4. Responsywny layout!

Wyzwania:

- Stworzysz stronę działającą na wszystkich komórkach i tabletach.
- Nauczysz się testowania utworzonych przez siebie stron mobilnych.
- Zoptymalizujesz strony dla urządzeń mobilnych.

Ocena kursu

Stale podnosimy jakość naszych szkoleń. W związku z tym bylibyśmy wdzięczni za wypełnienie krótkiej ankiety. Zajmie Ci ona dosłownie chwilę.

Ocena bootcampa WebDeveloper

*Wymagane

W której edycji bierzesz udział? *

Wybierz

Jaka jest Twoja dotychczasowa ocena bootcampa? *

	1	2	3	4	5	6	7	8	9	10	
Mega słaby	○	○	○	○	○	○	○	○	○	○	Świetny!

Twoja opinia na temat kursu (opcjonalnie)

Twoja odpowiedź

Imię i nazwisko/lub mail jakim logujesz się do bootcampa * *

Twoja odpowiedź

PRZEŚLIJ

Nigdy nie podawaj w Formularzach Google swoich haseł.

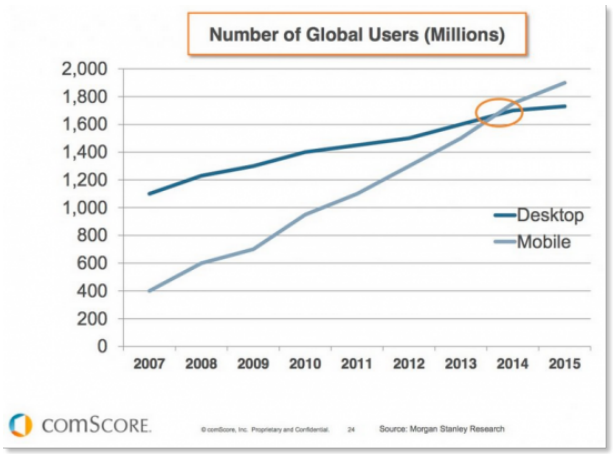
Formularze Google Ten formularz został utworzony w domenie Codemy S.A..

Dziękujemy!

4.1. Z czym to się je?

O tym, że internet przeglądamy nie tylko na komputerze, ale również na tablecie czy telefonie komórkowym, wspominaliśmy już niejednokrotnie.

Statystyki wskazują, że w niektórych krajach, np. w USA przekroczono już punkt, w którym udział ruchu użytkowników mobilnych jest większy niż desktopowych.



Dlatego też obecnie zadbanie o **prawidłowe działanie strony internetowej na urządzeniach mobilnych to być albo nie być dla wielu firm**. Jeśli klient zobaczy na swoim smartfonie niedostosowany ("rozwalony") sklep internetowy, to wpłynie to negatywnie na zaufanie do właściciela witryny (niska jakość) i raczej niczego wtedy nie kupi :)

AWD, RWD...

Z pewnością wiele razy trafiałeś na strony, których adres rozpoczyna się od **m**. lub **mobile**. (lub inne konwencje), gdy odwiedzasz je na urządzeniu mobilnym. Taki oddzielny adres jest stosowany właśnie w przypadku urządzeń przenośnych.

Gdy wpisujesz w telefonie np. adres wp.pl, ta strona za pomocą kodu JavaScript rozpoznaje typ urządzenia (np. smartfon) i przekierowuje Cię na specjalny adres, w tym wypadku http://www.wp.pl/mini.html. Jak widać, taka wersja jest lepiej dostosowana do mniejszych wyświetlaczy.

Opisane powyżej podejście to tzw. **AWD**, czyli **A**daptive **W**eb **D**esign. W takim wypadku strona mobilna (smartfony, tablety) i desktopowa (komputery stacjonarne, laptopy) to dwa oddzielnie realizowane projekty. Inny szablon front-endowy buduje się dla smartfona, a inny dla laptopa.

Takie podejście sprawia jednak, że trzeba napisać znacznie więcej kodu, by je stworzyć. Także wszelkie zmiany wyglądu, np. na stronie głównej, wymagają modyfikacji nie w jednym, ale w dwóch szablonach. Takie podejście jest stosowane, kiedy:

- grafik/projektant z jakiegoś powodu założył, że wersja mobilna będzie się bardzo różnić od desktopowej i łatwiej jest wtedy stworzyć oddzielny szablon,
- witryna ma ogromny ruch internetowy i przeznaczony dla mobile szablon jest zoptymalizowany pod kątem wydajności i szybkości ładowania versus obciążenie serwerów.

A propos pierwszej wymienionej przyczyny — trend jest taki, aby mocno ujednolicać wizualnie widok desktopowy i mobilny. Ma to głównie zaletę "brandingową" — niezależnie od urządzenia, z którego korzysta, internauta wie, na jakiej jest stronie.

Zupełnie odmiennym podejściem w zakresie dostosowywania witryny pod wiele urządzeń jest **RWD**, czyli **R**esponsive **W**eb **D**esign. Opiera się ono na tworzeniu jednego szablonu pod wszystkie urządzenia wraz z dodatkowymi regułami w CSS, które mówią o tym, jak ma wyglądać strona na poszczególnych rozdzielczościach.

Oznacza to, że niezależnie od tego, czy oglądasz stronę na telefonie, tablecie, komputerze, czy też telewizorze, Twoja przeglądarka otrzymuje zawsze te same pliki HTML oraz CSS i używa odpowiednich reguł CSS w zależności od rozmiarów ekranu. Np. poniższa strona to jeden szablon HTML/CSS, który dostosowuje się do różnych urządzeń.

Web-mobile a aplikacje mobilne

Istnieją dwie główne drogi dotarcia do użytkownika mobilnego. Pierwsza to aplikacje mobilne, tzw. natywne. Są to po prostu programy, które instalujesz np. z Google Play. Tworzy się je oddzielnie na każdą platformę (system) mobilną:

- Google Android — wersje od 4.0 do 8.x,
- iOS (iPhone OS) — obecnie głównie wersje 9+,
- Windows Mobile, Blackberry, Symbian — rzadko; głównie obsługuje się dwa powyższe systemy (link do statystyk).

Sposób ten ma oczywiście swoje niezaprzeczalne zalety, gdyż daje pełne możliwości wykorzystania telefonu, których nie daje nam przeglądarka. **W aplikacjach mobilnych możemy skorzystać z takich funkcji, jak np. aparat, GPS czy żyroskop**. Zazwyczaj wymaga to każdorazowego tworzenia, a następnie aktualizacji aplikacji z myślą o konkretnej platformie. Nie tylko oznacza to dużo więcej pracy, ale też wymaga dużo większej wiedzy na temat specyfiki danego systemu. Do tworzenia takich aplikacji najczęściej wykorzystuje się języki Java (dla Androida, nie mylić z JavaScriptem) i m.in. Swift (dla iOS).

Drugi typ rozwiązań mobilnych to responsywne strony internetowe (tzw. web-mobile). Są one podobne do tych tradycyjnych, jednak skonstruowane w taki sposób, aby uwzględniać charakterystyczne cechy urządzeń mobilnych (mały rozmiar ekranu, dotykowa obsługa).

Chociaż zewnętrznie często są do siebie podobne, to aplikacje i strony mobilne znacznie się różnią. Te pierwsze oferują więcej funkcjonalności (dostępnych także offline) i przeważnie są bardziej rozbudowane (np. gry).

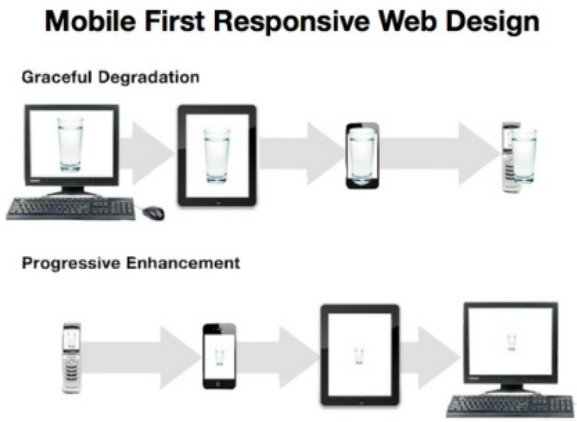
Strony mobilne są prostsze, nie działają bez dostępu do internetu, ale są bardziej uniwersalne. Działają na wszystkich urządzeniach i systemach mobilnych, na których jest przeglądarka internetowa. Ich zaletą jest też to, że działają niezwłocznie, bez konieczności pobierania i instalowania czegokolwiek.

