Fundusze Europejskie
Wiedza Edukacja Rozwój

Politechnika Wrocławska

Unia Europejska
Europejski Fundusz Społeczny

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Politechnika Wrocławska

Wydział Informatyki i Zarządzania

Zaawansowane Technologie Webowe

Laboratorium

Temat:	SASS oraz SCSS, preprocesory CSS
Opracował:	mgr inż. Piotr Jóźwiak
Data:	lipiec 2020
Liczba godzin:	2 godziny

Table of Contents

Wstęp

Dzięki CSS możemy zaprojektować wygląd strony internetowej dokładnie tak, jak tego chcemy. Jest to bardzo prosty język, który identyfikuje pożądane elementy HTML i odpowiednio je zmienia. W porównaniu do bezpośredniego stylowania w HTML, CSS jest świetny i przystępny. Pozwala oddzielić część wizualną od danych. Jednakże język arkusza stylów CSS ma swoje poważne ograniczenia z punktu widzenia prowadzenia dewelopmentu. Wielu projektantów stron internetowych poszukuje wygodniejszego oraz wydajniejszego sposobu pracy nad implementowaniem arkuszy styli. Odpowiedzią na to zapotrzebowanie są preprocesory CSS. Za pomocą języka arkusza stylów, który stanowi rozszerzenie podstawowego CSS, prace projektowe można znacznie ułatwić.

Korzystając z preprocesorów, programiści mogą uzyskać dostęp do szeregu funkcji, które mógłby posiadać CSS, gdyby nie był to tak prosty język: funkcje, zmienne, mixiny i inne. Preprocesory oszczędzają również znaczną ilość czasu, umożliwiając ponowne użycie predefiniowanych właściwości, zamiast zapisywania ich w wielu kopiach.

Przyjrzyjmy się głównej idei pomiędzy czystym CSS, a definicją dla preprocesora:

```css
/* WITHOUT A PREPROCESSOR */
.header-large {
   font-family:Tahoma;
   font-weight:bold;
   font-size:48px;
   font-variant:small-caps;
   text-decoration:underline;
   color:#ff0000;
}
.header-medium {
   font-family:Tahoma;
   font-weight:bold;
   font-size:36px;
   font-variant:small-caps;
   text-decoration:underline;
   color:#ff0000;
}
.header-small {
   font-family:Tahoma;
   font-weight:bold;
   font-size:18px;
   font-variant:small-caps;
   text-decoration:underline;
   color:#ff0000;
}
```

```css
/* WITH A PREPROCESSOR */
.header-large {
   font-family:Tahoma;
   font-weight:bold;
   font-size:48px;
   font-variant:small-caps;
   text-decoration:underline;
   color:#ff0000;
}
.header-medium {
   .large-heading;
   font-size:36px;
}
.header-small {
   .large-heading;
   font-size:18px;
}
```

W powyższym przykładzie jasno widać, że zwiększa się czytelność definicji, ponieważ wyeliminowaliśmy zbędne powtórzenia. To tylko mały przykład możliwości preprocesora CSS. Więcej tajników zostanie przedstawione w dzisiejszym laboratorium.

Najbardziej znanym z tych języków rozszerzeń jest SASS. Czym dokładnie jest „**Syntactically Awesome Style Sheet**”?

Czym jest SASS?

Aby zrozumieć, czym jest SASS i dlaczego go potrzebujemy, musimy najpierw zrozumieć CSS. Witryny internetowe są oparte o język HTML. Jednakże HTML nie dostarcza dobrych rozwiązań związanych z prezentacją danych. Kaskadowe arkusze stylów (CSS) są rozwinięciem stron HTML dostarczającym warstwę definicji wyglądu. CSS znajduje się na wierzchu kodu HTML, podobnie jak szablon, i określa projekt każdego elementu na stronie: czcionkę, kolor czcionki, tło. Projektanci stron internetowych mogą ustawić wszystkie te elementy projektu za pomocą CSS.

Na przykład, aby ustawić wszystkie nagłówki HTML na 22pt z czcionką Calibri, definiujemy to w arkuszu stylów, w którym podane są wszystkie wytyczne dotyczące wyglądu witryny. Reguły te są zwykle w osobnym pliku, do którego następnie odwołują się strony HTML. To ogromnie skraca ilość informacji na stronie HTML oraz pozwala zmniejszyć transfer poprzez zapamiętanie w pamięci podręcznej przeglądarki raz ściągniętych definicji styli.

