CSS Utility classes

Na przykładzie TailwindCSS

Mam wrażenie, że świat core'owych technologii frontendowych rozwija się w znacznie wolniejszym tempie i mniejszej skali niż świat uniwersalnych języków imperatywnych. Standard HTML 5 został opublikowany przeszło dekadę temu, a rozwój CSS i JavaScript odbywa się w zakresie pojedynczych obszarów i funkcjonalności (ostatnie duże zbiorcze, kompletne publikacje wersji odbyły się odpowiednio w 2018 i 2015 roku). Ponieważ więc na razie dużej rewolucji w tym obszarze nie możemy się raczej spodziewać, niektórzy programiści postanowili wziąć sprawy w swoje ręce i budując rozwiązania na fundamencie istniejących technologii, zaproponowali zupełnie nowe podejście do programowania frontendu.

DAWNO, DAWNO TEMU…

Cofnijmy się na chwilę do starych, dobrych czasów, czyli początku lat 2000, kiedy to w świecie frontendu (który jeszcze wtedy nie wiedział, że jest światem frontendu) królował standard HTML 4.01. Strony internetowe w przeważającej większości były wówczas po prostu… no, stronami internetowymi (w odróżnieniu od współczesnych *aplikacji webowych*), treści modyfikowało się poprzez ręczne podmienianie plików na serwerze, a zaimplementowanie ujednoliconego wyglądu strony niejednemu programiście spędzało sen z powiek. Nic zresztą dziwnego, bo mówimy przecież o tym wycinku historii, w którym każda przeglądarka próbowała narzucić światu swoje standardy i napisanie strony, która na każdej z nich wyglądałaby tak samo, graniczyło z cudem.

W tych właśnie czasach w standardzie języka HTML pojawił się szereg tagów, których zadaniem było definiowanie wyglądu prezentowanych danych. I tak, mogliśmy skorzystać ze znacznika `<font>`, za pomocą którego dało się ustalić czcionkę, krój i kolor fragmentu tekstu, znacznika `<center>`, który pozwalał wygodnie umieścić coś pośrodku ekranu, a także ze znaczników `<b>`, `<i>` oraz `<u>`, które działały tak samo jak odpowiadające im przyciski w popularnych (i często pozyskiwanych wówczas… powiedzmy, *nieoficjalną* drogą) pakietach biurowych.

Obecność w języku HTML tagów regulujących wygląd elementów strony internetowej powodowała więcej kłopotów niż pożytku i to z kilku powodów. Na przykład tag `<b>` technicznie służył do pogrubiania tekstu, ale z drugiej strony nic nie stało na przeszkodzie, żeby za pomocą arkusza stylów sprawić, aby zamiast tego, na przykład, zmieniał kolor tekstu na różowy (ba, niektóre współczesne biblioteki korzystają z taga `<i>` do… wstawiania obrazków). Analogicznie, kolor tekstu można było ustawić za pomocą arkusza stylów, ale potem równolegle zmodyfikować go tagiem `<font>`. Twórcy języka HTML szybko uświadomili sobie, że język ten zaczął niepotrzebnie robić dwie rzeczy naraz: definiował jednocześnie *logiczną strukturę* dokumentów, ale i równocześnie ich *wygląd*. Mało tego: o ile to pierwsze robił w miarę przyzwoicie, to to drugie raczej marnie, a to między innymi dlatego, że pozwalał on na sterowanie tylko niewielkim zbiorem wizualnych cech projektowanego dokumentu (głównie kolorami, grubościami linii oraz czcionkami). Dublował on w tym zakresie język CSS, który rozwijał się praktycznie w tym samym tempie, co HTML (w 1996 roku opublikowano niezależnie standard HTML 2.0, będący tak naprawdę pierwszym standardem tego języka, oraz standard CSS 1).

