**10. Projekt indywidualny - tydzień 2 z 4**

**Wyzwania:**

* Nauczysz się testować swoją stronę na wielu rozdzielczościach.
* Poznasz dobre praktyki w przygotowywaniu mobilnej nawigacji.
* Dowiesz się w jaki sposób na wąskich ekranach wyświetlać szerokie tabele i wykresy.
* Zrozumiesz czym jest responsywny grid.
* Przygotujesz obrazki dla różnych typów ekranu.

**Wstęp**

W zeszłym tygodniu omówiliśmy sobie, jak rozpocząć projekt, oraz na czym polega podejście komponentowe. W tym tygodniu przygotowaliśmy uzupełnienie dla modułu związanego z Responsive Web Design. Pamiętaj jednak, żeby zajrzeć do tamtego modułu — znajdziesz w nim wiele kluczowych informacji, takich jak np. działanie viewport, który jest niezbędny do poprawnego funkcjonowania responsywności strony.

**10.1. Testowanie RWD**

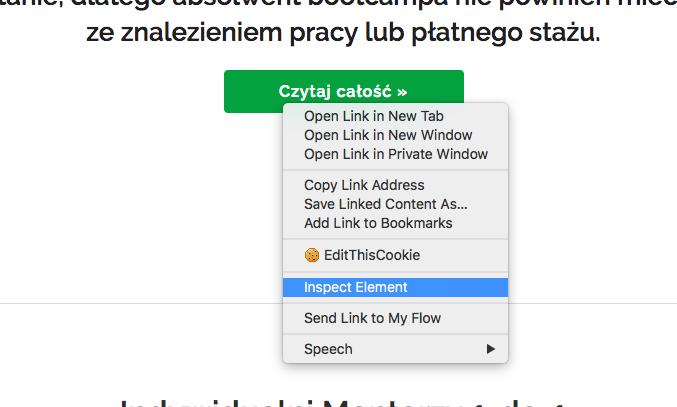
Jak dobrze wiesz z poprzednich modułów, w internecie mamy coraz więcej ruchu mobilnego, co oznacza, że coraz więcej użytkowników do przeglądania stron internetowych wykorzystuje różnego rodzaju urządzenia typu smartphone czy tablet. Dlatego musimy sprawić, aby strona dobrze się wyświetlała przynajmniej na najpopularniejszych rozdzielczościach. Jednak nie warto przy tym pomijać użytkowników ze specyficznymi wielkościami ekranów.

Kiedy zaczęto tworzyć responsywne strony internetowe, to testowanie polegało na sprawdzeniu witryny na tylu urządzeniach, ile się posiadało. Dzisiaj jest to o wiele łatwiejsze i przyjemniejsze, a większość problemów jesteśmy w stanie rozwiązać w przeglądarce na komputerze.

**Developers Tools w przeglądarkach**

Wszystkie popularne przeglądarki mają swoje narzędzia dla developerów stron internetowych. Dzięki nim możemy zobaczyć, jakie style działają na konkretny element, oraz zbadać jego właściwości.

Wystarczy, że klikniesz prawym przyciskiem myszy na dowolnym elemencie na stronie i z menu kontekstowego wybierzesz opcję "Zbadaj element" (ang. *Inspect element*). Otworzy Ci się okienko narzędzi developerskich.

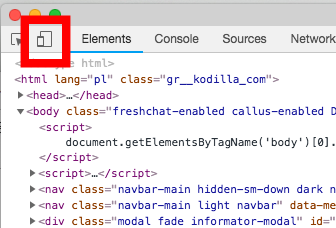
[](https://uploads.kodilla.com/bootcamp/wdp/wdp10/inspect.png)

Więcej o samych narzędziach możesz dowiedzieć się z archiwalnego [webinara Kodilli](https://www.youtube.com/watch?v=cWpUnGzcLrs" \t "_blank).

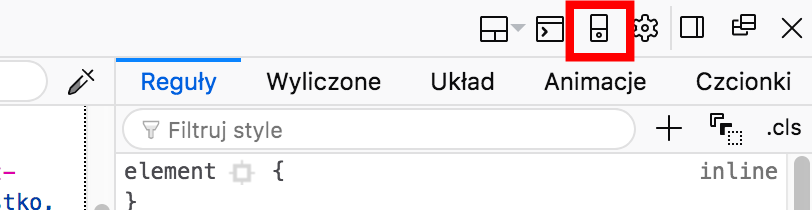
**Tryb responsywny**

Wszystkie przeglądarkowe narzędzia developerskie dysponują tzw. "trybem responsywnym", który służy do testowania naszej strony na różnych rozdzielczościach ekranu.