Ale CSS ma swoje poważne ograniczenia, co jest szczególnie wyraźne, gdy pracujemy z tym językiem od wielu lat. Zaleta CSS jest jednocześnie jego wadą: język jest zaprojektowany bardzo prosto. SASS sprawia, że wszystko jest nieco bardziej wyrafinowane i znacznie upraszcza pracę nad stworzeniem projektu.

Niestety nie wszystkie przeglądarki potrafią interpretować definicje SASS tak, jak potrafią dziś to zrobić z definicjami CSS. Młodsi deweloperzy stron internetowych mogą już tego nie pamiętać, ale początki CSS wyglądały także bardzo trudno. Każda przeglądarka wprowadzała pewne różnice w rozumieniu definicji arkuszy stylów. Dziś ten problem jest raczej marginalny i CSS w każdej przeglądarce działa bez zarzutów. Jednakże, pomimo że SASS jest już od 10 lat, to jednak nie doczekał się on takiego ujednolicenia. Dlatego dziś wykorzystuje się go jako preprocesor, który generuje na podstawie swoich definicji pliki CSS.

Język używany do implementacji SASS nazywany jest **SassScript**. Preprocesor został zaprojektowany na bazie języka **YAML**. Dlatego składnia SASS w dużej mierze różni się od CSS.

Dlaczego korzystać z SASS?

Gdy rozpoczyna się pracę nad projektowaniem stron internetowych to wydaje się, że CSS może być wystarczający. W końcu czysty CSS umożliwia tworzenie bardzo atrakcyjnych stron internetowych. Dlatego chcąc używać SASS nie można pominąć wykorzystania CSS. Staroświecki język arkuszy stylów będzie nadal pojawiał się w przyszłości jako podstawa prezentacji. SASS i inne języki bazują na szkielecie CSS.

Jednakże włączając SASS otrzymujemy wiele dodatkowych funkcjonalności niedostępnych w CSS:

- **Variables**: Za pomocą SASS możemy zapisać informacje w zmiennych, aby użyć ich później. Możliwe jest na przykład centralne przechowywanie wartości koloru pod zmienną.

- **Funkcje matematyczne**: W SASS można również używać operacji matematycznych, takich jak +, - , *, / lub%. Pozwala to na przykład wpływać na specyfikacje rozmiaru w sposób bardzo dynamiczny.
- **Funkcje**: ułatwiają pracę nad projektem. Pozwalają one między innymi modyfikować wartości kolorów lub analizować listy.
- **Pętle**: Kolejną zaletą SASS jest możliwość konfigurowania pętli. Są one powtarzane, dopóki nie osiągną określonego stanu.
- **Mixins**: najprościej mówiąc, to szablony. Możemy stworzyć je samodzielnie lub po prostu zintegrować z własnym kodem podczas korzystania z frameworku.
- **Wcięcia**: oryginalny SASS (w przeciwieństwie do SCSS) współpracuje z wcięciami i podziałami linii w celu ustrukturyzowania kodu. Nie potrzebujemy nawiasów, aby wyświetlić zagnieżdżenie lub średniki do oznaczenia końca linii.
- **Zagnieżdżanie**: CSS nie pozwala na pracę z zagnieżdżaniem w kodzie. SASS daje jednak użytkownikom możliwość wizualnego przedstawienia zależności w celu ograniczenia nadmiarowości i uproszczenia procesu pisania.
- **Dziedziczenie**: Możliwe jest dziedziczenie właściwości z jednego selektora na drugi. Oszczędza to pisania kodu i sprawia, że kod jest czytelniejszy.
- **Pliki częściowe**: Definicje można podzielić na większą ilość mniejszych plików, które włącza się odpowiednimi poleceniami.,

SCSS vs. SASS

SASS to nie jest jedna składnia, ale dwie. SASS jako oryginalna forma nazywany jest inaczej *„indented syntax”* w związku z tym, że używa YAMLa jako sposobu definiowania reguł. Ale istnieje kolejny wariant, który jest bardziej zorientowany na składnię CSS i nazywany jest przez to **sassy CSS (SCSS)**. Jakby tego było mało, to składnia ta pojawia się w samej definicji CSS w wersji 3. Dlatego można założyć, że za jakiś czas przeglądarki będą interpretowały poprawnie SCSS bez potrzeby jego wcześniejszej kompilacji. Powstaje pytanie – po co dwie składnie na to samo? Zanim odpowiemy na to pytanie to przyjrzyjmy się głębiej różnicom. Wtedy odpowiedź nasunie się sama.