Nic więc dziwnego, że wraz z premierą HTML 5 z definicji tego języka wyleciała z hukiem większość tagów służących tylko do definiowania wyglądu (między innymi niesławny tag `<font>`, ale też `<big>`, `<center>` czy `<strike>`), a w ich miejsce pojawił się szereg takich, które ułatwiły precyzyjne opisywanie *logicznych* fragmentów dokumentu, jak na przykład `<nav>`, `<article>` czy `<main>`.

Definicja języka HTML 5 pozostawiła jednak małą furtkę, która wciąż pozwalała na definiowanie wyglądu dokumentu bezpośrednio wewnątrz znaczników, a mam tu na myśli atrybut `style`. O ile więc uparci maniacy wizualnego formatowania dokumentów stracili możliwość stosowania wygodnego skrótu w postaci `<font color="red">`, to wciąż mogli przekuć go na napisany w locie fragment CSSu: `<span style="color: red;">`.

Jak to jednak zwykle bywa, życie szybko zweryfikowało takie podejście do projektowania stron internetowych (a potem również aplikacji internetowych). Opisana powyżej metoda pozwalała bowiem pisać strony szybko i wygodnie, ale tylko do momentu, w którym nagle okazywało się, że zamiast czerwonego tekstu potrzebujemy teraz tekstu pogrubionego i powiększonego. Aha – ale pod warunkiem, że nie znajduje się w obszarze menu, bo tam powinien być niebieski. No i oczywiście poza fragmentami zaprezentowanego na stronie kodu źródłowego, gdzie faktycznie powinien pozostać czerwony. Podejrzewam, że nawet najbardziej zagorzali zwolennicy formatowania „w miejscu" poddawali się wówczas i zaczynali prawidłowo stosować arkusze stylów.

Od pojawienia się HTML 5 wiodącą metodą projektowania stron internetowych stał się trójpodział władzy: HTML odpowiedzialny jest za strukturę dokumentu, CSS – za jego wygląd, zaś JavaScript – za zachowanie. W ten sposób zamiast bezmyślnie zmieniać kolor tekstu na czerwony, zastanawiamy się najpierw, *czym* jest tenże fragment tekstu, stosujemy doń odpowiedni tag HTML, a na koniec malujemy to *coś* CSSem na czerwono i problem mamy z głowy.

Ale również i CSS ma swoje wady.

Przede wszystkim arkusze stylów współczesnych stron internetowych potrafią składać się z setek, jeśli nie tysięcy linii. Wynika to między innymi z faktu, że wcale nie tak łatwo jest zdefiniować skończony i kompletny zbiór wszystkich semantycznych elementów aplikacji webowej w taki sposób, aby był on jednocześnie kompaktowy. Bardzo szybko okaże się, że o ile na stronie mamy wprawdzie mnóstwo linków i przycisków, to jednak linki i przyciski te muszą wyglądać inaczej, jeśli znajdują się w menu, w treści artykułu, w formularzu, w tabeli czy też w stopce – a odpowiedni wyjątek trzeba przecież stworzyć również dla tych sytuacji, w których linkiem jest na przykład obrazek. Idąc dalej, niedoskonałości CSS powodują trudne do opanowania mnożenie się wszelakich *wrapperów*, *containerów*, *fixów* i *centerów*, a dodatkową dawkę problemów dorzucają również rozbudowane definicje selektorów, które potrafią czasem przyprawić o ból głowy. Ostatnio miałem na przykład do czynienia z sytuacją, w której w celu naprawy błędu pomiędzy dwa `<div>`y musiałem wstawić trzeci, co położyło kompletnie wygląd aplikacji, bo ktoś najwyraźniej założył z góry, że będą one zawsze różnić się tylko jednym poziomem zagnieżdżenia.

No i w ten sposób siedzi człowiek, sumiennie projektując kolejne dziesiątki i setki klas CSS, wzdychając jednocześnie po cichu – gdy nikt nie widzi – do czasów, w których pogrubienie tekstu osiągało się przez… cóż, po prostu pogrubienie tekstu.