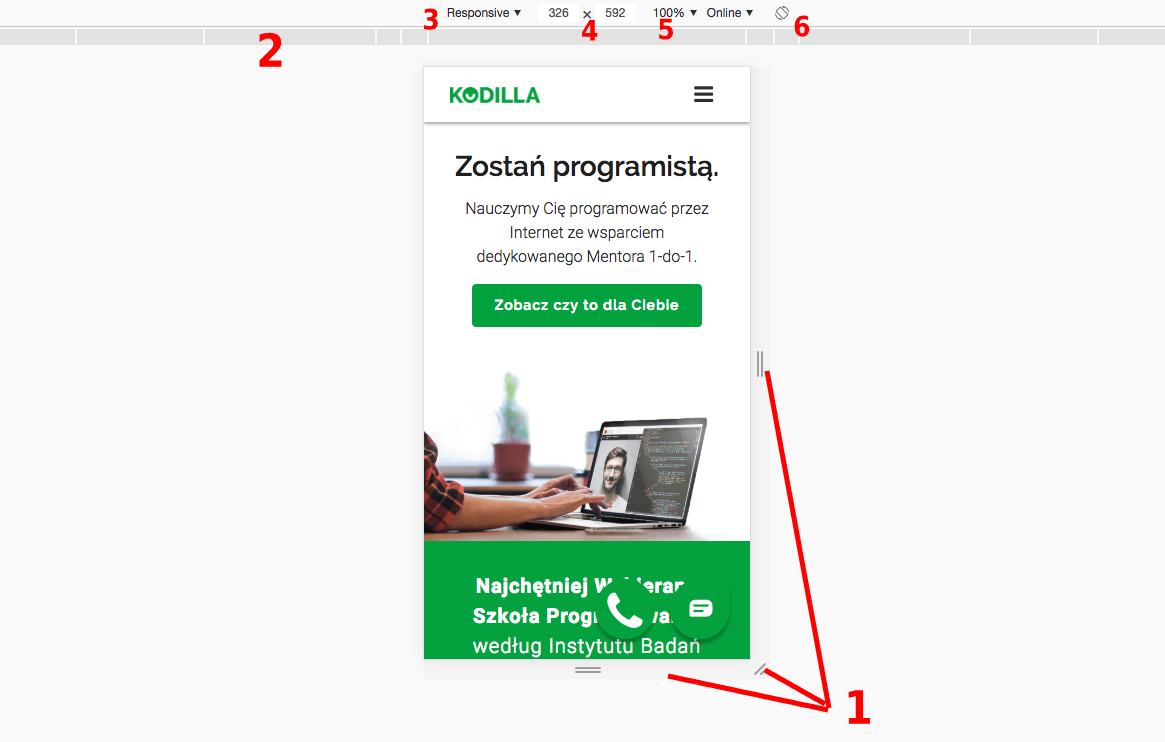
Aby przejść do tego trybu w Google Chrome, Safari czy Operze, wystarczy kliknąć w przycisk telefonu znajdujący się w lewym górnym rogu panelu narzędzi developerskich.

[](https://uploads.kodilla.com/bootcamp/wdp/wdp10/rwd-mode-chrome.png)

Natomiast w Firefoksie znajdziemy go w prawym górnym rogu.

[](https://uploads.kodilla.com/bootcamp/wdp/wdp10/rwd-mode-ff.png)

Kiedy go włączymy, zobaczymy, że nasza strona wygląda zupełnie inaczej. Omówmy sobie elementy tego widoku.

[](https://uploads.kodilla.com/bootcamp/wdp/wdp10/responsive-mode.png)

**1. Suwaki** — wystarczy je przeciągnąć, aby zmienić rozdzielczość symulowanego urządzenia.

**2. Popularne szerokości** — kliknij w odpowiedni panel, a strona dostosuje się do jednej z najpopularniejszych szerokości ekranu. Dobrą praktyką jest dostosowanie strony do każdej z nich. Dzięki temu zyskamy pewność, że obsłużymy większość urządzeń.

**3. Predefiniowane urządzenia** — możemy zdefiniować i wybrać wielkości ekranów różnych urządzeń. Dzięki temu zobaczymy, jak nasza strona może wyświetlać się na iPhone'ach czy innych zdefiniowanych urządzeniach.

**4. Wymiary ekranu** — poza przeciąganiem i wyklikiwaniem wielkości ekranów, możemy sami podać dowolne jego rozmiary.

**5. Powiększanie** — możemy ustawić sobie powiększenie naszego widoku. Jest to przydatne, jeśli mamy zbyt dużą rozdzielczość, aby zobaczyć całą stronę lub zbyt małą rozdzielczość, aby dostrzec detale.