Oryginalna składnia SASS działa w oparciu o wcięcia i podziały linii, jako że wykorzystuje proces zaadaptowany z YAML. Aby zakończyć wiersz kodu, wystarczy wstawić podział wiersza. Wcięcia działają po prostu za pomocą przycisku tabulacji. Grupy, zwane blokami deklaracji, są tworzone przez zmianę pozycji w kroju pisma. Nie jest to możliwe w przypadku samego CSS. W tym przypadku należy użyć nawiasów dla grupowania i średników dla deklaracji właściwości. To jest dokładnie to, co jest potrzebne do SCSS. Powoduje to tyle, że kod jest zwięźlejszy oraz łatwiejszy do kopiowania i wklejania. Nie musimy dbać o pary nawiasów, etc. Z drugiej strony wielu użytkowników narzekało, że trzeba się nauczyć dwóch różnych składni do wykonania jednego zadania. Dlatego powstał drugi wariant zwany SCSS.

Podstawową zaletą SCSS jest fakt, że jest on nadzbiorem CSS. Innymi słowy, każdy plik CSS jest także SCSS – ale nie na odwrót. To już dużo. Jeśli deweloper pozna CSS to przy nauce SCSS rozwija tylko te elementy, które wprowadzają nową funkcjonalność do arkuszy styli.

Teraz już widać odpowiedź na wcześniej postawione pytanie. Nowa składnia ma zbliżyć preprocesor do jego wynikowego pliku CSS, aby ułatwić process uczenia się. Jest jeszcze jedna zaleta. Gdy deweloper

podejmuje decyzję, że chce zacząć używać SCSS, to wystarczy zmienić rozszerzenie plików CSS na SCSS i poddać je preprocesingowi. Sama zmiana będzie przyrostowa. Nie jest to możliwe przy składni SASS.

Instalacja preprocesora SASS

Pora na przykłady w praktyce. Zanim zaczniemy, musimy zainstalować niezbędne oprogramowanie. Przedstawię poniżej jak zainstalować SASS zarówno w systemie Linux jak i Windows.

Instalacja w systemie Linux

W systemie Linux najłatwiej jest zainstalować SASS z wykorzystaniem managera pakietów **npm**. Można to zrobić poleceniem:

```
npm install -g sass
```

To rozwiązanie ma swoje wady. Niestety takie podejście zainstaluje silnik SASS w czystym JavaScript, który jest wolniejszy od innych dostępnych kompilacji. Zachowuje jednak pełną funkcjonalność.

Inną metodą jest instalacja **Ruby**, a następnie instalacja SASS z wykorzystaniem managera **gem**. Można to zrobić w poniższy sposób:

```
apt-get install ruby-full build-essential rubygems
gem install sass
```

Instalacja w systemie Windows

W systemie Windows najlepiej skorzystać z managera pakietów **chocolatey**. Nie jest on oczywiście dostępny w każdym systemie z Redmont, jednakże zachęcam do rozpoczęcia pracy z jego wykorzystaniem. Znacząco ułatwia on prace z instalatorami – na wzór managera znanego z Debiana -**aptitude**. Więcej o managerze można przeczytać tutaj: https://chocolatey.org/.

Jeśli **choco** jest już zainstalowane w naszym systemie, to instalacja SASS będzie równie łatwa jak w przypadku systemu Linux. Wystarczy wykonać poniższe polecenie:

```
choco install sass
```

Jeśli natomiast z jakiś przyczyn nie chcemy instalować managera w systemie, to będziemy zmuszeni zainstalować **Ruby**, a następnie wykorzystać instalatora **gem** tak jak w przykładzie z instalacją w Linux.

Więcej o instalowaniu SASS można przeczytać tutaj: https://sass-lang.com/install

Weryfikacja instalacji SASS

Aby zweryfikować działanie preprocesora najłatwiej jest wykonać polecenie wyświetlające jego wersję. Służy do tego polecenie:

```
sass --version
1.26.10 compiled with dart2js 2.8.4
```

Jeśli w wyniku otrzymaliśmy numer zainstalowanej wersji SASS, to znaczy, że jesteśmy gotowi do pracy z preprocesorem.

Uruchomienie preprocesora

Oprogramowanie mamy już gotowe. Zatem przyszedł czas na kilka przykładów. Zanim je sobie omówimy, musimy nauczyć się korzystać z preprocesora.