Ponadto strony internetowe posiadają wspólny kod dla wszystkich platform, a aplikacje na iOS oraz Androida lub inny system tworzone są w zupełnie inny sposób.

Na marginesie dodać można, że różnice między aplikacjami mobilnymi a web-mobile w ostatnim czasie nieco się zacierają, gdyż powstają coraz bardziej skuteczne techniki tworzenia aplikacji natywnych z wykorzystaniem HTML/CSS/JS (tzw. hybrid mobile apps).

Dwie drogi w RWD

W tym kursie zajmujemy się web-mobile z użyciem RWD. W zakresie tworzenia stron responsywnych wyróżnia się dwa podejścia.



Graceful degradation

Czyli przypadek, w którym zaczynamy od wersji strony dla ekranów o dużej rozdzielczości (desktop) i dopiero potem dostosowujemy ją do ekranów mniejszych.

Polega to najczęściej na upraszczaniu zbudowanego layoutu pod mobile i zmianie ułożenia poszczególnych elementów tak, aby zapewnić im odpowiednie wyświetlanie na mniejszych ekranach. Raczej powinno się unikać ograniczania funkcjonalności strony, ale czasami nie ma wyjścia :) Nie wszystko da się "upakować" w dziesięciokrotnie mniejszy ekran.

Progressive enhancement

Znane także jako *mobile first*. Polega na stworzeniu najpierw wersji dostosowanej do rozdzielczości właściwych urządzeniom mobilnym, a następnie rozbudowywanie jej i przystosowanie do większych ekranów. Zgodnie z tą metodą tworzymy najpierw prostą wersję mobilną, którą następnie rozbudowujemy dla ekranów szerszych. W tym podejściu unikamy sytuacji, w której tworzymy spektakularny produkt wykorzystujący wiele zaawansowanych technologii w wersjach desktopowych tylko po to, żeby odkryć, jak wiele z nich nie działa na przeglądarkach mobilnych.

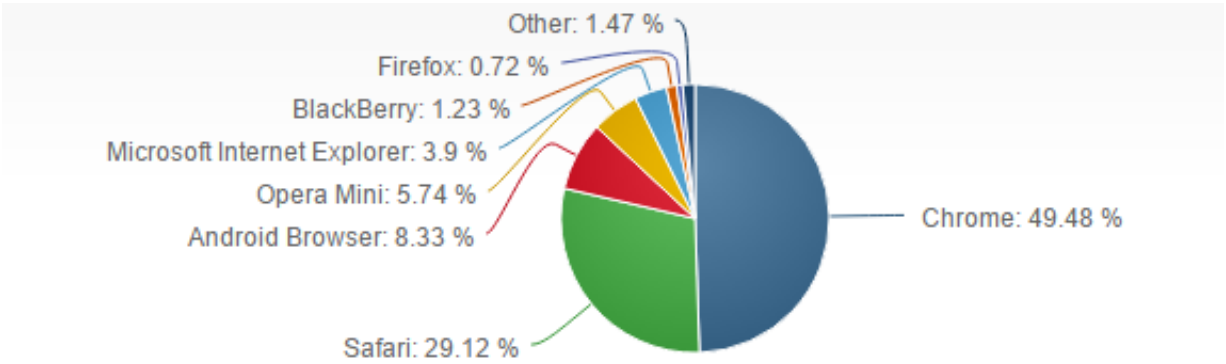
Które podejście wybrać? Nie jest jednak tak, że jedno podejście jest w naturalny sposób lepsze czy gorsze. W praktyce ciągle trwa wśród front-endowców gorąca debata o wyższości jednego sposobu nad drugim. My — po początkowym zachwycie nad mobile first — skłaniamy się bardziej ku temu, aby zaczynać od desktopów, ale ze świadomością, że potem czeka nas mobile.

Przeglądarki mobilne

Sytuacja na rynku przeglądarek mobilnych szybko się zmienia, dlatego dobrze jest mieć ogólne pojęcie o popularności poszczególnych przeglądarek. Wśród desktopowych królują: Google Chrome, Firefox, Safari, Internet Explorer (Edge) oraz Opera.

Na rynku mobilnym do najpopularniejszych należą "odchudzone" wersje przeglądarek desktopowych, czyli: Chrome, Opera, Safari czy Internet Explorer. Poza tym popularne są również przeglądarki typowe dla systemów mobilnych, jak Android Browser czy UC Browser.

Kwestia ilu użytkowników ma dana przeglądarka mobilna jest istotna z tego względu, że starsze wersje przeglądarek nie wspierają nowoczesnych rozwiązań HTML, CSS oraz JavaScript. Może się więc okazać, że nasza mobilna strona internetowa, która korzysta z takich rozwiązań, nie zadziała u sporej części użytkowników. Dlatego też przy tworzeniu strony mobilnej musimy zastosować metodę podobną do tej, którą omawialiśmy w poprzednim module i rozważyć, jakie przeglądarki będziemy wspierać (poniżej statystyki użycia przeglądarek internetowych).



Aby nie testować strony na pięćdziesięciu przeglądarkach mobilnych, można ogólnie przyjąć obsługę Chrome (prawie to samo, co Firefox i Android Browser) i Safari. Z racji niskiego udziału w rynku Windows Phone na rynku można odpuścić Internet Explorera, chyba że trafisz na klienta z takim telefonem :) Opera mini jest także używana na starych telefonach, ale robienie pod nią RWD jest praktycznie niemożliwe. **Postaw na razie na Chrome i Safari :)**

Budując stronę z RWD, trzeba też wziąć pod uwagę specyfikę urządzeń mobilnych. Użytkownicy korzystający z komputerów desktopowych mają do dyspozycji klawiaturę i mysz, a w przypadku tabletów i smartfonów interakcja z użytkownikiem odbywa się za pomocą ekranu dotykowego.

W efekcie musimy wziąć pod uwagę specyfikę tej interakcji przy tworzeniu interfejsu użytkownika. **Elementy nawigacji powinny być dostatecznie duże**, tak aby zapewnić ich łatwą klikalność na ekranie dotykowym. Jednak nie powinny one przysłaniać pozostałej treści. Trzeba też pamiętać, że na urządzeniach mobilnych nie ma takiego stanu jak `:hover`, co jest logicznie zrozumiałe (nie ma stanu palca nad elementem).

Dobrze jest też zastanowić się, czy użytkownicy mobilni nie będą korzystać ze strony w poszukiwaniu nieco innych informacji niż desktopowi. W praktyce może to oznaczać, wizualna hierarchia poszczególnych treści na urządzeniach mobilnych będzie się nie różnić.

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

✔Zapoznałe(a)m się!

4.2. Jak to się robi?

Responsywne strony budowane są przy użyciu trzech głównych metod:

- regułach media queries,
- layoucie dostosowującym się do szerokości okna przeglądarki,
- responsywnych obrazkach.

Media queries

Jest to najczęściej wykorzystywana do budowania stron w podejściu RWD reguła w ramach CSS. **Media queries** umożliwiają dostosowanie sposobu wyświetlania witryny, najczęściej do rozmiarów ekranu. Są to dodawane w pliku CSS reguły sprawiające, że niektóre style zostają uruchomione tylko dla urządzeń określonego typu lub o określonej rozdzielczości.

Dzięki temu możemy sprawić, że na komputerze strona będzie oferować więcej treści bądź rozszerzoną funkcjonalność w stosunku do wersji mobilnej. Za pomocą tych reguł można także sprawić, że dany element inaczej wyświetli się w mniejszej rozdzielczości (będzie np. mniejszy).

Przykład użycia

Zobaczmy to na przykładzie.W osobnym oknie przeglądarki otwórz stronę https://www.smashingmagazine.com/. Pobaw się nią, zmniejszając i zwiększając stopniowo szerokość przeglądarki. Obserwuj, jak zachowują się elementy na stronie (szczególnie linki w menu oraz zdjęcia). Postaraj się znaleźć rozmiary, w których strona zasadniczo zmienia swój wygląd (np. z układu z dwoma lub trzema kolumnami przechodzi w jedną).

Jeśli otwierasz stronę w Chrome, włącz także Developer Tools (F12, aby był na dole). Wtedy zmieniając rozmiar przeglądarki, w prawym górnym rogu zobaczysz aktualną rozdzielczość.