**6. Obracanie ekranu** — klikając w ten przycisk zamienimy wysokość z szerokością, symulując zmianę orientacji ekranu.

Jeśli masz wątpliwości do powyższego materiału, to - zanim zatwierdzisz - zapytaj na czacie :)

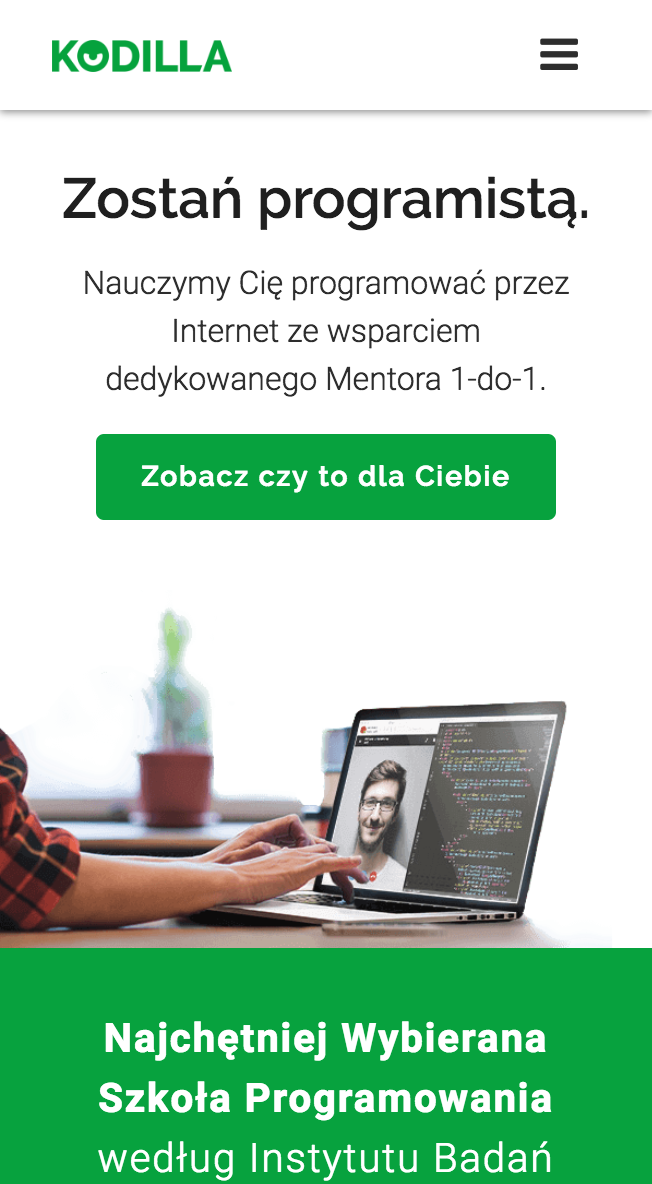
Zapoznałe(a)m się!

**10.2. Mobilne menu**

Jedną z największych bolączek początkujących webdeveloperów jest przystosowanie nawigacji strony do standardów RWD. Często bywa tak, że nie dostaniemy zaprojektowanego widoku mobilnego, a tym bardziej widoku mobilnego menu. Co w takiej sytuacji można zrobić? Zastosować sprawdzone praktyki z innych stron.