Najłatwiej jest rozpocząć pracę poprzez zdefiniowanie dwóch folderów:

- css
- scss

Jak sama nazwa wskazuje, pierwszy z nich będzie służył do przechowywani plików CSS. To do tego folderu preprocesor będzie generował pliki styli. Drugi folder jest kontenerem na nasze pliki źródłowe.

W naszym przykładzie posłużymy się plikami SCSS, które posiadają rozszerzenie **`.scss`**. Zanim jednak napiszemy pierwszy plik to uruchomimy preprocesor do pracy w trybie ciągłym. Oznacza to, że będzie on monitorował folder wejściowy na pojawiające się tam modyfikacje i jeśli jakaś nastąpi to automatycznie skompiluje pliki **`scss`** do plików **`css`**.

W tym celu wykonujemy polecenie:

```
sass --watch scss:css
```

Komunikat "*Sass is watching for changes. Press Ctrl-C to stop.*" Informuje nas, że preprocesor działa i nasłuchuje zmiany. Pliki wynikowe będą pojawiały się automatycznie w folderze **`css`**. Pora na przykłady.

Podstawy SASS

Przejdźmy zatem do omówienia podstawowych możliwości SASS. W naszych przykładach będziemy posługiwać się składnią **SCSS**, gdyż jest ona łatwiejsza do porównania z CSS – od razu widać co się zmieniło. Nic nie stoi na przeszkodzie, aby przykłady napisać w SASS.

Zmienne

Pojęcia zmiennej nie trzeba nikomu tłumaczyć. Chyba jest to najbardziej brakujący element w CSS. Przez brak tego konceptu np. definicja głównego koloru na stronie musiała być zapisana w setkach miejsc. Przy jego zmianie na inną wartość wyśledzenie wszystkich miejsc z jego użyciem było niełatwym zadaniem. SASS wprowadza koncept zmiennych wykorzystując symbol **$** przed nazwą, aby zaznaczyć, że dany element jest definicją zmiennej. Poniższy przykład najlepiej odda ideę:

SCSS

```
$font-stack:    Helvetica, sans-serif;
$primary-color: #333;

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

CSS

```
body {
  font: 100% Helvetica, sans-serif;
  color: #333;
}
```

W momencie przetwarzania, SCSS pobiera zdefiniowane przez nas zmienne dla **$font-stack** i **$primary-color** i generuje standardowy CSS z wartościami zmiennych umieszczonymi w definicjach stylów. Takie rozwiązanie jest niezwykle wydajne podczas pracy np. z kolorami i utrzymaniem ich spójności w całej witrynie.

Zagnieżdżenia

Zagnieżdżenia są bardzo dobrze widoczne w czystym języku HTML. Wynika to z faktu, że HTML jest pewnego rodzaju XMLem. Natomiast CSS pomimo, że odnosi się bezpośrednio do HTML nie posiada takiego konceptu.

SASS umożliwia zagnieżdżanie selektorów CSS w ten sam sposób co hierarchia HTMLa. Sprawia to łatwiejsze oraz bardziej intuicyjne definiowanie reguł odnoszących się do skomplikowanych reguł. Poniżej przykład wykorzystania tej funkcjonalności:

SCSS

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

    li { display: inline-block; }

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

CSS

```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;
}
```

Mixins

Niektóre rzeczy w CSS są żmudne do napisania, szczególnie w CSS3 kontekście prefiksów związanych z przeglądarkami. Mixin pozwala tworzyć grupy deklaracji CSS, które można ponownie wykorzystać w witrynie. Można nawet przekazać wartości, aby uczynić swój mixin bardziej elastycznym. Najlepiej obrazuje to przykład z property transform:

SCSS

```scss
@mixin transform($property) {
    -webkit-transform: $property;
    -ms-transform: $property;
    transform: $property;
  }

.box {
```

CSS

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

```
    @include transform(rotate(30deg));
}
```

Aby zdefiniować mixin należy wykorzystać dyrektywę **@mixin** oraz nadać jej nazwę. W naszym przykładzie mixin został nazwany **transform**. Wykorzystaliśmy także zmienną **$property**, która działa w roli parametru. Po utworzeniu mixin poniżej został przedstawiony sposób użycia do definicji stylu **.box**. Aby wykorzystać mixin należy w miejscu użycia wykorzystać dyrektywę **@include** po czym wskazać nazwę mixina.

Extend/Inheritance

Extend jest jedną z najużyteczniejszych funkcjonalności SASS. Korzystając z wyrażenia **@extend** możemy współdzielić propertisy arkuszy stylów pomiędzy różnymi selektorami. Funkcjonalność ta pozwala utrzymać czysty kod. W swym działaniu przypomina to dziedziczenie klas. Dobrze obrazuje to poniższy przykład, w którym przedstawiam proste selektory do stylowania komunikatów o błędach, ostrzeżeniach i powodzeniu jakiejś akcji.

SCSS