Chyba właśnie tak musieli się czuć twórcy koncepcji *utility CSS classes* (znanej również jako *atomic CSS* lub *functional CSS*). Tyle że oni zamiast narzekać, zakasali rękawy, siedli do metaforycznej deski kreślarskiej i zaczęli się zastanawiać, co da się zrobić, żeby ten stan rzeczy zmienić.

UTILITY CSS CLASSES

Pomysł narzędziowych klas CSS jest bardzo prosty. Zamiast projektować duże klasy opisujące szczegółowo określone semantyczne elementy strony, korzystamy z ogromnego wachlarza malutkich klas, z których każda odpowiedzialna jest za pojedynczy aspekt wyglądu jakiegoś elementu. I o ile znam przypadki, gdy ktoś przygotował taki właśnie wachlarz samodzielnie, to obecnie możemy skorzystać z istniejącego rozwiązania – takiego jak na przykład Tailwind CSS.

Koncepcję tę najłatwiej jest wyjaśnić na przykładzie. Rozważmy więc następujący przykład: zależy nam na przygotowaniu stylów dla uniwersalnego boksa, w którym będziemy potem umieszczać różne informacje, na przykład komunikaty dla użytkownika.

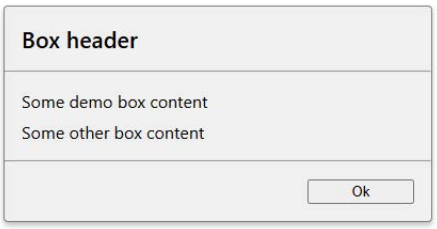
Odpowiedni fragment kodu HTML i CSS może wyglądać następująco:

Listing 1. Kod HTML i CSS dla małego boksa z informacjami

```
<html>
<head>
    <style type="text/css">
        body {
            font-family: 'Segoe UI', 'Verdana', 'Arial', serif;
        }

        .box {
            background: #f0f0f0;
            border: 1px solid #808080;
            border-radius: 4px;
            box-shadow: 0 3px 5px #00000040;
            max-width: 400px;
        }

        .box .box-header {
            padding: 1em;
        }

        .box .box-header h1 {
            font-weight: 500;
            font-size: 16pt;
            margin: 0;
        }

        .box .box-content {
            padding: 1em;
        }

        .box .box-content p {
            margin-top: 0;
            margin-bottom: 0.5em;
        }

        .box .box-content p:last-of-type {
            margin-bottom: 0;
        }

        .box .box-footer {
            padding: 1em;
            display: flex;
            flex-direction: row;
            justify-content: end;
            padding: 1em;
        }

        .box .box-footer button {
            min-height: 23px;
            min-width: 100px;
        }

        .box hr {

            margin: 0;
        }

    </style>
</head>
<body>

    <div class="box">
        <div class="box-header">
            <h1>Box header</h1>
        </div>
        <hr />
        <div class="box-content">
            <p>Some demo box content</p>
            <p>Some other box content</p>
        </div>
        <hr />
        <div class="box-footer">
            <button>Ok</button>
        </div>
    </div>

</body>
</html>
```

Oczywiście w normalnych warunkach style umieścilibyśmy w osobnym pliku współdzielonym przez wszystkie strony, ale tym razem zawarłem je wewnątrz kodu HTML, abyśmy cały czas mieli na uwadze, ile kodu w każdym przypadku musimy napisać samodzielnie.

Końcowy efekt możemy podziwiać na Rysunku 1. Szału może nie ma, ale cel został osiągnięty.

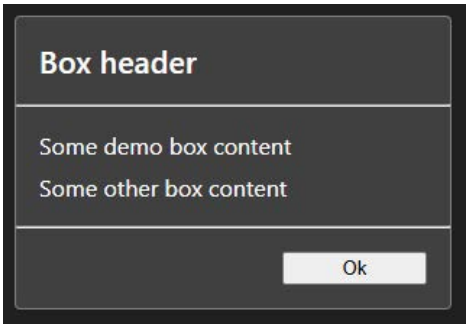


Rysunek 1. Prosty boks zaprojektowany w HTML i CSS

No i niby wszystko jest w porządku, ale tak naprawdę nie jest to wcale kompletna implementacja. Tak napisany CSS nie jest bowiem responsywny: być może warto byłoby zastanowić się, jak powyższy boks powinien wyglądać na mniejszych i większych ekranach. Dobrze byłoby również dostarczyć ciemną wersję tematu dla projektowanej strony, aby ta lepiej dopasowywała się do systemowych ustawień. W praktyce więc powyższy styl powinien wyglądać mniej więcej tak:

Listing 2. Kod CSS uwzględniający responsywność i ciemny temat

```
<html>
<head>
    <style type="text/css">
        body {
            font-family: 'Segoe UI', 'Verdana', 'Arial', serif;
        }

        .box {
            background: #f0f0f0;
            border: 1px solid #808080;
            border-radius: 4px;
            box-shadow: 0 3px 5px #00000040;
            max-width: 400px;
        }

        @media (max-width: 800px) {

            .box {
                max-width: 300px;
            }
        }

        body.dark {
            background-color: #202020;
        }

        body.dark .box {
            background: #404040;
            border: 1px solid #808080;
            box-shadow: none;
            color: white;
        }

        .box .box-header {
            padding: 1em;
        }

        .box .box-header h1 {
            font-weight: 500;
            font-size: 16pt;
            margin: 0;
        }

        .box .box-content {
            padding: 1em;
        }

        .box .box-content p {
            margin-top: 0;
            margin-bottom: 0.5em;
        }

        .box .box-content p:last-of-type {
            margin-bottom: 0;
        }

        .box .box-footer {
            padding: 1em;
            display: flex;
            flex-direction: row;
            justify-content: end;
            padding: 1em;
        }

        .box .box-footer button {
            min-height: 23px;
            min-width: 100px;
        }

        .box hr {

            margin: 0;
        }
    </style>
</head>
<body class="dark">

    <div class="box">
        <div class="box-header">
            <h1>Box header</h1>
        </div>
        <hr />
        <div class="box-content">
            <p>Some demo box content</p>
            <p>Some other box content</p>
        </div>
        <hr />
        <div class="box-footer">
            <button>Ok</button>
        </div>
    </div>

</body>
</html>
```



Rysunek 2. Boks w ciemnym temacie, na małym ekranie

Zobaczmy teraz, jak ten sam kod, z dokładnie tymi samymi cechami, wyglądałby napisany za pomocą gotowych stylów, dostępnych w bibliotece Tailwind CSS:

Listing 3. Korzystamy z Tailwind CSS

```
<html>
<head>
    <script src="https://cdn.tailwindcss.com"></script>
<body>
    <div class="bg-gray-100 border border-gray-800
shadow-md dark:bg-gray-800 dark:border-gray-200
dark:shadow-none rounded-md sm:max-w-[400px]
max-w[300px] m-2">
        <div class="p-4">
            <h1 class="text-lg font-medium m-0">Box header</h1>
        </div>
        <hr class="border-gray-800 dark:border-gray-200" />
        <div class="p-4">
            <p>Some demo box content</p>
            <p>Some other box content</p>
        </div>
        <hr class="border-gray-800 dark:border-gray-200" />
        <div class="p-4 flex flex-row justify-end">
            <button class="border border-gray-800 min-h-4
min-w-28 rounded-sm">Ok</button>
        </div>
    </div>
</body>
</html>
```

