Bartosz Chucherko

# Sass

## Nowoczesne arkusze stylów

Helion

Redaktor prowadzący: Małgorzata Kulik

Projekt okładki: Bartosz Chucherko

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
*http://helion.pl/user/opinie/sasspp*
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-283-2619-4



Printed in Poland.

# Spis treści

# Optymalizacja technik Sass

Jeśli uważnie czytałeś poprzednie dwie części książki, zapewne zwróciłeś uwagę, że w wielu miejscach pojawiły się już liczne wskazówki dotyczące najlepszych praktyk. Tak było np. w przypadku zasad stosowania komentarzy, zmiennych, domieszek, rozszerzeń, importowania plików czy kwestii wyboru pętli. W tym rozdziale podsumujemy i uzupełnimy tę wiedzę.

Swego czasu w Internecie pojawiło się wiele dyskusji i krytyki na temat tego, że preprocesory takie jak Sass generują kod słabej jakości, gorszy od arkuszy CSS pisanych „ręcznie". Wraz z upływem czasu, popularyzacją preprocesorów, a co za tym idzie wypracowaniem dobrych praktyk sprawa nieco się rozjaśniła. Niezmienna jednak pozostaje stara prawda — **słaby kod Sass wygeneruje słaby kod CSS**. Nadużywanie i nieumiejętne stosowanie którejkolwiek z omawianych dotąd technik, takiej jak na przykład rozszerzanie, może szybko doprowadzić do powstania rozdętego kodu CSS i problemów ze specyficznością selektorów.

**Celem optymalizacji kodu Sass jest otrzymanie wydajnego, przejrzystego, łatwego w modyfikacji i utrzymaniu kodu, zgodnego z zasadą DRY („bez zbędnych powtórzeń")**. Jeśli piszesz swój kod uważnie od samego początku, prawdopodobnie nie będziesz musiał wiele optymalizować. Często jednak kod pisze się szybko, zostawiając optymalizację na koniec, a w rezultacie nigdy nie ma ona miejsca. Warto więc pamiętać o dobrych praktykach i od początku tworzyć wydajny i schludny kod.

Prawdopodobnie najlepszą, ogólną radą, jaką mogę przekazać wszystkim osobom piszącym w Sass (a szczególnie tym początkującym), jest **zachowanie prostoty i umiaru**. Być może brzmi to banalnie, ale wielu deweloperów próbuje niekiedy „na siłę" stosować wymyślne i skomplikowane techniki kodowania w Sass tylko dlatego, że są one dostępne, tracąc przy tym czas i niepotrzebnie komplikując kod. Patrząc na kod, powinieneś od razy być w stanie zrozumieć jego działanie i przewidzieć, co zostanie wygenerowane do postaci CSS. **Pamiętaj, że celem Sass jako narzędzia jest umożliwienie twórcom arkuszy stylów pisania czystszego, bardziej czytelnego i łatwiejszego w utrzymaniu oraz modyfikacji kodu CSS**. Nieumiejętne i zbyt częste wykorzystywanie elementów składni Sass, takich jak domieszki, rozszerzenia czy zagnieżdżanie, może doprowadzić do zbędnego skomplikowania Twojego kodu. **Wszystkie elementy i techniki języka Sass, które** dotąd poznałeś, powinny być używane zgodnie z przeznaczeniem, odpowiednio do sytu**acji, a ich zastosowanie powinno mieć logiczne uzasadnienie i przynosić wymierną korzyść**. Pamiętaj również, że napisany przez Ciebie kod może kiedyś (mimo że teraz tego nie zakładasz) trafić w inne ręce, a im prostszy i bardziej czytelny kod, tym łatwiej będzie nowej osobie rozpocząć z nim pracę.

Przypomnijmy zatem dobre praktyki stosowania rozmaitych technik w Sass.