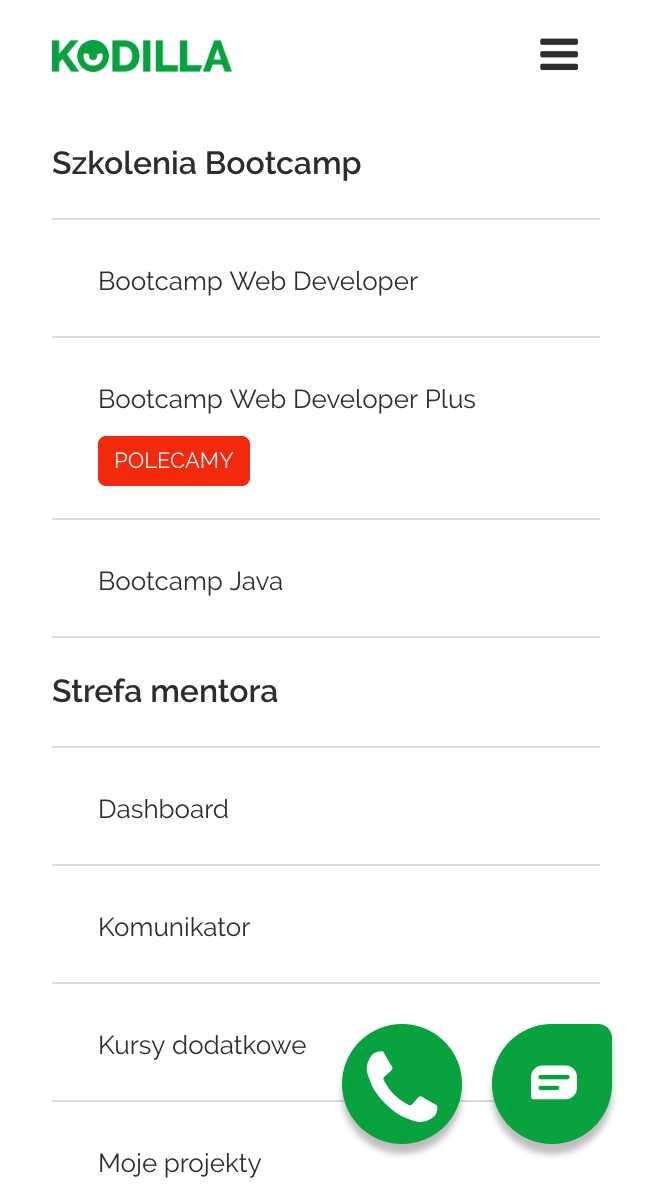
**Popularne rozwiązania**

Kiedy spojrzymy na większość stron, to okaże się, że duża ich część będzie miała tzw. typowy podział, czyli logo i przycisk menu zwany potocznie "hamburgerem".

[](https://uploads.kodilla.com/bootcamp/wdp/wdp10/menu.png)

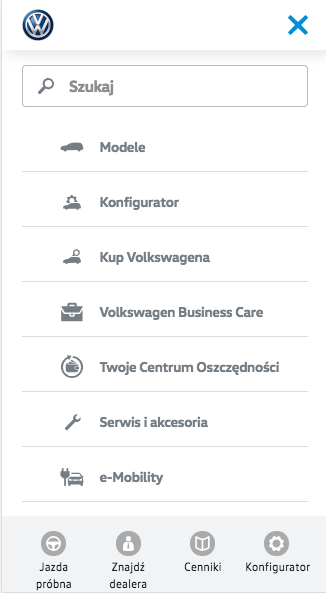
Ale jak powinno wyglądać otwarte menu? Tutaj opcji mamy bardzo dużo, ale do najpopularniejszych należą:

**Rozwijanie w dół z przesunięciem** — polega na tym, że menu rozszerza nam górny panel, a cała treść zjeżdża w dół, pozostając tuż za menu. Tego rozwiązania nie używamy w połączeniu ze *sticky menu*, czyli paskiem menu zawsze widocznym w oknie przeglądarki.

[](https://uploads.kodilla.com/bootcamp/wdp/wdp10/menu-opuszczone.png)

**Rozwijanie w dół z nakładką** — podobnie jak wyżej. Różnica polega na tym, że menu przykrywa częściowo treść — to rozwiązanie jest najczęściej stosowane przy małej liczbie elementów w menu.

**Nakładane menu** — otwarte menu przykrywa nam cały ekran i wygląda tak, jakbyśmy przeszli do innej podstrony. To menu może zawierać więcej elementów, ponieważ może być przewijane.

[](https://uploads.kodilla.com/bootcamp/wdp/wdp10/menu-nakladane.png)

**Offcanvas** — bardzo efektowna opcja, polegająca na wypchnięciu menu od boku, przy czym treść również jest przesuwana. Warto obejrzeć różne [przykłady offcanvas menu](https://tympanus.net/Development/SidebarTransitions/).

**Budowa menu**

Budując menu powinniśmy zastanowić się nad tym, jak bardzo jego układ i wygląd mają pasować do siebie na desktopie oraz urządzeniach mobilnych. Innymi słowy — czy tym samym HTMLem z różnymi stylami będziemy w stanie zbudować menu w dwóch wersjach. W zależności od sytuacji powinniśmy wybrać odpowiednie rozwiązanie:

**1. Podobna struktura** — jeśli ilość elementów, z których ma składać się nasze menu, jest taka sama na desktopie jak i na telefonach, to powinniśmy zrobić wyłącznie jeden kod HTML, a tylko i wyłącznie stylami zróżnicować ich wygląd.

**2. Różne struktury** — jeśli chcemy, aby nasze menu mobilne miało więcej elementów, możemy albo stworzyć rozbudowaną strukturę HTML, na desktopie ukrywać niektóre elementy, a na telefonie pokazywać wszystkie, LUB wydzielić menu do osobnego kontenera i mieć w HTMLu dwa niezależne elementy menu.