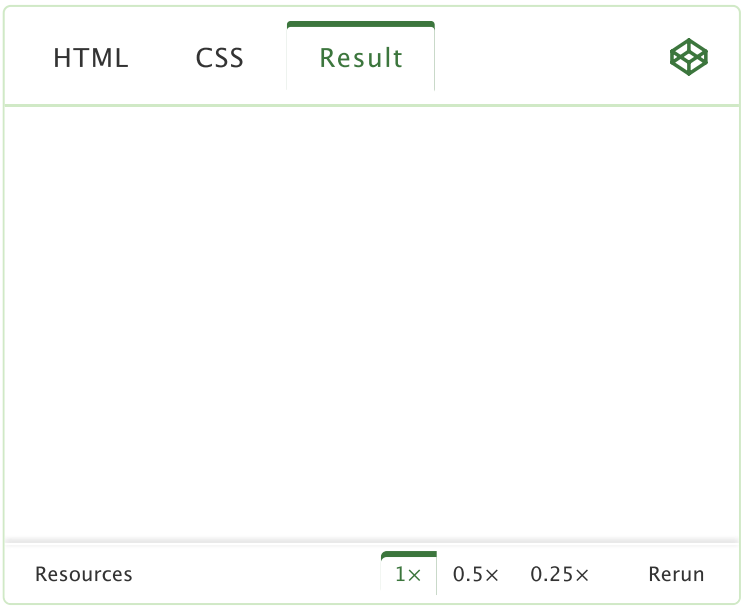
Breakpointy

Momenty zmiany układu strony to tak zwane **breakpointy**. Projektując czy tnąc stronę, określamy takie breakpointy. Potem określamy reguły z wartościami pikselowymi tych breakpointów w CSS (przy pomocy media queries). Tworzą się swego rodzaju przedziały (pomiędzy breakpointami), w obrębie których strona przyjmuje dany układ/wygląd).

Poniżej podajemy składnię reguły media queries (w nawiasach zwykłych nigdy nie dajemy średnika, a nawiasy wąsy, jak widać, są podwójnie zagnieżdżone):

```
@media (max-width: 600px) {
/* Niżej dodajemy reguły CSS, które zadziałają tylko  dla ekranów o m
    .class {
    color: #000;
    }
}
```

Działanie powyższego przykładu możesz zobaczyć w prosty sposób: zmień szerokość podglądu i zobacz, jak zmienia się kolor sekcji.



Rodzaje obsługiwanych urządzeń

W regułach `@media` możemy także określić typ urządzenia, które chcemy obrać za c
Najpopularniejsze to **all**, **screen**, **tv** czy **print**. Istnieją też bardziej egzotyczne rodzaje
braille — dla urządzeń dla niewidomych czy **3d-glasses** dla okularów 3d.

Możesz nic nie wpisywać (tak robi się najczęściej) i wtedy dana reguła zostanie zastosowana do wszystkich urządzeń, w tym screen, których rozdzielczość najbardziej nas interesuje.

Operatory logiczne

Poza typami urządzeń możemy także używać **operatorów logicznych** takich jak **and**, **not**, **only**. Dzięki temu możemy stworzyć rozbudowane wyrażenia, w których precyzyjnie określimy, jaki typ urządzenia chcemy objąć regułą.

Poniżej przykład wykorzystania operatorów logicznych do określenia typu urządzenia i jego rodzaju:

```
@media screen and (min-width: 800px) and (max-width: 1024px) {...}
```

Reguła ta wskazuje na **ekran urządzenia** i zostanie zastosowana w przypadku, gdy jego szerokość będzie się mieściła w przedziale od 800px (minimalna szerokość) do 1024px (maksymalna szerokość). Poza tym przedziałem zdefiniowane w regule style (w {...}) nie zadziałają.

Warunki

W regule, w zwykłych nawiasach, określa się następujące właściwości i ich wartości:

1. **Min-width i max-width** — określające szerokość urządzenia (a dokładnie viewportu, o czym przeczytasz poniżej).
2. **Orientation** — orientacja pionowa lub pozioma urządzenia. Dostępne wartości to portrait (wysokość ekranu jest większa niż szerokość) oraz landscape (na odwrót). Ma to szczególnie zastosowanie w dostosowywaniu strony do wyświetlacza tabletu lub telefonu, które mogą być trzymane przez użytkownika zarówno pionowo jak i poziomo.
3. **Resolution** — przy jej pomocy możemy określić reguły CSS, które będą miały zastosowanie tylko, jeśli "gęstość upakowania pikseli" w urządzeniu będzie miała określoną wartość (podanej w dpi, czyli plamce na cal lub dpcm — plamce na centymetr).

W 99% przypadków używa się właściwości `max-width` i `min-width`, gdzie określa się dodatkowe działanie stylów w danych przedziałach rozdzielczości poziomej (pomiędzy breakpointami).

Viewport

Przy okazji RWD, należy wspomnieć o viewport. Jest to ta część strony, która jest widoczna w danym momencie na ekranie użytkownika. Im mniejsze urządzenie, tym mniejsza widoczna przestrzeń.

Aby strona była responsywna, to oprócz odpowiednich reguł media queries warto najpierw określić odpowiedni rozmiar viewport. Ma to głównie na celu to, aby strona wgrana na urządzenie mobilne nie była wyjściowo powiększona (zoom), ale dostosowana do wielkości urządzenia (100% szerokości ekranu tego urządzenia).

Najczęściej używa się rzadko modyfikowanego meta-tagu w HTML (w sekcji head) mającego postać (można ją uznać za standard):

```
<meta name="viewport" content="width=device-width, initial-scale=1.0"
```

Pierwsza część (`content="width=device-width`) ustawia szerokość strony tak, aby odpowiadała szerokości urządzenia. Druga (`initial-scale=1.0`) ustawia początkowy poziom powiększenia (zoom) na urządzeniach mobilnych.

Co modyfikujemy, używając RWD?

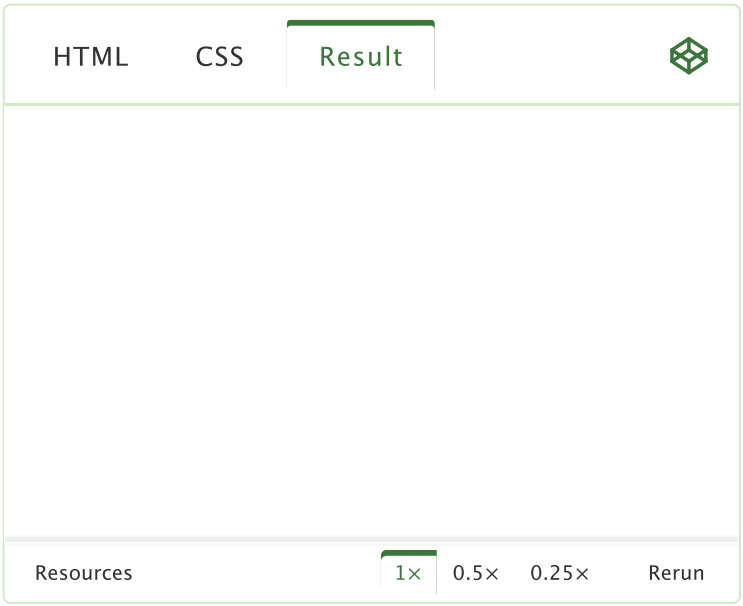
RWD wykorzystuje się głównie do odpowiedniego wyświetlania układu strony na różnych urządzeniach. Do roku 2005 do budowania layoutu wykorzystywano tabele HTML. Często zagnieżdżano takie tabele jedną w drugiej i tworzono coś w rodzaju siatki, w którą następnie wkładano treść.

Jednak tabela w HTML ma jedną zasadniczą wadę — **słabo nadaje się do rozwiązań typu RWD.** Modyfikowanie w locie takiej tabelarycznej siatki układu pod małe rozdzielczości jest praktycznie niemożliwa.

RWD a grid

Grid (budowaliśmy już takowy) to siatka złożona z wierszy i kolumn, która w płynny sposób może się zmieniać. W ramach różnych rozdzielczości możemy np. z 4-kolumnowego układu strony zrobić 1-kolumnowy. **Jednokolumnowy układ dla mobile to bardzo częsta konwencja w RWD**, bo na wąskim urządzeniu łatwej wtedy czytać treść.

Teraz wejdź w nowym oknie na poniższy link z przykładem (kliknij "Edit on codepen")



Zwróć uwagę, że w powyższym kodzie wykorzystaliśmy zbudowany wcześniej grid, a na końcu dodaliśmy tam **dwa breakpointy** w ramach media queries. Można się zastanawiać, jaki jest sens tworzenia dwóch zestawów klas o identycznych wartościach. Jest on taki, że teraz jeden zestaw klas działa tylko do określonej szerokości urządzenia, a drugi powyżej niej. Dzięki temu możemy określić, że np. element będzie zajmował 50% szerokości na dużym ekranie, a 25% na średnim.