## Stosowanie zmiennych

Zmienne z pewnością są jednym z największych dobrodziejstw Sass i elementem wykorzystywanym niemal w każdym projekcie opartym na Sass. **Użycie zmiennej ma sens wtedy, kiedy wiemy, że określona wartość ma istotne znaczenie, będzie wykorzystywana wielokrotnie**

**lub jej wartość będzie często zmieniana**. Przykładowo idealnymi kandydatami są wartości określające wielkości w projekcie (marginesy, rozmiary czcionek itp.), definiujące siatkę (ilość i szerokość kolumn, wielkość odstępów) bądź kolory tworzące schemat kolorystyczny.

Wszelkie miary warto określić w oparciu o jakąś regułę bądź schemat proporcji. Przykładowo możemy zdefiniować globalny margines, a wszystkie marginesy dla poszczególnych elementów definiować w jego oparciu. Podobnie można uczynić z rozmiarem tekstu na stronie bądź z kolorami — zdefiniować kolor podstawowy, a inne kolory w schemacie wygenerować za pomocą odpowiednich funkcji do pracy z kolorami. W ten sposób zmiana wariantu kolorystycznego strony będzie niezwykle szybka.

Pamiętaj również o **przyjęciu spójnej konwencji nazewnictwa zmiennych**, abyś nie musiał później zastanawiać się, jakie jest znaczenie konkretnej zmiennej. Zagadnienie to będzie omówione bardziej szczegółowo w dalszych rozdziałach.

Zmienne warto wydzielić do osobnego pliku konfiguracyjnego (np. *_variables.scss*, *_config.scss*). Na początku takiego pliku na ogół na samej górze definiujemy zmienne globalne, bazowe, będące podstawą naszego projektu. Mogą się tam znaleźć definicje podstawowych kolorów, miar, czcionek i elementów typografii, parametrów siatki czy wartości `z-index`. Tutaj warto również zadbać o porządek, grupując zmienne, czyli np. najpierw definicja siatki, później czcionek, kolorów itd.

Po zdefiniowaniu bazowych zmiennych możemy określić zmienne poszczególnych komponentów, np. nagłówka, artykułu, przycisków.

Przykładowo plik taki może wyglądać następująco:

```
// ================================================================
// GLOBAL VARIABLES
// ================================================================

// Base
$base-margin: 0.625em;
$base-padding: 0.625em;
$base-radius: 0.25em;

// Grid
$grid-column: 4.2358em;
$grid-gutter: 1.618em;
$grid-columns: 12;
$grid-max-width: 1280px;

// Colors
$color-brand: #cc6699;
$color-primary: #333333;
$color-secondary: lighten($color-brand, 20%);
$color-link: blue;

$body-bg: #ffffff;
$text-color: #111111;
```

```
// Fonts
$font-family-sans-serif:  "Helvetica Neue", Helvetica, Arial, sans-serif;
$font-family-serif:       Georgia, "Times New Roman", Times, serif;

$text-size: 1rem;

// Global z-index
$zindex-topbar:            10000;
$zindex-tooltip:           10100;
$zindex-modal:             10200;


// ================================================================
// COMPONENT VARIABLES
// ================================================================

// Header
$header-bg: #333333;
$header-height: 56px;

// Buttons
$button-radius: $base-radius;
$button-padding: $base-padding $base-padding*2;
```

Należy przypomnieć, że istnieją inne typy danych, takie jak na przykład mapy, które idealnie nadają się do grupowania wielu zmiennych. W powyższym przykładzie moglibyśmy z powodzeniem zamienić listę zmiennych definiujących właściwość `z-index` w następujący sposób:

```
$zindex: (
  'topbar': 10000,
  'tooltip': 10100,
  'modal': 10200,
);
```

Warto również pamiętać o istnieniu flagi `!default`, która pozwala określać domyślną wartość zmiennej w przypadku, gdy przewidujemy możliwość jej nadpisania.

Dobrym pomysłem jest również podpatrzenie, jak z definiowaniem ustawień i zmiennych radzą sobie uznane, popularne frameworki, takie jak Bootstrap (*https://github.com/twbs/bootstrap-sass/blob/master/assets/stylesheets/bootstrap/_variables.scss*) czy Foundation (*https://github.com/zurb/foundation-sites/blob/develop/scss/settings/_settings.scss*). Pozwoli Ci to poznać dobre praktyki innych deweloperów.