**Sterowanie menu**

Mimo że podstawy JavaScriptu omówiliśmy sobie dopiero niedawno i prawdopodobnie jest to jeszcze dla Ciebie czarna magia, to wbrew pozorom sterowanie menu nie wymaga od Ciebie niczego nowego.

Omówmy sobie założenia działania menu. To, jak ma ono wyglądać na desktopie, tabletach i telefonie, jesteś w stanie zrobić za pomocą CSS. Jedyne, co musisz zrobić, to sterować menu tak, żeby wiedzieć, czy ma być ono widoczne, czy nie. W tym celu warto założyć sobie, że sterowanie będzie polegać na dodawaniu klasy, jeśli menu ma być widoczne, oraz jej usunięciu, jeśli będziemy chcieli je ukrywać. Warto tutaj pamiętać, że ta reguła powinna działać tylko na szerokościach, kiedy mamy mobilne menu, czyli istnienie klasy nie powinno mieć wpływu na wyświetlanie się menu na desktopie. To również jesteśmy w stanie przygotować sobie w CSS.

Kiedy mamy już przygotowane wszystkie style związane z menu, przechodzimy do oskryptowania jego działania.

W pierwszej kolejności napiszmy sobie uniwersalną funkcję, która będzie pokazywać i ukrywać menu:

function toggleMenu(visible) {

document.querySelector('.menu').classList.toggle('show', visible)

}

Po wywołaniu toggleMenu(true), do obiektu z klasą menu zostanie dodana klasa show, a po wywołaniu toggleMenu(false) zostanie ona usunięta. Dodatkowo jeśli użyjemy tej funkcji bez argumentu toggleMenu(), to w zależności od tego, czy klasa show już istnieje czy nie, to zostanie dodana lub usunięta.

Później wystarczy ją podczepić do wybranych elementów za pomocą:

document.querySelector('.hamburger').addEventListener('click', function(e) {

e.preventDefault();

toggleMenu()

});

Jak widać, stworzenie sterowania menu może być całkiem łatwe.

Jeśli masz wątpliwości do powyższego materiału, to - zanim zatwierdzisz - zapytaj na czacie :)

Zapoznałe(a)m się!

**10.3. Responsywne tabele i wykresy**

Kolejnym z problemów, na który możesz natrafić, jest problem z wyświetlaniem długich treści, których nie można zwinąć lub zmniejszyć, takich jak wykresy czy tabele. Jeśli tabela jest wąska, to wystarczy dodać jej width: 100% lub skorzystać z przestawnych tabel, które mogą wyglądać jak w [tym przykładzie](https://codepen.io/SitePoint/pen/azeYqx).

Ale jeśli mamy dużo kolumn lub jeśli potrzebujemy pokazać szeroki obrazek, który na telefonie może być nieczytelny, wtedy powinniśmy skorzystać z tzw. scroll wrappera.

**Scrollowany wrapper**

Scrollowany wrapper to nic innego jak element, który zawiera w sobie większy element i jest możliwy do scrollowania w poziomie. Podobne rozwiązanie można zauważyć również w bootstrapie. W jaki sposób zrobić takie rozwiązanie?

Zacznijmy od stworzenia tabelki:

<table>

...

</table>

Następnie owińmy ją divem, który będzie naszym scroll wrapperem i dodajmy mu taką oto klasę:

<div class="scroll-wrapper">

<table>

...

</table>

</div>

A później dodajmy style:

.scroll-wrapper {

display: block;

width: 100%;

overflow-x: auto;

}

.scroll-wrapper table {

width: 100%;

max-width: 100%;

}

I dzięki temu, kiedy będzie trzeba, to tabelka będzie możliwa do scrollowania.

W podobny sposób możemy postąpić z obrazkami wykresów, tyle, że trzeba pamiętać, że obrazek musi mieć wtedy width: auto;.

Jeśli masz wątpliwości do powyższego materiału, to - zanim zatwierdzisz - zapytaj na czacie :)

Zapoznałe(a)m się!

**10.4. Responsywny grid**

Kolejną rzeczą, którą warto lepiej poznać, jest praca z responsywnym gridem. Na początku kursu powiedzieliśmy Ci, czym jest grid, i w jaki sposób go wykorzystać. Teraz czas na pewną dobrą praktykę związaną z responsywnym gridem, czyli takim, który możemy definiować dla różnych szerokości ekranu.

Zapewne pamiętasz, jak używaliśmy Bootstrapa, gdzie mieliśmy możliwość określić szerokość kolumny osobno dla wielkości small, medium i large, np. col-sm-12 col-md-6 col-lg-3. Dzięki takim klasom kolumna na małych ekranach zajmuje 100% miejsca, na tabletach połowę, a na desktopie jedną czwartą.