WORDPRESS JAKO HEADLESS CMS. INTEGRACJA Z NOWOCZESNYM FRONT-ENDEM DLA ELASTYCZNYCH APLIKACJI

INDEX: 285358

www.programistamag.pl

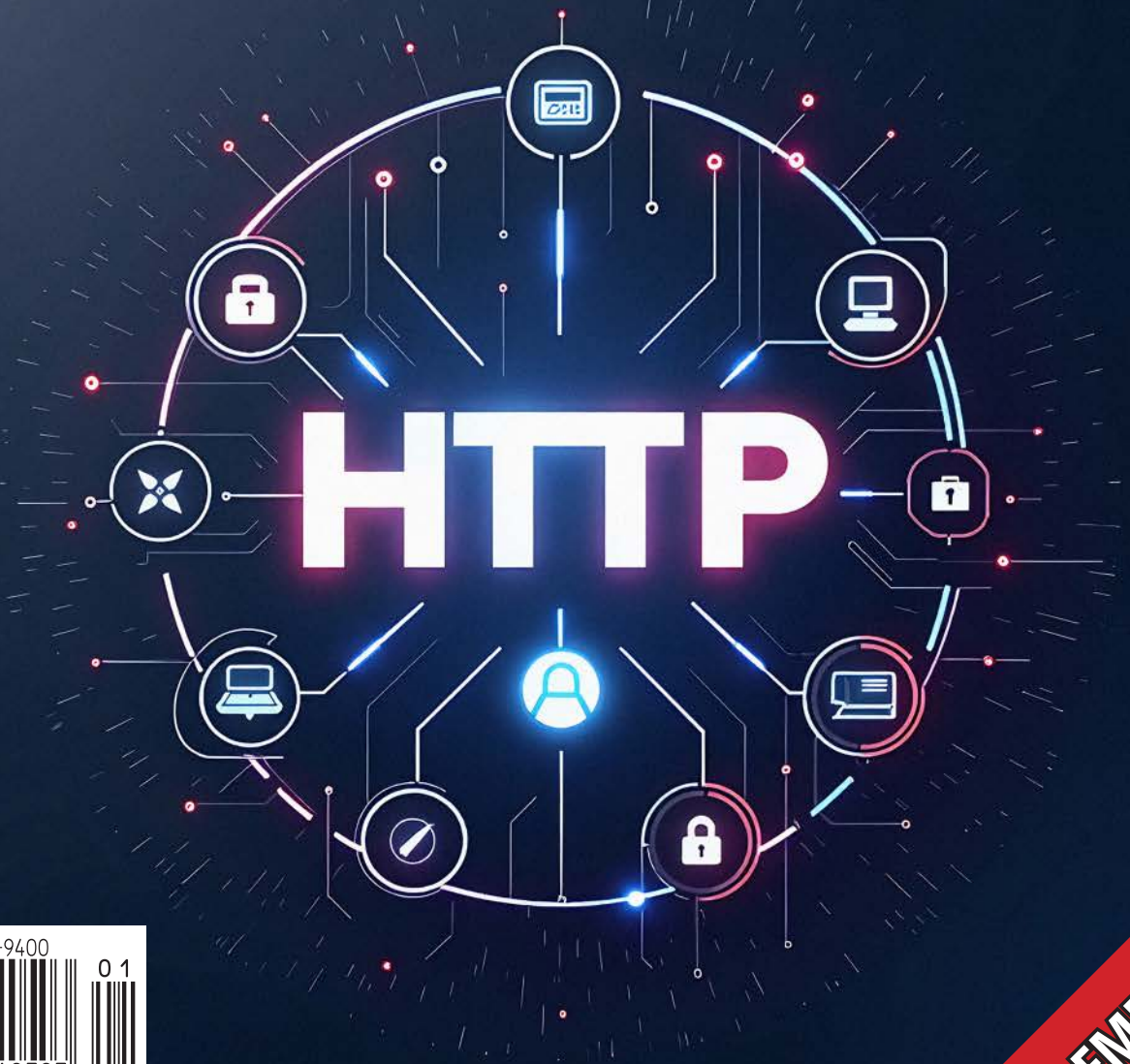
Magazyn programistów i liderów zespołów IT

programista

1/2025 (116)

Cena 32.90 zł (w tym VAT 8%)

JAK DZIAŁA INTERNET



ISSN 2084-9400

01

9 772084 940503

Współbieżność CSP w języku Go

Rekin w strumieniu... ...czyli hakujemy Stream Decka

Gwiazda m… …orutyny, czyli Sea… …aktyce

CSS… …lasses …adzie TailwindCSS

NOWY NUMER JUZ W EMPIKACH