## Zagnieżdżanie selektorów

Możliwość zagnieżdżania jest bardzo przydatną cechą języka Sass — pozwala na grupowanie kodu i czyni go łatwiejszym w zarządzaniu, jednak zbyt głębokie zagnieżdżanie może być przyczyną problemów. Technika ta przyczyniła się do złej sławy preprocesorów na samym początku ich istnienia — krytykowano je dlatego, że generowały rozległe, nader specyficzne selektory i nieeleganckie, rozdęty kod CSS. Jednak jak to często bywa, przyczyną było nieprawidłowe używanie zagnieżdżania i nieprzestrzeganie kilku prostych zasad.

Po pierwsze: **stosując zagnieżdżenia, staraj się unikać odwzorowywania struktury kodu HTML**. Odwzorowanie struktury HTML w selektorach skutkuje sztywnym kodem, który ciężko jest przenosić i stosować do innych elementów interfejsu. Ponadto może prowadzić do utworzenia bardzo długich i mocno specyficznych selektorów, co w rezultacie ma również negatywny wpływ na wydajność. Używaj więc klas i myśl w kategoriach mikrokomponentów, które można stosować w dowolnym miejscu strony, niezależnie od struktury dokumentu HTML. Unikaj więc zagnieżdżeń, które generują selektory typu:

```
body #page .main .box div.body > div a
```

Drugą zasadą powszechną wśród wielu deweloperów jest **stosowanie maksymalnie 3 – 4 poziomów zagnieżdżeń**, choć niektórzy idą dalej, rekomendując maksymalnie 2 poziomy. Choć nie zawsze jest to możliwe, zasada ta skutecznie zmusza nas do pisania kodu w oparciu o komponenty interfejsu użytkownika i minimalizuje ryzyko tworzenia zbyt specyficznych selektorów. Jeżeli w Twoim kodzie zagnieżdżenia schodzą na głębsze poziomy bądź jesteś zmuszony do używania deklaracji `!important`, powinieneś pomyśleć o zoptymalizowaniu i przepisaniu kodu.

Jeśli odpowiednio przemyślisz strukturę elementu interfejsu użytkownika i zastosujesz odpowiednie nazewnictwo klas (stosując takie konwencje jak choćby *OOCSS*, *BEM* czy *Atomic Design*), jest wysoce prawdopodobne, że w ogóle nie będziesz musiał stosować zagnieżdżania lub je zminimalizujesz. Oczywiście nie chodzi o to, aby zagnieżdżania unikać całkowicie. Należy **używać go tam, gdzie trzeba, a nie dlatego, że po prostu można**.

Zagnieżdżanie jest rekomendowane na przykład wtedy, gdy używamy pseudoklas i pseudoelementów. Pozwala to utrzymać porządek i strukturę komponentu w jednym miejscu.

Pamiętaj również, że warto od czasu do czasu (szczególnie na początku pracy z Sass) **sprawdzać wygenerowany kod CSS**. Jeśli widzisz w nim zbyt długie bądź niepożądane selektory, warto wrócić do kodu Sass i nieco go zoptymalizować.

## Domieszki, rozszerzenia i selektory zastępcze

Techniki te opisałem dość szczegółowo w części I książki i na tym etapie powinieneś wiedzieć, kiedy każda z tych technik ma zastosowanie.

Domieszki to bardzo przydatne narzędzie w Sass, które umożliwia pisanie kodu wielokrotnego użytku, bez niepotrzebnych powtórzeń. Tak jak w pozostałych przypadkach należy jednak pamiętać o umiarze i prostocie. Tak jak nie ma sensu przerabianie każdego niemal fragmentu kodu na domieszkę, tak również należy wystrzegać się tworzenia bardzo skomplikowanych i rozbudowanych domieszek, które mają „robić wszystko". Jeśli napisana przez Ciebie domieszka osiąga zbyt duże rozmiary (przykładowo nie mieści się na jednym ekranie edytora), to być może należy pomyśleć o rozbiciu jej na mniejsze.