Przygotujmy podobny grid, ale tym razem wykorzystamy do tego celu Sassa i spróbujemy go wygenerować dynamicznie.

Zacznijmy od przygotowania sobie założeń dla gridu:

* budowa bazująca na flexboksie,
* podejście boxed i fluid,
* 12 kolumn,
* podejście mobile-first, czyli domyślnie definiujemy dla najmniejszych urządzeń, a tylko większe będziemy rozszerzać,
* 4 zakresy: do 767px, od 768px, od 992px, od 1200px. , Jak pamiętasz w poprzednich modułach używaliśmy gridu opartego o floaty. Pokażemy Ci, że można zbudować grid na różne sposoby. Warto wiedzieć, że Bootstrap w wersji 4 zaczął korzystać z flexboksa, dlatego wspólnie zrobimy podobne rozwiązanie.

Zacznijmy od wyseparowania klas odpowiedzialnych za budowę boxed i fluid. Podejście boxed polega na tym, że główny kontener ma swoją określoną maksymalną szerokość, natomiast fluid ma pełną dostępną szerokość.

\* {

box-sizing: border-box;

}

.container {

width: 100%;

max-width: 1170px;

margin: 0 auto;

padding-left: 15px;

padding-right: 15px;

}

.container-fluid {

width: 100%;

padding-left: 15px;

padding-right: 15px;

}

**Row**

Teraz przejdziemy do stworzenia bazy dla naszego grida, czyli *row*.

.row {

display: flex;

flex-wrap: wrap;

}

**Kolumny**

A następnie przygotujemy sobie wzór styli kolumn.

.col-sm-1 {

flex: 0 0 8.33333%;

max-width: 8.33333%;

}

.col-sm-2 {

flex: 0 0 16.66667%;

max-width: 16.66667%;

}

.col-sm-3 {

flex: 0 0 25%;

max-width: 25%;

}

...

Spróbujmy z tego zrobić pętlę. Jeśli chcesz lepiej zrozumieć działanie pętli, zapoznaj się z [dokumentacją](https://sass-lang.com/documentation/file.SASS_REFERENCE.html#for).

@for $i from 1 through 12 {

.col-sm-#{$i} {

$width: 100% / 12 \* $i;

flex: 0 0 $width;

max-width: $width;

}

}

Następnie stworzymy mixin, który na podstawie prefiksu dla danej szerokości zbuduje nam listę klas:

@mixin grid($prefix) {

@for $i from 1 through 12 {

.col-#{$prefix}-#{$i} {

$width: 100% / 12 \* $i;

flex: 0 0 $width;

max-width: $width;

}

}

}

**Implementacja grida w media queries**

Na końcu dodamy media queries:

@media(max-width: 767px) {

@include grid('sm');

}

@media(min-width: 768px) {

@include grid('md');

}

@media(min-width: 992px) {

@include grid('lg');

}

@media(min-width: 1200px) {

@include grid('xl');

}

W ten sposób nawet jeśli nie zadeklarujemy, jak ma wyglądać kolumna na desktopie, ale zadeklarujemy, jak ma wyglądać na telefonie, to odziedziczy ona szerokość niższego rzędu. Dzięki temu nasze rozwiązanie spełni warunek mobile-first.

Teraz bez większych przeszkód możesz samodzielnie rozszerzyć grida, lub po prostu z niego skorzystać do zbudowania responsywnego układu.

Jeśli masz wątpliwości do powyższego materiału, to - zanim zatwierdzisz - zapytaj na czacie :)

Zapoznałe(a)m się!

**10.5. Retina i problemy z obrazkami**

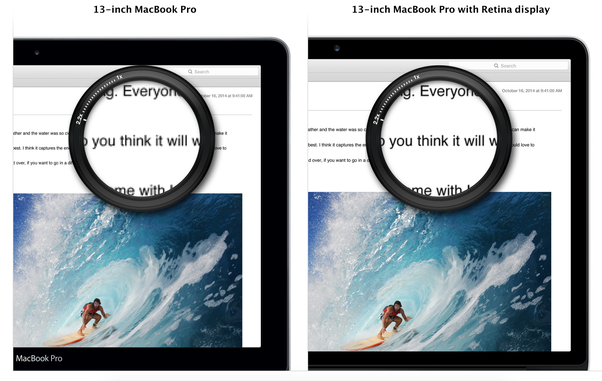
Ostatnią rzeczą, którą omówimy w tym tygodniu, będzie sprawa obrazków oraz sposobów na ich osadzenie na stronie. Temat może wydawać się dość trywialny, jednak głębsze przypatrzenie się problemom, które są z nim związane, może diametralnie zmienić Twoje podejście do niego.