Poniżej ustalonych granic (dla najmniejszych urządzeń) znajduje się ponadto selektor atrybutu ustalający szerokość 100% dla wszystkich kolumn, dzięki czemu cała treść wyświetli się w jednej kolumnie. Dzięki powyższym technikom możemy z łatwością **kontrolować układ naszego layoutu** i dostosowywać go do zmieniającej się szerokości ekranu, na którym wyświetlana jest strona.

RWD a elastyczne jednostki

Poza layoutem, czasami warto też zmienić wartości, w których określaliśmy szerokość elementów, ze stałych (najczęściej są to piksele) na relatywne (wymiary elementów wyrażone w procentach oraz rozmiar fontów w em).

Istnieją także jednostki takie jak `vh` (np. `40vh` oznacza zajęcie przez element 40% wysokości viewportu, ale nie 40% wysokości całej strony) oraz `vw` (analogiczny, ale dotyczący szerokości).

Nie chodzi o to, aby rezygnować z jednostek sztywnych, np. pikseli, a raczej o to, by unikać określania stałej szerokości/wielkości dla wybranych bardzo dużych elementów (jak slogan w headerze), które powinny lepiej się dostosowywać.

Zwróć uwagę na przykład poniżej. Znajdują się tam dwa proste układy, z których jeden ma ustaloną szerokość za pomocą **wartości relatywnych, a drugi stałych**. Sprawdź zachowują się oba układy przy zmianie rozmiaru okna przeglądarki:

https://codepen.io/kodilla/pen/6f8b7e33e388ec4d9654e4e4446c1c4f

Następnie zmieniamy układ wyświetlania elementów naszej strony (jak już pisaliśmy — najczęściej z poziomego na pionowy, z uwagi na wąski ekran urządzeń mobilnych). Możemy to zrobić, odpowiednio manipulując wartością deklaracji z float. W takim wypadku dodajemy float dla dużych ekranów, zaś dla małych — szerokość 100%.
https://codepen.io/kodilla/pen/eb24e22a0cea0bc2df4a72370584d574

Rozmiary czcionek

Kolejnym przykładem dostosowań w RWD jest manipulacja rozmiarem czcionek za pomocą odpowiednich jednostek. Przyjrzyjmy się teraz relatywnym jednostkom rozmiaru, jakie daje nam CSS3 — są to `em` oraz `rem`. Za ich pomocą możemy nadać tekstowi pożądany rozmiar będący wielokrotnością rozmiaru tekstu ich elementu rodzica (`em`) oraz elementu głównego (`rem`), tak jak w przykładzie poniżej:

```
.relative-root {
    font-size: 1.2rem;
}
.relative-parent {
    font-size: 0.8 em;
}
```

W przykładzie powyżej (rem) dla klasy relative-root ustaliliśmy rozmiar fonta na 1.2 wielkości elementu głównego - rozmiar rem obliczany jest w tym wypadku jako 1.2 wymiaru fonta elementu `<html>` (domyślnie 16 px). Inny przykład: jeżeli dla html mamy ustalony rozmiar `20px`, to w klasie powyżej będzie to 24px.

W analogiczny sposób działa jednostka `em`, jednak w tym wypadku rozmiar jest obliczany na podstawie elementu rodzica. Korzystanie z tych jednostek pozwala nam na ustalenie wielkości tekstu jako proporcji w odniesieniu do innych elementów.

Powyższe pozwala ustalić jeden rozmiar w pikselach, najczęściej dla body (lub dla rodziców), a dla reszty elementów określać wypadkową wg hierarchii, **na zasadzie dziedziczenia**. Warto znać te jednostki, ale trend pokazuje, że coraz częściej wraca się do starych, dobrych pikseli :)

Grafiki

Kolejnym elementem stron, który trzeba ograć dla mobile (RWD), są grafiki (np. zdjęcia). W tym celu musimy ustawić następujące wartości dla selektora img:

```
img {
    max-width: 100%;
    height: auto;
}
```

Dzięki zastosowaniu deklaracji `max-width` zamiast tylko `width`, zdjęcia będą się zmniejszały wraz ze zmieniającą się rozdzielczością, zachowując odpowiednie proporcje. Nie będą jednak większe niż oryginalny rozmiar zdjęcia. Ponadto zdjęcia takie będą przyjmować rozmiar swojego elementu rodzica.

Aby kontrolować rozmieszczenie i rozmiary zdjęć, pomocny może się okazać w tym przypadku element `<figure>` HTML5, w którym umieszcza się znacznik img. Używa się go do oznaczania ilustracji w kodzie strony, dzięki czemu możemy mu nadać określony wymiar, a obrazkom w środku szerokość 100%.

Nic nie stoi na przeszkodzie, aby skorzystać też ze zwykłego elementu `<div>` zamiast `<figure>`.

To jednak nie wszystko. Urządzenia mobilne w dalszym ciągu muszą pobrać duże zdjęcie (przystosowane dla desktopu), które następnie będzie pomniejszone przez przeglądarkę **(zmniejszaj je najpierw do wymaganego rozmiaru programem graficznym)**

Dlatego dobrze byłoby przygotować różne rozmiary zdjęć (np. dla tła headera) i pobrać zdjęcie o rozmiarze przeznaczonym dla danego przedziału rozdzielczości, by nie obciążać niepotrzebnie łącza internetowego na telefonie.

Jest to możliwe dzięki nowym atrybutom elementu `img` (takim jak `srcset` i `sizes`) oraz elementowi `<picture>`. Jest to jednak ponownie rozwiązanie nowe i niedziałające na starszych przeglądarkach, ale warto spróbować go użyć.

Te atrybuty pozwalają na ustalenie różnego źródła obrazka w zależności od tego, jaka jest szerokość ekranu. Za pomocą atrybutu `srcset` ustalamy, który plik zostaje pobrany w zależności od tego, jaka jest szerokość ekranu. Następnie, za pomocą atrybutu `sizes`, ustalamy, jaka będzie szerokość obrazka w zależności od rozmiaru ekranu.

```
<img
srcset="quilt_3/large.jpg  1240w,
        quilt_3/medium.jpg  620w,
        quilt_3/small.jpg   310w"
sizes="(min-width: 41.25em) 38.75em,
       calc(100vw - 2.5em)"
src="quilt_3/medium.jpg"
alt="A crazy quilt whose irregular fabric scraps are fit into a latti
```

Takie rozwiązanie ma również pewien koszt w postaci stosunkowo dużej ilości kodu wymaganej do dodania pojedynczego obrazka. Z tego względu należy się zastanowić, czy takie rozwiązanie jest opłacalne. Przy małym projekcie nie musimy korzystać z tak rozbudowanej składni i ustalenie odpowiednich wymiarów, zawsze dla tej samej grafiki, może być rozwiązaniem wystarczającym.

Jakie breakpointy?

Pewnie zastanawiasz się teraz, jak dobrać punkty zmiany układu/wyglądu dla swojej strony i ile powinno ich być.

Najlepiej skorzystać z wartości stosowanych w produkcie używanym przez dziesiątki tysięcy developerów na świecie. Np. Twitter Bootstrap proponuje 3 takie progi (tym samym 4 przedziały rozdzielczości).

- `@media(max-width: 767px)` — urządzenia bardzo małe (telefony),
- `@media(min-width: 768px) and (max-width: 991px)` — urządzenia małe (tablety),
- `@media(min-width: 992px) and (max-width: 1199px)` — urządzenia średnie,
- `@media(min-width: 1200px)` — urządzenia duże (desktopy, monitory, tv).

Dla bardziej niskobudżetowych witryn można przyjąć np. 2 progi: poniżej 992px i powyżej. W końcu najmniej ludzi surfuje na tabletach, więc dostaną uproszczony widok jednokolumnowy :)

Inna sprawa, że zdarzają się klienci, których monitor ma ogromną rozdzielczość i wtedy możemy się spodziewać, że dołożą kolejne progi.

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

✔Zapoznałe(a)m się!

4.3. Tworzymy stronę responsywną!

Planowanie

Zobaczymy teraz, jak wygląda proces tworzenia strony responsywnej w praktyce. Mamy prosty prototyp, który w oknie przeglądarki komputera stacjonarnego ma wyglądać następująco:

Najpierw poczyńmy pewne założenia co do breakpointów/przedziałów:

1. **do 767px** — obsługa dla nowszych telefonów i starszych tabletów,
2. **768px — 1199px** — obsługa dla tabletów i starszych komputerów,
3. **1200px i więcej** — obsługa nowych komputerów.