Często można jeszcze spotkać domieszki pisane tylko po to, aby generowały kod CSS obsługujący przedrostki producentów. Taka praktyka była kiedyś powszechna, jednak odkąd istnieją specjalne narzędzia zajmujące się problemem kompatybilności przeglądarek i obsługi prefiksów, takie rozwiązanie nie ma sensu. Szerzej opisuję ten problem w części IV — „Przegląd innych domieszek biblioteki Bourbon".

Jeśli chodzi o stosowanie rozszerzeń, temat ten został obszernie omówiony w pierwszej części książki. Dla usystematyzowania wiedzy powtórzmy najistotniejsze wskazówki:

- W zdecydowanej większości przypadków użycie domieszek jest sensowniejsze. W przeciwieństwie do rozszerzeń — trudniej o nieprzewidziane rezultaty, a sam kod Sass jest również łatwiejszy do zrozumienia.
- Używaj domieszek, jeśli istnieje potrzeba przekazania argumentów, a gdy chcesz wykorzystać wielokrotnie powtarzające się fragmenty kodu, dobrym rozwiązaniem będą selektory zastępcze (ang. *placeholders*).
- Dyrektywy `@extend` używaj tylko wtedy, gdy istnieje silna relacja między elementami, a jeśli takiej relacji nie ma, używaj domieszek. Grupowanie selektorów, jako rezultat rozszerzania, nie może następować przypadkowo, powinny być one logicznie powiązane ze sobą.
- Nie wykorzystuj dyrektywy `@extend` do rozszerzania selektorów (klas) występujących w arkuszach więcej niż raz ani selektorów zagnieżdżonych — pozwoli to uniknąć nieprzewidzianych sytuacji. Jeszcze bezpieczniej będzie wykorzystać w takiej sytuacji selektory zastępcze, tworzone raz w całym arkuszu.
- Jeśli nie jesteś w stanie łatwo przewidzieć rezultatu działania dyrektywy `@extend` w danym miejscu bądź wynikowy kod CSS zawiera fragmenty, których się nie spodziewałeś, popraw swój kod, wykorzystując selektory zastępcze lub domieszki.
- Sprawdzaj podczas używania dyrektywy `@extend`, czy wynikowy kod CSS jest zgodny z Twoimi przewidywaniami.
- Selektory zastępcze możesz z powodzeniem wykorzystać w tandemie z domieszkami. Oto dobry przykład:

```
%btn {
  border: 0;
  cursor: pointer;
  display: block;
  font-size: 14px;
  margin: 10px;
  padding: 10px;
  text-align: center;
}

@mixin  button($bg) {
  @extend %btn;
  background-color: $bg;
  &:hover {
    background-color: lighten($bg,10%);
  }
}

.btn-default {
  @extend %btn;
  border: 1px solid grey;
}
```

```
.btn-primary {
  @include button(blue);
}
```

## Funkcje i pętle

Jak już wiesz, funkcja sama z siebie nie generuje żadnego kodu CSS, zwraca natomiast wartość jakiegoś działania, która może być użyta w kodzie. Jeśli wykonujesz wielokrotnie jakieś obliczenia, warto wykorzystać do tego celu gotowe funkcje lub stworzyć własną.

Nie wahaj się więc używać funkcji, szczególnie w przypadkach takich jak wyliczenia wielkości i miar (np. elementy układu strony), konwersje jednostek (np. z wartości `px` na `em`) czy operacje na kolorach (np. zmiana odcienia, rozjaśnienie koloru). Wykorzystajmy cechę Sass, która pozwala wykonywać obliczenia zamiast na sztywno wpisywać wynik działania w kodzie.

Zastosowanie funkcji będzie również konieczne, jeśli planujesz wykorzystać bardziej złożone typy danych, jak listy lub mapy.