**Gęstość pikseli i Retina**

Zacznijmy od wyjaśnienia sobie pewnego pojęcia: **gęstość ekranu**. Jaki rozmiar ma 1 piksel? Ile to cm? Na to pytanie nie ma jednoznacznej odpowiedzi, ponieważ wszystko zależy od gęstości ekranu. Monitory są różnych wielkości: 20, 24, 32 cale, a telewizory mogą być jeszcze większe, typu 42, 50 czy 60 cali. Ale w zależności od ich matrycy mogą mieć podobne rozdzielczości ekranu np. FullHD(1920px X 1080px), niezależnie od ich fizycznej wielkości. I wtedy dochodzimy do określenia **gęstości ekranu**, która mówi o liczbie pikseli bądź punktów na jednostkę miary. Takimi jednostkami gęstości ekranu mogą być DPI (Dots Per Inch) czy PPI (Points Per Ich). Rzadziej trafiamy na DPC (Dots Per Cm) czy inne odwołujące się do innych jednostek miary.

Dlatego duże ekrany mają tą gęstość mniejszą, np. 72 czy 96 DPI, a mniejsze urządzenia typu telefon mają nawet po 300 DPI. A na co wpływa gęstość pikseli? Na jakość wyświetlanych grafik. Im więcej pikseli na małej powierzchni (czyli im większa gęstość), tym wyraźniejszy obraz możemy otrzymać.

Czym zatem jest Retina? Jest to opatentowany przez Apple rodzaj ekranów, gdzie gęstość wynosi zazwyczaj powyżej 330 DPI. W praktyce wygląda to tak, że jeśli ekran ma rozdzielność 1920px i takiej wielkości będą wyświetlane obiekty, to fizycznie tę szerokość obsługiwać będzie 2 razy więcej pikseli. Dzięki temu obraz jest o wiele bardziej czytelny.

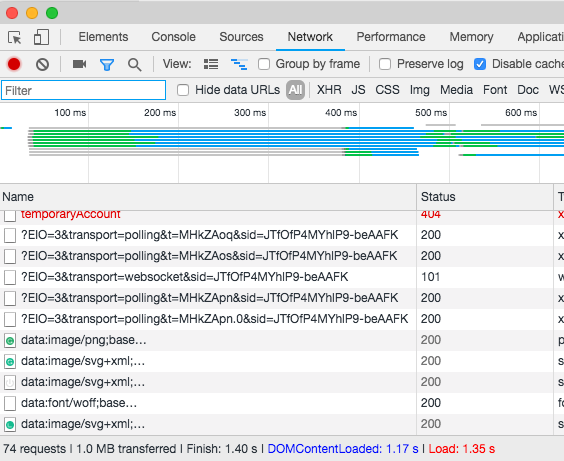


Ceną za lepszą jakość obrazu jest to, że wszystkie elementy statyczne (np. obrazki) powinny mieć 2 razy większe źródła. Stąd mówimy o "Retina Ready" czyli stronach, które wyświetlają się równie dobrze na normalnych urządzeniach oraz na tych z ekranami Retina.

**Obrazki a Web Performance**

Drugi problem, który może pojawić się z obrazkami na naszej stronie, to ich wpływ na Web Performance, czyli na osiągi strony. Za tym pojęciem kryje się m.in. czas ładowania strony, włączając w to wszystkie jej elementy (w tym obrazki), czy to, jak płynnie strona działa. Można pomyśleć, że na prędkość działania komputera użytkownika nie mamy wpływu. I to jest prawda, ale mamy wpływ na to, żeby strona była lekka, szybko się ładowała i łatwo się po niej nawigowało.

Tutaj istotym jest przede wszystkim ograniczenie wagi obrazków. Jak sprawdzić, ile nasza strona waży dla użytkownika? Otwórz narzędzia developerskie w przeglądarce i przejdź do zakładki Performence: na dole znajdziesz informację o wadze wszystkich elementów i czasie ładowania strony.



Naszym zadaniem jest zatem osadzanie tylko obrazków w rozmiarach, które faktycznie są używane, oraz ich kompresja. Dlatego jeśli znajdziesz jakieś fantastyczne zdjęcie, to warto najpierw zobaczyć, jaką ono ma wagę. W różnego rodzaju stockach (bankach zdjęć) znajdziemy przepiękne, wysokiej rozdzielczości zdjęcia. Tylko jeśli na stronie zamieścimy 5 zdjęć, z których każde waży po 5-15 MB, to strona może nie tylko się długo ładować, ale i wolno chodzić.

