border: 2px dotted;
  }
  @if null {
    border: 3px double;
  }
}

$type: 'bird';
p {
  @if $type == ocean {
    color: blue;
  } @else if $type == matador {
    color: red;
  } @else if $type == monster {
```

```
    color: green;
  } @else {
    color: black;
  }
}
```

Kod w CSS:

```
body {
  border-width: 2px;
}

p {
  border: 1px solid;
  background: yellow;
  border: 2px dotted;
}

p {
  color: black;
}
```

h) pętle – SASS, w przeciwieństwie do LESS, w pełni wspiera pętle programowe znane z innych języków:
- @for $zmienna from <wartosc_poczatkowa> through <wartosc_koncowa>{/*kod SASS*/} - wykonuje pętlę zwiększając wartość $zmienna od wartości początkowej do wartości końcowej
- @each $zmienna in <wartosc1, wartosc2, wartosc 3 {/*kod SASS*/} - wykonuje pętlę dla każdej z podanych wartości (odpowiednik foreach w PHP bądź each w JavaScript)
- @each $zm1, $zm2 in (<w1, w2>), (<w21, w22>) {/*kod SASS*/} - wariacja dyrektywy @each pozwalająca na przypisanie wielu zmiennych w ramach pojedynczego wykonania pętli; w tym wypadku zmienne w nawiasie „mapowane są" do zmiennych wymienionych przed operatorem in (proszę pamiętać, że nie jest to prawdziwa mapowanie a jedynie łączenie podzbiorów); kolejne nawiasy będą mapować wartości w kolejnych iteracjach.

INFORMACJA: Oczywiście zmienne w tej postaci można prawdziwie mapować. W tym celu za zmienne podstawia się pełne wyrażenie właściwość: wartość; np.:

```
@each $zm1, $zm2 in (p: red, span: blue) {
        #{$zm1} {
                color: $m2;
        }
}
```

- @while (warunek) {/*kod SASS*/} - ostatnia z pętli pozwalająca wykonywać kod z nawiasu klamrowego do czasu aż warunek będzie spełniony (działa tak jak pętle w innych językach).

Przykład kodu SCSS:

```
@for $i from 0 through 3 {
  .item-#{$i} { width: 2px * $i; }
}
```

```
@for $i from 5 through 0 {
  #przycisk-#{$i} { width: 2em * $i; background: red;}
}

@each $animal in 'puma', 'zolw', ryba {
  .#{$animal}-icon {
    background-image: url('/images/#{$animal}.png');
  }
}

@each $animal, $color, $cursor in (puma, black, default),
                                  (zolw, blue, pointer),
                                  ('ryba', white, move) {
  .#{$animal}-icon {
    background-image: url('/images/#{$animal}.png');
    border: 2px solid $color;
    cursor: $cursor;
  }
}

@each $header, $size in (h1: 2em, h2: 1.5em, h3: 1.2em) {
  #{$header} {
    font-size: $size;
  }
}

$i: 6;
@while $i > 0 {
  .ramka-#{$i} { width: 2em * $i; }
  $i: $i - 2;
}
```

Przykład w kodzie CSS:

```
.item-0 {
  width: 0px;
}

.item-1 {
  width: 2px;
}

.item-2 {
  width: 4px;
}

.item-3 {
  width: 6px;
}

#przycisk-5 {
  width: 10em;
```

```
  background: red;
}

#przycisk-4 {
  width: 8em;
  background: red;
}

#przycisk-3 {
  width: 6em;
  background: red;
}

#przycisk-2 {
  width: 4em;
  background: red;
}

#przycisk-1 {
  width: 2em;
  background: red;
}

#przycisk-0 {
  width: 0em;
  background: red;
}

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

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

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

.puma-icon {
  background-image: url("/images/puma.png");
  border: 2px solid black;
  cursor: default;
}

.zolw-icon {
  background-image: url("/images/zolw.png");
  border: 2px solid blue;
  cursor: pointer;
}
```

```
.ryba-icon {
  background-image: url("/images/ryba.png");
  border: 2px solid white;
  cursor: move;
}

h1 {
  font-size: 2em;
}

h2 {
  font-size: 1.5em;
}

h3 {
  font-size: 1.2em;
}

.ramka-6 {
  width: 12em;
}

.ramka-4 {
  width: 8em;
}

.ramka-2 {
  width: 4em;
}
```

Oczywiście SASS posiada jeszcze więcej możliwości (jak chociażby SassScripts czy dyrektywy kontrolne). W celu pełnego zapoznania się z możliwościami tego języka warto zapoznać się z jego dokumentacją dostępną bezpośrednio pod adresem projektu.

3. Zadanie do realizacji.

Przy wykorzystaniu opisanej technologii Bootstrap oraz SASS bądź LESS stworzyć panel administracyjny do tworzonego obecnie projektu strony. Można również wykorzystać niektóre części opisywanych technologii w już obecnie stworzonym projekcie, o ile zaistnieje taka możliwość bądź potrzeba. W przypadku braku kompilatorów LESS/SASS można wykorzystać parsery na stronach WWW (dostępne w materiałach).

MATERIAŁY:

http://getbootstrap.com
https://www.bootstrapcdn.com
https://css-tricks.com/sass-vs-less/
http://lesscss.org
http://sass-lang.com
http://marksheet.io/sass-scss-less.html
https://www.w3schools.com/bootstrap/default.asp
https://bower.io/docs/about/
https://www.npmjs.com
http://less2css.org
http://thesassway.com/editorial/sass-vs-scss-which-syntax-is-better
https://www.sassmeister.com
http://sass-lang.com/documentation/file.SASS_REFERENCE.html