Pętle to dyrektywy sterujące umożliwiające automatyzację generowania kodu. W praktyce najczęściej wykorzystuje się pętle `@for` oraz `@each`. Pamiętaj, że **zasadniczym celem stosowania pętli jest zautomatyzowanie pewnych zadań, redukcja powtórzeń kodu oraz uczynienie naszego kodu bardziej zwięzłym**. Jeśli masz do napisania reguły, w których zauważasz jakiś schemat (np. selektory bądź właściwości CSS różniące się o ustaloną wartość numeryczną), to warto zastosować pętle. Pamiętaj o technice interpolacji, która będzie w takim przypadku pomocna. Dobrym przykładem może być generowanie stylów do określania szerokości kolumn w siatce.

Bardziej zaawansowanym deweloperom pętle przydadzą się również do pracy z mapami, na przykład do generowania schematów kolorystycznych.

Przypomnijmy jeszcze ogólne wytyczne do stosowania poszczególnych rodzajów pętli:

- Jeśli znasz dokładną liczbę powtórzeń (iteracji) pętli lub potrzebujesz wygenerować określoną ilość reguł, powinieneś wykorzystać pętlę `@for`. Jest ona także wskazana tam, gdzie konieczne jest użycie zmiennej licznikowej, np. generując selektory typu `:nth-child()`. Zmienna licznikowa jest częścią pętli, jest więc zawsze dostępna.
- Jeśli pracujesz z listami lub mapami, wygodniejsze będzie użycie pętli `@each` z uwagi na jej specyficzny mechanizm działania.
- Pętla `@while` ma podobne zastosowanie jak `@for`, z tą różnicą, że daje nam większą kontrolę nad zmienną licznikową. W praktyce jest jednak najrzadziej wykorzystywanym typem pętli.

## Importowanie

Importowanie plików cząstkowych przy użyciu dyrektywy `@import` jest konieczne niemal w każdym projekcie. Jak wiesz, umożliwia podział całości kodu na mniejsze, osobne pliki i ich późniejsze łączenie, co w przypadku dużych, rozbudowanych projektów znacznie ułatwia organizację arkuszy stylów.

W dalszym rozdziale dowiesz się, jakie są różne strategie podziału i organizacji kodu. Teraz natomiast warto wspomnieć o kilku podstawowych zasadach dotyczących samego importowania.

Przede wszystkim staraj się, aby **zawsze istniał w Twoim projekcie jeden główny plik Sass, który będzie służył wyłącznie do importowania innych plików** i „składania" poszczególnych modułów w całość. Plik taki najczęściej przyjmuje postać *main.scss*, *global.scss* czy też *<nazwa-aplikacji>.scss* i znajduje się w głównym folderze z arkuszami Sass. Warto, aby plik ten był również uporządkowany i przejrzysty. W zależności od struktury naszego projektu plik taki mógłby wyglądać następująco:

```
#vendor libraries
@import "vendor/normalize";
@import "vendor/mq";
@import "vendor/bourbon/bourbon";

#settings
@import "config/variables";

#helpers
@import "helpers/mixins";
@import "helpers/helpers";
@import "helpers/animations";

#base styles
@import "base/base";
@import "base/typography"

#components
@import "components/header";
@import "components/menu";
@import "components/search";
```

Pamiętaj, że kolejność importowania jest istotna, dlatego pliki zawierające na przykład definicje zmiennych bądź elementy wykorzystywane w obrębie całego projektu (domieszki, biblioteki zewnętrzne) muszą być dołączane jako pierwsze.

Gdy z jakiegoś powodu musisz ograniczyć działanie selektora bądź uczynić go bardziej specyficznym, możesz zastosować **importowanie zagnieżdżone**:

```
#main-page {
  @import "base/base";
  @import "base/typography"
}
```

Przy okazji importowania warto jeszcze wspomnieć o jednej rzeczy, mianowicie o „niechcianym kodzie", inaczej określanym jako „plik wstydu". To proste rozwiązanie zaproponowali panowie Chris Coyier, Harry Roberts i Dave Rupert (*http://csswizardry.com/2013/04/shame-css/*). O co chodzi?