```
/* This CSS will print because
   %message-shared is extended. */
%message-shared {
    border: 1px solid #ccc;
    padding: 10px;
    color: #333;
  }

  /* This CSS won't print because
     %equal-
heights is never extended. */
  %equal-heights {
    display: flex;
    flex-wrap: wrap;
  }

  .message {
    @extend %message-shared;
  }

  .success {
    @extend %message-shared;
    border-color: green;
  }

  .error {
    @extend %message-shared;
    border-color: red;
```

CSS

```
/* This CSS will print because
   %message-shared is extended. */
.warning, .error, .success, .message {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
}

.success {
  border-color: green;
}

.error {
  border-color: red;
}

.warning {
  border-color: yellow;
}
```

```
}

.warning {
  @extend %message-shared;
  border-color: yellow;
}
```

Powyższy kod powoduje, że selektory **.message**, **.success**, **.error** i **.warning** zachowują się tak, jak %message-shared. Oznacza to, że wszędzie tam, gdzie pojawia się **@extend** %message-shared, w danym miejscu zostanie wstawiona definicja z referowanego selektora. Magia dzieje się w wygenerowanym CSS, gdzie każda z tych klas otrzyma te same właściwości CSS, co %message-shared. Pomaga to uniknąć konieczności pisania wielu nazw klas na elementach HTML.

Zauważmy, że selektor %equal-heights nie jest generowany do pliku wynikowego. Dzieje się tak, ponieważ nie został nigdzie użyty – przez co kompilator pominą jego definicję.

Operatory

Czasami wykonanie podstawowych wyrażeń matematycznych jest bardzo pożądane przy definiowaniu styli. Tu z pomocą przychodzą operatory takie jak +,-,*,/ oraz %. Pozwalają one budować proste wyrażenia.

SCSS

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

  article[role="main"] {
    float: left;
    width: 600px / 960px * 100%;
  }

  aside[role="complementary"] {
    float: right;
    width: 300px / 960px * 100%;
  }
```

CSS

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

article[role=main] {
  float: left;
  width: 62.5%;
}

aside[role=complementary] {
  float: right;
  width: 31.25%;
}
```

W powyższym przykładzie utworzyliśmy prosty fluid grid bazując na rozmiarze 960px. Operatory w SASS ułatwiają nam pracę związaną z przeliczaniem wartości pikseli na wartości procentowe.

Funkcje

W SASS odnajdujemy także wiele funkcji wspomagających wykonywanie skomplikowane obliczenia. Ich lista jest dostępna np. tutaj: https://www.w3schools.com/sass/sass_functions_string.asp. Oprócz tego można wykorzystywać wyrażenia sterujące **@if**, **@else** (https://sass-lang.com/documentation/at-

rules/control/if) czy pętle **@for** oraz **@while**. Możliwości jest bardzo dużo, tym więcej, że możemy sami dopisywać swoje funkcje. Przyjrzyjmy się poniższemu przykładowi:

SCSS

```
@function pow($base, $exponent) {
    $result: 1;
    @for $_ from 1 through $exponent {
      $result: $result * $base;
    }
    @return $result;
  }

  .sidebar {
    float: left;
    margin-left: pow(4, 3) * 1px;
  }
```

CSS

```
.sidebar {
  float: left;
  margin-left: 64px;
}
```

Jak widać na powyższym przykładzie zdefiniowaliśmy funkcję **pow**, której zadaniem jest policzenie potęgi. W czystym CSS nie jest to możliwe, jednakże z SASS skomplikowane funkcjonalności mogą zostać opakowane w funkcje i udostępnione jako biblioteki do wykonywania powtarzalnych zadań.

Podsumowanie

Omówiliśmy sobie podstawowe zasady działania wraz z przykładamy preprocesorów CSS. Pełnią one bardzo użyteczne działanie w świecie projektowania stron internetowych. Im więcej zaczniemy używać tej technologii tym bardziej przekonamy się o jej zaletach. Więcej szczegółowych informacji o działaniu SASS można przeczytać w dokumentacji: https://sass-lang.com/documentation/syntax.