Wskazówka: nadaj najpierw wszystkim elementom layoutu jakiś wyrazisty kolor tła lub ramki, bez umieszczania w nim treści. Nie wygląda to ładnie, ale najpierw ocenisz czy Twój układ klocków zachowuje się tak, jak trzeba w ramach RWD. Jeśli będzie ok, możesz wsadzać treść i upiększać stronę.

Wersja desktop

Kontener

Zaczniemy od budowy wersji desktopowej. Najpierw zajmijmy się ostatnim przedziałem. Musimy określić maksymalną szerokość elementów umieszczonych w kontenerze na max-width: 1200px; oraz stworzyć dobrze nam już znaną klasę pomocniczą `container`. Dzięki temu elementy o klasie container będą miały szerokość 1200px, a na większych urządzeniach będą miały marginesy (`margin: 0 auto;`). Na niższych niż 1200px rozdzielczościach kontenery tym samym będą zajmować 100% szerokości. Stwórz styl `.container` z takimi właściwościami.

Górna nawigacja i stopka

Niech będą na 100% szerokości. Do nich dodaj zagnieżdżonego diva i nadaj mu klasę `container`, aby treść w nawigacji i stopce była w ramach szerokości `1200px`, ale tło tych elementów rozciągało się na 100%.

Następnie wykonujemy pasek nawigacji tak, jak w poprzednim module. Będą to zagnieżdżone linki wewnątrz elementów listy `<li>`. Nadajemy im wyświetlane liniowo-blokowe, stan na `:hover` oraz wyśrodkowujemy w pionie. Za pomocą float przesuwamy linki w prawo. Nie zapomnij o logo!

Następnie czas na zbudowanie headera (główna sekcja, tzw. hero). Ustalmy wysokość elementu na `100vh`, co oznacza, że będzie on miał wysokość równą 100% viewportu. Tło headera może się rozciągać na 100%, ale teksty w nim zawarte mają być w kontenerze.

Galeria

Poniżej znajduje się nasza galeria. Nadajemy jej klasę container oraz clearfix, by zapobiec wybieganiu zdjęć poza obszar kontenera. Ponadto tworzymy też oddzielną klasę dla tej sekcji, by nadać jej margines od góry i od dołu, aby nie przylegała do innych elementów.

Przyszedł czas na zajęcie się zdjęciami. Musimy je zamknąć w zewnętrznych elementach z klasą np. photo, aby można nadać im szerokość 100%. Tworzymy dla nich oddzielną klasę i nadajemy jej float: left, szerokość 30% oraz margines z prawej strony równy 5%. Następnie przydałoby się ostatniemu elementowi galerii zlikwidować margines. Możemy go wybrać za pomocą selektora `.photo:last-of-type`, a następnie ustalić mu margines z prawej strony równy 0. W ten sposób stworzymy tr elementy o szerokości 30% z marginesami równymi 5% (z wyjątkiem ostatniego!), co daje nam razem 100% szerokości kontenera.

Dodatki

Poniżej dodajemy jeszcze jedną sekcję wewnątrz kontenera oraz stopkę z jakimś tekstem rozciągniętą na całą szerokość.

Pierwszy próg — 1200px i w dół

Teraz sprawdźmy, jak powinna zachowywać się strona w drugim przedziale rozdzielczości. Poniżej 1200px nasza strona powinna mieć stale 100% w stosunku do wielkości okna przeglądarki (viewport). Powinniśmy zobaczyć coś takiego:

Dodatkowo dla tego przedziału możesz przy pomocy media query zmniejszyć odległości pomiędzy linkami (jeśli masz ich wiele) oraz prawe marginesy pomiędzy zdjęciami.

Drugi próg — 768px i w dół

Następnie musimy ustalić wygląd naszego projektu dla urządzeń o najmniejszej rozdzielczości. W tym przypadku trzeba zmienić ustawienie niektórych elementów układu na naszej stronie (linków w nawigacji oraz zdjęć w galerii) **z układu poziomego na pionowy**. Dzięki temu będą one czytelne nawet na małym ekranie. Po tych zmianach nasza strona powinna wyglądać mniej więcej tak:

Elementy, które mają 100% szerokości, możemy zostawić bez większych zmian, modyfikując jedynie rozmiar czcionki (np. za duży napis w headerze). Więcej zabawy czeka nas w przypadku elementów ułożonych obok siebie (linki w nawigacji i galeria).

Modyfikacje zachowań elementów

Floaty

Najpierw musimy anulować deklarację float, aby wybrane elementy powróciły do swoich podstawowych właściwości (`float: none`). W przypadku nawigacji wystarczy zmienić sposób wyświetlania elementów listy na blokowy. Dzięki temu linki ustawią się jeden pod drugim. Teraz wystarczy, że ustawimy deklarację `text-align: center`, aby ustawić logo i linki w jednej linii.

Galeria

Pozostaje nam jeszcze zmiana ustawień zdjęć w galerii. Tu, podobnie jak w przypadku linków, musimy anulować deklarację float, dzięki czemu zdjęcia znajdą się jedno pod drugim. Zadbaj o responsywne wyświetlanie zdjęć oraz dodaj boczne paddingi dla całej treści w kontenerze, aby nie była przyklejona do krawędzi przeglądarki.

Mapa

Teraz przychodzi kolej na mapę. Google udostępnia możliwość dodawania statycznych map bez używania kodu JavaScript. Jest to możliwe poprzez wstawienie do treści strony obrazka z atrybutem `src` równym np. poniższemu linkowi:

```
https://maps.googleapis.com/maps/api/staticmap?center=Brooklyn+Bridge
```

Podobnie jak w przypadku serwisów z wypełniaczami, tak i tutaj, w wywołaniu HTTP ukryte są parametry mapy.

Skopiuj dokładnie powyższy link do atrybutu src obrazka. Dzięki umieszczeniu go w kontenerze i nadaniu mu szerokości 100% będzie się on dostosowywał do wymiaru ekranu. W ten sposób dodamy do naszej strony responsywną mapę (możemy też za pomocą Google Maps wygenerować własną).

Testy

Przydatną stroną do sprawdzenia działania naszej strony pod kątem RWD jest narzędzie stworzone przez Google do analizy prędkości ładowania strony:

https://developers.google.com/speed/pagespeed/insights/

Może się przydać (także w innych kontekstach) również ten adres:

http://mattkersley.com/responsive/

Narzędzie to wykrywa także błędy przy tworzeniu interfejsu responsywnego, np. gdy elementy nawigacji są za blisko siebie, za małe, niewidoczne etc.

Zadanie: Budujemy stronę

Twoja kolej! W projekcie zbuduj własną stronę zgodnie ze wskazówkami, które przedstawiliśmy powyżej.

Możesz modyfikować układ strony, dodać tam swój look&feel oraz zamieścić ciekawe komponenty CSS :)

Podgląd zadania

Przejdź do projektu ✔

4.4. Przystosowanie istniejącej strony do konwencji RWD

Dlaczego to robimy?

W świecie zdominowanym przez urządzenia mobilne decyzja o utworzeniu responsywnej strony, zawsze jest krokiem w dobrym kierunku. Nie zawsze jednak naszym zadaniem jest stworzenie takiej strony od zera.

Zdarza się, że decyzja o nadaniu strony responsywności zapada już po jej utworzeniu, a dyktowana jest chęcią poszerzenia grona odbiorców (np. blog) bądź klientów (sklepy, agencje usługowe).

Jak ugryźć RWD

W przypadku, gdy mamy już gotową stronę i klient dopiero w kolejnym kroku zapragnie mieć jej wersję mobilną lub po prostu zleci nam tylko wykonanie mobilnej wersji strony już istniejącej, stosujemy podejście *graceful degradation*.

Przede wszystkim należy się zastanowić, jak rozmieścić istniejące na stronie elementy w wersji mobilnej. Najlepiej byłoby, jeśli wraz ze zleceniem od klienta otrzymamy projekt mobilnej wersji od jego grafika. W przeciwnym przypadku, zadanie to spoczywa na naszych barkach. Dobrze jest wówczas przygotować makietę wersji mobilnej, by ułatwić sobie pracę.

Pamiętaj o umieszczeniu w części strony metatagu viewport, by jej szerokość dostosowywała się do szerokości urządzenia, na którym jest oglądana.

Planując zmiany wprowadzane wraz ze stworzeniem responsywnej wersji strony, musimy przeanalizować każdy z elementów. Niektóre z nich, na przykład przyciski, nie wymagają dużych zmian. Zastanów się jednak, czy ich użycie będzie wygodne na urządzeniu z ekranem dotykowym? Czy nie są za małe, przez co kliknięcie ich może sprawiać problem? A może są umieszczone zbyt blisko siebie, tak że łatwo jest niechcący trafić w inny przycisk?