Mianowicie istnieją sytuacje, gdy niestety jesteśmy zmuszeni do umieszczenia kodu, z którego nie jesteśmy specjalnie dumni, jest on konieczny do obsługi jakiejś starszej wersji strony, nadpisania reguł, zawiera tymczasowe poprawki i łaty czy też jest to zapomniany, ale niezbędny kod CSS, którego nikt nie ma czasu porządnie przepisać. Cały ten niechciany kod warto wtedy umieścić w jednym pliku o nazwie *_shame.scss* (bądź *_shame.css*). Plik ten importujemy na samym końcu pliku głównego (*_global.scss*). Warto też dodać komentarze wyjaśniające powód umieszczenia tych stylów.

## Instrukcja warunkowa

Jak wiesz, Sass umożliwia wykonanie instrukcji warunkowej przy użyciu dyrektyw `@if` i `@else`. Najczęściej stosuje się je w przypadku tworzenia domieszek, funkcji, własnych bibliotek ogólnego przeznaczenia czy po prostu komponentów wielokrotnego użytku, kiedy w jednym fragmencie kodu potrzebujemy uzyskać inny rezultat w zależności od określonego warunku.

Spójrzmy na poniższy przykład:

```
@function bg-text-color($bg-color) {
  @if lightness($bg-color) < 50% {
    @return white;
  } @else {
    @return black;
  }
}

.btn-dark {
  background-color: #333333;
  color: bg-text-color(#333333);
}

.box-light {
  background-color: #eeeeee;
  color: bg-text-color(#eeeeee);
}
```

Tworzymy ogólną funkcję, która ma za zadanie zwrócić prawidłowy kolor tekstu w zależności od koloru tła (prawidłowy, czyli widoczny niezależnie od koloru tła). Dzięki temu prostemu rozwiązaniu wystarczy użycie jednej funkcji i zawsze zostanie automatycznie wygenerowany prawidłowy kolor.

## Debugowanie kodu przy użyciu map źródłowych (source maps)

*Source maps* to dodatkowe pliki generowane przez kompilator, które mogą ułatwić nam znajdowanie błędów w kodzie. Jaki to ma związek z optymalizacją i wydajnym kodem? Otóż może być to przydatne szczególnie w kwestii analizy generowanych selektorów. Jeśli zdarzy nam się, że

w wynikowym pliku CSS otrzymamy zbyt specyficzny, długi, rozbudowany bądź nieoczekiwany selektor, narzędzia deweloperskie w przeglądarce wraz z mapami źródłowymi umożliwią nam szybkie znalezienie przyczyny.

Mapy źródłowe pozwalają nam zobaczyć, która linijka kodu Sass odpowiada za wygenerowanie konkretnej reguły w CSS. Dzięki temu bez konieczności przeszukiwania kodu źródłowego panel deweloperski w przeglądarce od razu wskaże nam właściwe miejsce.

Aby można było włączyć generowanie *source maps*, konieczne jest ustawienie odpowiedniego trybu kompilacji. Gdy w wierszu poleceń uruchamiamy standardową komendę do „obserwowania" i kompilacji plików Sass, dodajemy dodatkowo parametr `--sourcemap`:

```
sass --watch sass_project/scss:sass_project/css --sourcemap
```

Jeśli teraz zajrzysz do wygenerowanego pliku CSS, zobaczysz, że na końcu znajduje się dodatkowa linijka komentarza o postaci:

```
/*# sourceMappingURL=styles.css.map */
```

Ten fragment kodu to nic innego jak wskazanie na plik mapy źródłowej, który to z kolei ma za zadanie mapować skompilowany kod CSS z deklaracji w pliku Sass.

Drugą czynnością, którą musimy wykonać, aby skorzystać z tego narzędzia, jest sprawdzenie obsługi map źródłowych w przeglądarce — obsługuje je większość nowoczesnych przeglądarek takich jak Chrome, Firefox czy Safari.