Grzechem byłoby nie wspomnieć też o użytkownikach mobilnych, którzy często mają ograniczoną prędkość transmisji danych, a co więcej, urządzenia mobilne gorzej radzą sobie z przetwarzaniem dużych plików. Sprawi to, że samo przewijanie strony już będzie problemem — o ile strona się załaduje do końca.

Dlatego przed użyciem jakiegokolwiek obrazka z sieci warto sobie zadać pytania:

* Czy jego rozmiary są odpowiednie do osadzenia na stronę?
* Czy jego waga nie jest zbyt duża?

**PORADA:** Czasem zmniejszenie obrazka może tylko nieznacznie zmniejszyć jego wagę, lub po prostu potrzebujemy dużego obrazka. W takich przypadkach ważna jest jego kompresja. Jednym z serwisów, który umożliwia darmową kompresję obrazków, jest [TinyPNG](https://tinypng.com/" \t "_blank), który potrafi zmniejszyć wagę obrazków nawet o 80% bez utraty ich jakości!

**HTML5 srcset**

Skoro już wiemy, **CO** mamy zrobić, to warto jeszcze odpowiedzieć na pytanie, **JAK** to wdrożyć. Kiedy wrócisz do modułu o responsive, znajdziesz fragment o działaniu srcset z HTML5. Tym razem rozwiniemy nieco temat, ponieważ jest prosta metoda, aby przekazać różnym rodzajom urządzeń różne obrazki. Innymi słowy, możemy przygotować sobie zestaw obrazków dla różnych wielkości ekranu i serwować je w zależności od dostępnego viewportu.

<img

src="/images/foo.png"

alt="bar"

srcset="/images/foo-medium.png 1024w,

/images/foo-large.png 2048w,

/images/foo.png 800w"

/>

Jest jeszcze jedna na to metoda, polegająca na użyciu tagu picture, któremu przekazujemy źródła plików wraz z regułami. W tym przypadku użyte zostanie pierwsze źródło, którego reguły będą pasować.

<picture>

<source media="(min-width: 1024px)" srcset="foo-large.jpg 1024w, foo-medium.jpg 640w, foo-small.jpg 320w" sizes="50vw" />

<source srcset="foo@2x.jpg 2x, foo.jpg 1x" />

<img src="foo.jpg" alt="Bar" />

</picture>

**Obrazki SVG**

Kolejną metodą wyświetlania obrazków, szczególnie tych rysowanych czy też ikon są pliki SVG, czyli pliki wektorowe. Są one opisane matematycznie, dzięki czemu ich waga wynosi tyle, ile wynosi opisanie ich obiektów. Jakie są zalety plików wektorowych? Nieważne, czy będziemy chcieli wyświetlić 30x30px czy 4000x4000px, obrazek będzie zawsze idealnie równy — nie ma szans na pikselozę.

Osadzać SVG można na dwa sposoby. Możemy klasycznie podać źródło obrazka w atrybucie src:

<img src="file.svg">

lub po po prostu wkleić zawartość pliku SVG do kodu HTML:

<div>

<svg width="100" height="100">

<circle cx="50" cy="50" r="40" stroke="green" stroke-width="4" fill="yellow" />

</svg>

</div>

**Manipulacja SVG przez CSS**

Jeśli osadzimy SVG jako HTML, to zyskujemy możliwość manipulowania nim za pomocą CSS.

Powyższy przykład generuje nam żółte kółko o rozmiarze 100x100px z zieloną obramówką. Jeśli chcielibyśmy zmienić jego kolor na czerwony, to musimy odwołać się do niego tak samo jak, do elementów HTML, tylko z użyciem właściwości zgodnych z SVG. W tym przykładzie zamiast background-color użyjemy fill:

svg circle {

fill: red;

}

**Czcionki z ikonami**

Ostatnią rzeczą dotyczącą SVG i obrazków są czcionki, które zamiast znaków mają ikony. Przykładem takiej czcionki jest [Font Awesome](https://fontawesome.com/). Zalet używania takich zestawów czcionek jest wiele:

* łatwe użycie,
* mała waga,
* mniejsza liczba rzeczy do pobrania,
* prosta manipulacja.

Używając takich czcionek, możemy szybko zmieniać wielkość czy kolor ikonek. O czym warto pamiętać to to, że takie ikonki muszą być **jednokolorowe**, tzn. każda ikona będzie miała tylko jeden kolor.

Jeśli chcesz stworzyć własny zestaw ikon na podstawie swoich plików SVG lub skorzystać z już stworzonych ikon, możesz taką czcionkę sobie wygenerować przy pomocy [Fontello](http://fontello.com/" \t "_blank).