Standardowym podejściem w przypadku treści umieszczonych w wierszu jest rozbicie wierszy na poszczególne kolumny i rozmieszczenie ich jedna pod drugą. Na przykład, jeśli w wersji desktopowej znajdują się trzy zdjęcia obok siebie, umieszczamy je pod sobą. Skorzystamy tutaj z naszego gridu i zadbamy o to, by był responsywny.

Style globalne oraz mobile first

Poznaliśmy już ogólne zalety grida. Teraz, krok po kroku, sprawimy, że wykorzystywany wcześniej grid będzie responsywny. Dla odmiany zastosujemy tutaj podejście mobile first, czyli stworzymy najpierw style do wersji mobilnej. Skorzystamy z media queries, by nadpisać odpowiednie właściwości CSS dopiero dla szerszych rozmiarów ekranu.

Zacznijmy od standardowych właściwości pomocniczych, które pozwolą nam zbudować grid. W ich wypadku nic się nie zmienia:

```
* {
    box-sizing: border-box;
}
.container {
    width: 100%;
    max-width: 1200px;
    margin: auto;
}
.row:before,
.row:after {
    content:"";
    display: table ;
    clear: both;
 }
```

Następnie przechodzimy do grupy właściwości przypisanych do wszystkich kolumn za pomocą selektora atrybutu. Ponieważ nasze style domyślnie (mobile first) są przeznaczone dla wąskich ekranów, nadamy na razie wszystkim kolumnom szerokość 100%. Nasz selektor będzie zatem wyglądał następująco:

```
[class*="col-"] {
    width: 100%;
    padding: 12px;
   float:left;
}
```

Dalej za pomocą media queries nadamy im odpowiednie szerokości dla wyższych rozdzielczości.

Media Queries

Teraz dodajmy odpowiednie media queries i wewnątrz nich umieścimy pozostałą część naszego gridu, tak jak w poprzednim module. Aby zachować konwencję, nadamy dla małych urządzeń nazwy klas `col-s-X`, a dla większych `col-m-X` ("col" od column, s — small, m — medium). Stwórzmy najpierw odpowiednie selektory, a potem zobaczymy, jak stosuje się to w praktyce. Wartości procentowe nie różnią się niczym w stosunku do poprzedniego modułu:

```
@media (max-width: 767px){
    /* For tablets: */
    .col-s-1 {width: 8.33%;}
    .col-s-2 {width: 16.66%;}
    .col-s-3 {width: 25%;}
    .col-s-4 {width: 33.33%;}
    .col-s-5 {width: 41.66%;}
    .col-s-6 {width: 50%;}
    .col-s-7 {width: 58.33%;}
    .col-s-8 {width: 66.66%;}
    .col-s-9 {width: 75%;}
    .col-s-10 {width: 83.33%;}
    .col-s-11 {width: 91.66%;}
    .col-s-12 {width: 100%;}
}
@media (min-width: 768px) {
    /* For desktop: */
    .col-m-1 {width: 8.33%;}
    .col-m-2 {width: 16.66%;}
    .col-m-3 {width: 25%;}
    .col-m-4 {width: 33.33%;}
    .col-m-5 {width: 41.66%;}
    .col-m-6 {width: 50%;}
    .col-m-7 {width: 58.33%;}
    .col-m-8 {width: 66.66%;}
    .col-m-9 {width: 75%;}
    .col-m-10 {width: 83.33%;}
    .col-m-11 {width: 91.66%;}
    .col-m-12 {width: 100%;}
}
```

Nasuwa się pytanie o powód stworzenia dwóch zestawów klas o identycznej zawartości. Zwróć uwagę, że każdy z nich jest aktywowany w zależności od tego, jaki jest rozmiar ekranu. Możemy nadawać temu samemu elementowi różne szerokości kolumn na różnych ekranach w taki sposób, że przypiszemy mu więcej niż jedną klasę. Robimy to tak, jak w przykładzie poniżej:

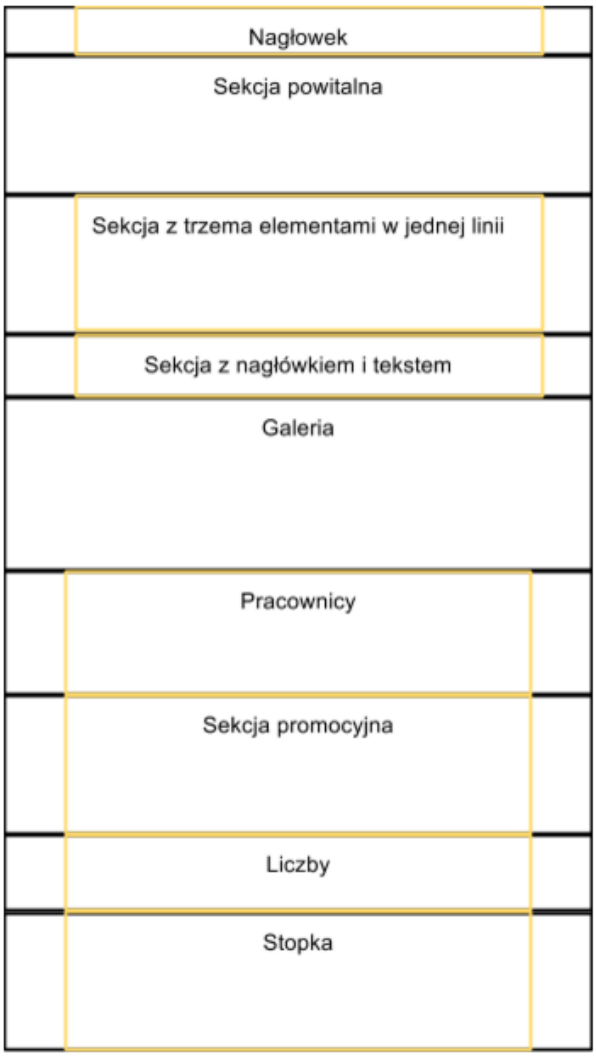
```
<div class="col-s-6 col-m-3">
```

Powyższy element będzie się rozciągać na 6 (50% szerokości) kolumn na małym ekranie, a na średnim na 3 (25% szerokości). W ten sposób możemy precyzyjnie kontrolować jego wymiary. To daje nam pełną kontrolę nad rozmiarami elementów na różnych ekranach.

Jest to tylko przedstawienie alternatywnego podejścia do obsługi urządzeń o różnych szerokościach ekranu. Wykonując zadanie w ramach tego submodułu, sugeruj się jedynie następną sekcją :)

Nadawanie responsywności krok po kroku

Jak już wspominaliśmy, najlepszym podejściem będzie przeanalizowanie i nadanie responsywności każdej części strony po kolei. Przypomnijmy sobie układ sekcji w projekcie, który cięliśmy w submodule 3.3:



Nagłówek

Aby zwiększyć czytelność menu na urządzeniach o niewielkich ekranach, zmień orientację jego elementów z poziomej na pionową. Możemy tego dokonać usuwając float oraz wyświetlanie liniowo-blokowe. Poniżej, w sekcji powitalnej, musimy zadbać o to, by obrazek prawidłowo skalował się do wymiarów ekranu.

Wykorzystujemy grid

Jeszcze niżej znajduje się sekcja z trzema elementami w jednym rzędzie. Z pomocą przyjdzie nam tutaj nasz grid. Dodając elementom klasę col-m-4, zadbamy o to, by powyżej breakpointu były one wyświetlane w rzędzie, zaś styl mobile first zadba o to, by na małym ekranie wyświetliły się pionowo.

Galeria

Podobnie w przypadku galerii. Ponieważ szerokość zdefiniowana jest procentowo, to elementy niemieszczące się w wierszu będą się przesuwać do wiersza poniżej. Aby jednak mieć większą kontrolę nad ich zachowaniem, możemy przypisać odpowiednie klasy z grida tak, aby poszczególne zdjęcia zajmowały więcej miejsca na małym ekranie. Musimy jednak pamiętać, że na małym ekranie użytkownik nie będzie miał do dyspozycji stanu `:hover`, przez co przyciski nie będą dostępne. Dlatego też trzeba będzie wewnątrz media queries umieścić odpowiednie style tak, aby ikony były widoczne.

Sekcja kadry

Sekcja z pracownikami również wykorzystywała grid. Możemy lekko zmodyfikować markup, by poszczególne elementy zajmowały 50% szerokości na tablecie. Wystarczy do tego celu dodać odpowiednią klasę. Znajdująca się poniżej sekcja z obrazkami posiadała dwa elementy zajmujące odpowiednio 1/ 3 oraz 2/3 szerokości. Poeksperymentuj z różnymi szerokościami w gridzie i dostosuj jej wygląd do małego ekranu. Podobna sytuacja ma miejsce w przypadku statystyk znajdujących się poniżej.