### Chrome

W przeglądarce Chrome od wersji 39 obsługa source maps jest włączona domyślnie, więc nie musisz niczego konfigurować, aby z nich skorzystać. Jeśli z jakiegoś powodu ich obsługa jest wyłączona, możesz to zmienić, wchodząc w *Narzędzia dla programistów*, a następnie wybierając ustawienia i odnajdując opcję *Włącz obsługę map źródłowych CSS* (ang. *Enable CSS source maps*).

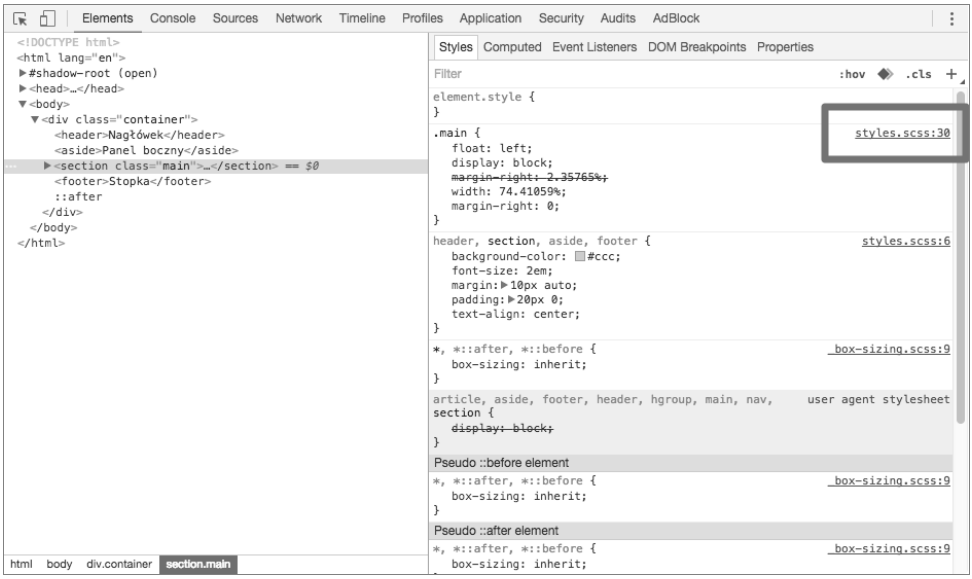
### Firefox

W przeglądarce Firefox od wersji 29 obsługa map źródłowych CSS jest również włączona. Możesz to sprawdzić, włączając *Inspektora*, a następnie wejść w *Ustawienia narzędzi* i odszukać opcję *Oryginalne źródła* (ang. *Show original source maps*).

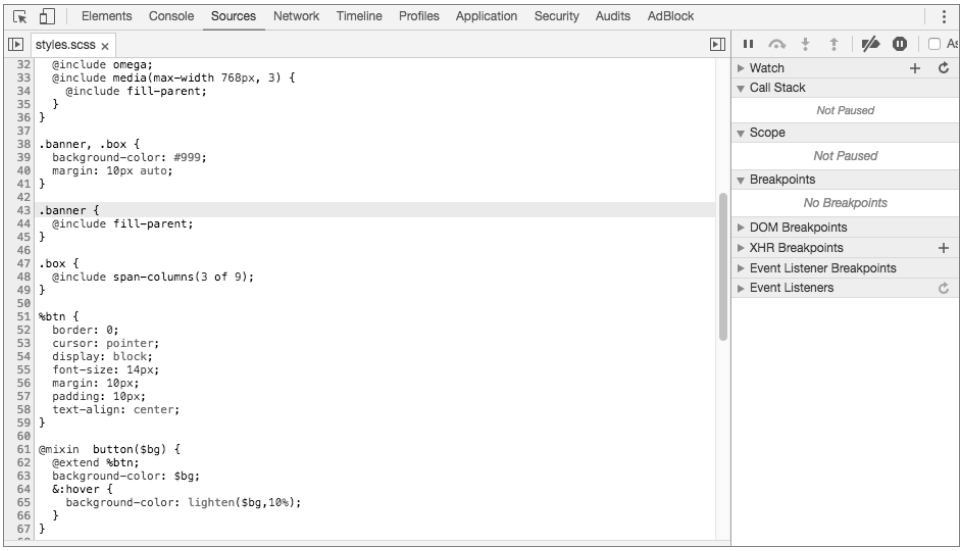
### Safari

W Safari również obsługa map jest włączona domyślnie. Żadna dodatkowa konfiguracja nie jest potrzebna.

Kiedy obsługa map jest włączona, posługiwanie się nimi jest nadzwyczaj proste. Wystarczy w *Inspektorze WWW* otworzyć panel stylów, zbadać odpowiedni element w dokumencie HTML bądź znaleźć interesującą nas regułę CSS. Po jej prawej stronie zobaczymy wskazanie na linijkę kodu Sass, która jest odpowiedzialna za wygenerowanie tej reguły CSS. Poniższy rysunek ukazuje to na przykładzie przeglądarki Chrome:




Kiedy klikniemy z kolei link przy danej regule CSS, otworzy się panel z plikiem źródłowym Sass, wskazując konkretną linijkę kodu.




W powyższym przykładzie znalezienie odpowiedniej linijki w kodzie Sass bez użycia map źródłowych oczywiście byłoby bardzo łatwe, jednak w przypadku mocno rozbudowanego projektu i podziału kodu na wiele plików, stosowania zagnieżdżeń czy rozszerzeń ich pomoc może okazać się niezastąpiona.

# Skorowidz

## D



## E



## F

## G



## H



## I



## J



## K

## P



## R



## S

## T



## V



## W

## Z

**Sass to metajęzyk**, który umożliwia szybsze i wydajniejsze tworzenie bardziej zaawansowanych arkuszy stylów, niż jest to możliwe przy użyciu samego języka CSS. Co więcej, ten świetny metajęzyk zapewnia pełną zgodność ze standardowym CSS-em. Sass eliminuje wady i ograniczenia CSS, a ponadto oferuje możliwości znane z innych języków programowania. Twórcom stron WWW przydaje się to szczególnie w dużych i rozbudowanych projektach komercyjnych. A wszystko to można osiągnąć niewielkim kosztem — jeśli znasz CSS, opanowanie Sass nie wymaga zbyt dużo czasu ani wysiłku.

**Jeśli tworzysz strony WWW** i chcesz wzbogacić swój warsztat o znajomość jednego z ciekawszych oraz najdynamiczniej rozwijających się narzędzi designerskich i deweloperskich, sięgnij po książkę *Sass. Nowoczesne arkusze stylów*. Zrozumiesz dzięki niej zarówno podstawowe, jak i bardziej zaawansowane elementy składni języka, poznasz dobre praktyki tworzenia projektów Sass oraz nauczysz się je optymalizować. Zaznajomisz się również z praktycznymi przykładami zastosowania tego narzędzia oraz bibliotekami i dodatkami rozszerzającymi jego możliwości. Daj się wprowadzić w świat Sass!

- **Podstawowe informacje na temat preprocesora Sass, jego działania i składni**
- **Zaawansowane elementy języka wraz z przykładami**
- **Liczne przykłady praktycznych technik przydatnych podczas tworzenia stron WWW**
- **Biblioteki i narzędzia ułatwiające pracę projektanta**
- **Techniki optymalizacji i zasady tworzenia prawidłowego kodu**
- **Metody pracy przy rozbudowanych projektach z wykorzystaniem wzorców i architektury**
- **Tworzenie skórek przy użyciu Sass i Bootstrap**
- **Instalacja i konfiguracja środowiska pracy i dodatków**

## Opanuj Sass i twórz przebojowe strony WWW w nowoczesny sposób!



Helion

księgarnia internetowa

http://helion.pl

zamówienia telefoniczne

0 801 339900

0 601 339900

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

ISBN 978-83-283-2619-4

9 788328 326194

Informatyka w najlepszym wydaniu

cena: 59,00 zł