Stopka

Pewnych problemów może nastręczyć sekcja stopki, gdyż znajduje się tam zagnieżdżony grid. Ponownie możemy wykorzystać klasy stworzone na potrzeby urządzeń mobilnych. Musimy jednak przypilnować, by zawartość mieściła się wewnątrz stopki i nie wypływała za nią.

Zadanie: Dodaj responsywność do swojego projektu!

Bazując na projekcie utworzonym w submodule 3.3, nadaj swojej stronie responsywność zgodnie ze wskazówkami z sekcji "Nadawanie responsywności krok po kroku".
Najlepiej będzie, jeśli skopiujesz kod z 3.3 i wkleisz do tego projektu.

Podgląd zadania

Przejdź do projektu ✔

4.5. Flexbox

Sugestia

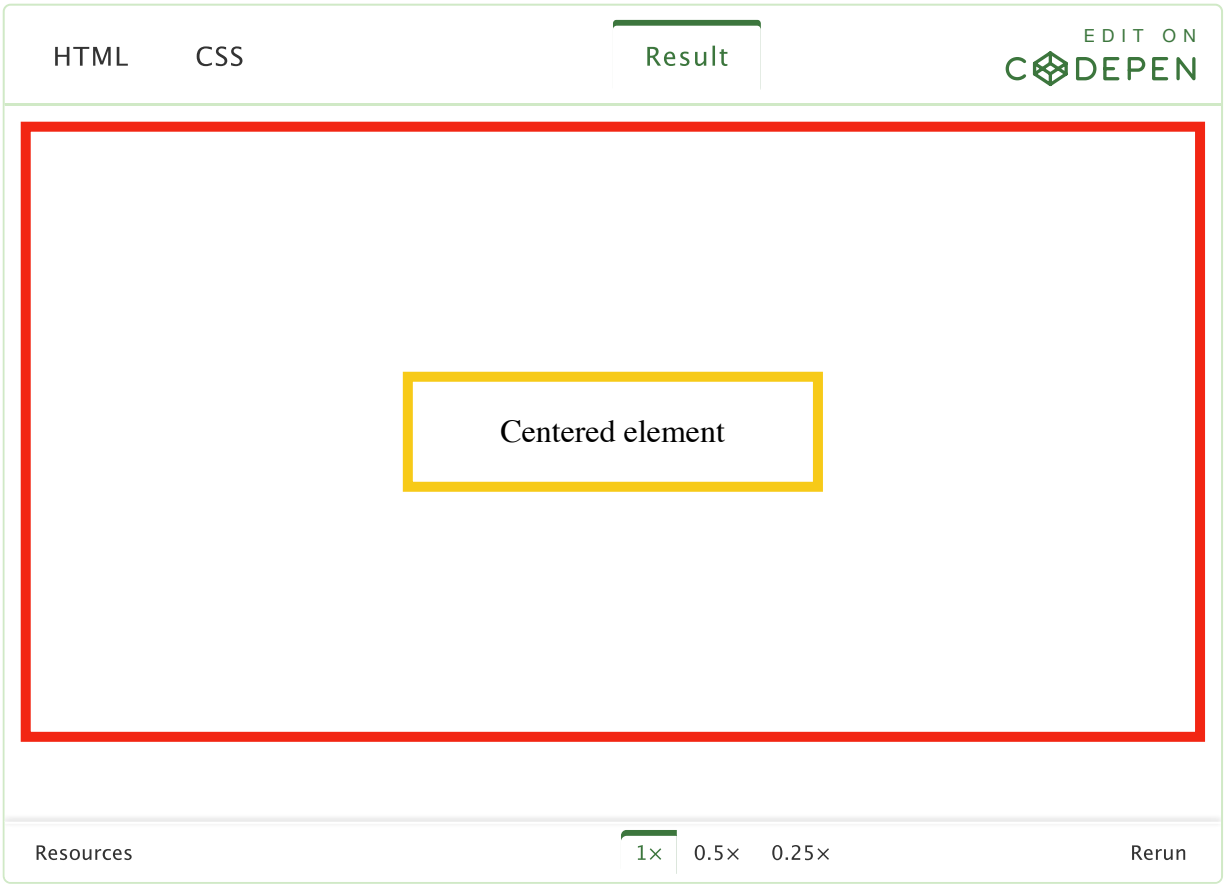
Przed rozpoczęciem tego modułu (lub po teorii) warto przejść ten kurs Flexbox, aby przećwiczyć w praktyce podstawowe zagadnienia, co pomoże przy wykonywaniu zadań.

Czym jest flexbox i kiedy z niego korzystać?

Aby sprostać wyzwaniom związanym z prawidłowym wyświetlaniem elementów na różnych urządzeniach, stworzony został tzw. flexbox, który jest odpowiedzią na wiele bolączek związanych z tworzeniem layoutu.

Podstawową ideą przyświecającą temu sposobowi tworzenia layoutu jest umożliwienie kontenerowi zmiany wymiarów oraz rozmieszczenia elementów potomnych. Dzięki temu elementy znajdujące się w tym kontenerze zostaną rozciągnięte, by odpowiednio wypełnić dostępne miejsce bądź zmniejszone, aby nie przekraczać granic kontenera. Takie rozwiązanie pozwala na zapewnienie prawidłowego wyświetlania strony na szerokiej gamie ekranów i przeglądarek.

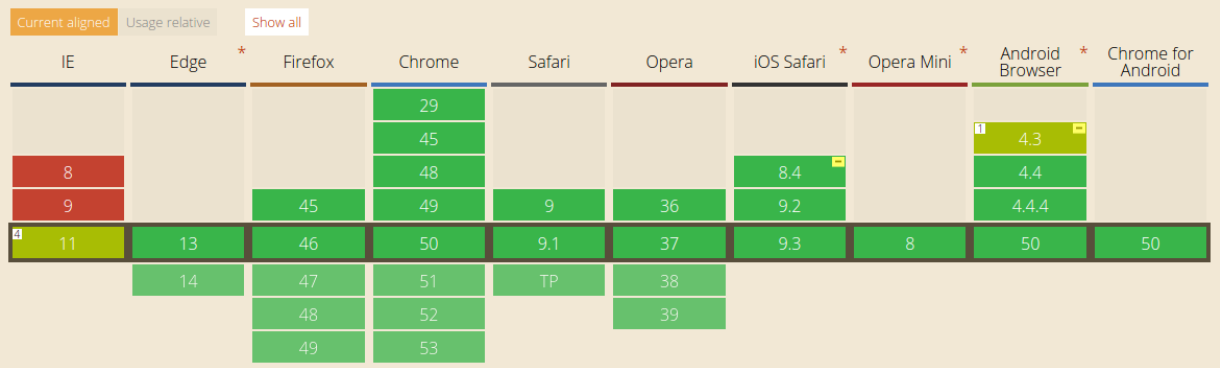
Do czego możemy go wykorzystać? Przy jego użyciu możemy stworzyć inteligentny kontener, który w prosty sposób wyrówna treść. Dzięki flexboksowi takie zadania, jak np. wyśrodkowanie elementów w pionie stają się dużo prostsze, tak jak w przykładzie poniżej:



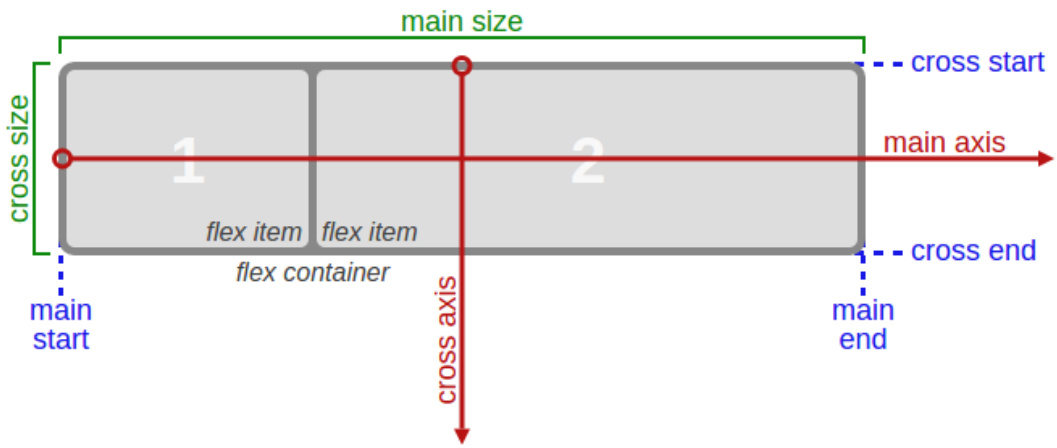
Możemy również dostosowywać wielkość elementów, aby np. dwa z nich miały określony wymiar, a pozostałe będą zajmować pozostałą szerokość. Jest to oczywiście wykonalne bez użycia fleksa, lecz wielu developerów jest zdania, że stosowanie tego rozwiązania jest dużo prostsze niż budowa layoutu w klasyczny sposób. Nie musimy wówczas uciekać się do takich sztuczek jak clearfix, ponadto zyskujemy dodatkowe możliwości, np. zmiany kolejności elementów bez ingerowania w kod HTML.

Jest to jednak rozwiązanie stosunkowo nowe, przez co wiele starszych przeglądarek nie wyświetla poprawnie layoutu zbudowanego w oparciu o flexbox. W miarę jednak zmniejszania się udziału starszego oprogramowania w rynku staje się coraz mniejszy, flexbox zdobywa coraz więcej zwolenników i jest coraz częściej wybierany do tworzenia layoutu.

Jak widać poniżej, problematyczny w przypadku flexboksa jest Internet Explorer we wcześniejszych wersjach. Co do zasady, jeżeli nie zależy nam na wsparciu starszych przeglądarek, możemy skorzystać z flexboksa.



Terminologia



Anatomię layoutu tworzonego przy użyciu flexboksa przedstawia nam powyższy schemat. Będziemy korzystać przede wszystkim z dwóch elementów: **flex container**, czyli rodzica interesujących nas elementów, oraz **flex item**, czyli jego bezpośrednich potomków w drzewie dokumentu. **Main axis** to główna oś, wzdłuż której nasze elementy będą rozmieszczane, zaś jej długość określana jest jako **main size**. Dla drugiej osi (**cross axis**) będzie to określane analogicznie za pomocą **cross start** oraz **cross size**.

Kontrolowanie zachowania kontenera

Wszystkie właściwości, jakie udostępnia nam flexbox, możesz przetestować na stronie http://the-echoplex.net/flexyboxes/

Aby rozpocząć budowę layoutu opartego o flexbox, musimy nadać kontenerowi poniższą właściwość:

```
.flex-container {
display: -webkit-flex;
display: flex;
}
```

Dodana w ten sposób własność display wystarczy do uruchomienia tej funkcjonalności w większości przeglądarek. Powyższa deklaracja stworzy kontener będący elementem blokowym, możemy stworzyć również element liniowy za pomocą deklaracji

```
.inline-flex-container {
display: -webkit-inline-flex;
display: inline-flex;
}
```

Przyjrzymy się teraz deklaracjom, które możemy wykorzystać do manipulowania rozmieszczeniem elementów wewnątrz kontenera.

Flex-direction

Ustala kierunek i zwrot głównej osi (main-axis). Domyślną wartością jest `row`, czyli rozmieszczenie elementów w poziomym rzędzie, od lewej do prawej. Za pomocą wartości `column` tworzymy pionową kolumnę. Możemy odwrócić kolejność elementów wewnątrz kontenera, dodając odpowiednio `row-reverse` oraz `column-reverse`.

Flex-wrap

Domyślne elementy wewnątrz kontenera będą umieszczone w jednej linijce, jednak za pomocą tej właściwości możemy to zmienić. Gdy przypiszemy kontenerowi właściwość `flex-wrap: wrap`, elementy będą rozmieszczone w wielu linijkach. Podobnie jak poprzednio możemy tutaj zmienić kierunek, deklarując `wrap-reverse`.

Justify-content

Pozwala na rozmieszczenie elementów wzdłuż głównej osi. Mamy tutaj następujące możliwości:

- `flex-start` (domyślne) — elementy będą umieszczone na początku linii,
- `flex-end` — na końcu linii,
- `center` — na środku,
- `space-between` — elementy będą rozciągnięte wzdłuż głównej osi; pierwszy będzie na początku linii, ostatni na końcu, a pozostałe będą rozmieszczone na środku,
- `space-around` — podobne do space-between, jednak zachowane będą równe odległości pomiędzy poszczególnymi elementami; pierwszy i ostatni element będą

oddalone od krawędzi kontenera jedynie o połowę odległości pomiędzy poszczególnymi komórkami.

Align-items

Za jego pomocą możemy modyfikować rozmieszczenie elementów na osi poprzecznej (cross-axis). Można przyjąć, że jest to funkcjonalność podobna do `justify-content`, jednak w wersji dla osi poprzecznej. Podobnie jak w poprzednim przypadku mamy tutaj do dyspozycji różne możliwości.

- `flex-start` — górne krawędzie elementów będą umieszczone przy górnej krawędzi kontenera,
- `flex-end` — analogicznie jak w przypadku flex-start, jednak dla dolnych krawędzi: będą one dociągnięte do dolnych krawędzi kontenera,
- `center` — elementy będą umieszczone centralnie na osi poprzecznej,
- `baseline` — elementy będą rozmieszczone tak, by zachować ich linię bazową, tj. by litery w poszczególnych elementach były umieszczone na jednej linii,
- `stretch` — zawartość będzie rozciągnięta, aby wypełnić kontener (warto przy tym zaznaczyć, że wartości min-width/max-width będą zachowane).

Align-content

Umożliwia rozmieszczenie linii z elementami wewnątrz kontenera w sytuacji, gdy jest ich więcej niż jedna linijka (dlatego też nie będzie to mieć znaczenia wtedy, gdy nie zadeklarowaliśmy odpowiednio właściwości `flex-wrap`).

Właściwości flexboksa dla elementów wewnątrz kontenera

Flexbox daje nam również możliwość manipulowania poszczególnymi elementami. Dzięki nim możemy przykładowo zmieniać kolejność elementów lub ich rozmiar.

Order

Elementy wewnątrz flexboksa będą domyślnie wyświetlane w takiej kolejności, w jakiej znajdują się w źródle strony. Możemy to zmodyfikować, dodając właściwość order do interesującego nas elementu. Przyjmuje on wartość liczby całkowitej.

Flex-grow

Definiując `flex-grow` możemy proporcjonalnie powiększyć konkretny element, jeżeli tego potrzebujemy. Przykładowo: jeżeli wszystkie elementy mają zdefiniowany `flex-grow: 1`, a jeden będzie określony z `flex-grow: 2`, będzie on zajmował dwa razy więcej wolnego miejsca (jeżeli będzie taka możliwość wewnątrz kontenera).

Flex-shrink

Podobnie jak w poprzednim przypadku, jednak tym razem nasz element zostanie pomniejszony.

Flex-basis

Określa domyślny rozmiar elementu. Za nim rozmieszczona jest pozostała zawartość i przydzielone jej miejsce wewnątrz kontenera. Wartość auto powoduje, że będzie to rozpatrzone na podstawie własności `width` oraz `height`. Możemy również przydzielić rozmiar elementu na podstawie zawartości nadając wartość content.

Flex — właściwość skrótowa

Podobnie jak w wypadku definiowania marginesów, możemy zdefiniować jednocześnie `flex-grow`, `flex-shrink` oraz `flex-basis`. Domyślną wartością jest `flex: 0 1 auto;`.

Align-self

Ta właściwość to nic innego jak nadpisanie położenia pojedynczego elementu, które było zdefiniowane przy użyciu `align-items`. Dlatego też może przyjmować identyczne wartości.

https://codepen.io/kodilla/pen/5f2a660935d96ebaf17c090017257bf3/
Istnieją gry, które szlifują umiejętności posługiwania się flexboksem — Flexbox Froggy oraz Flexbox Defense. Jest to świetna opcja by utrwalić materiał.

Zadanie: Komponenty na bazie flexboksa.

Skorzystamy z flexboksa, aby zbudować w oparciu o niego elementy layoutu strony.

1. Zacznijmy od nawigacji. Stwórz element <header>, a w nim jeden z klasą container. Nadaj mu właściwość `display:flex`. To będzie nasz kontener. Pamiętaj o odpowiednich prefiksach. Wewnątrz stwórz listę <ul> z kilkoma linkami wewnątrz elementów <li>. Niech lista, w wyniku dodania tej samej właściwości również sama będzie kontenerem.

2. Za pomocą właściwości `justify-content: flex-end;` przenieś linki nawigacji na prawą stronę.
3. Stworzymy jumbotron. W tym celu stwórz sekcję z nagłówkiem `<h1>`. Wyśrodkowanie nagłówka nigdy nie było prostsze! Dodaj jumbotronowi właściwość `display: flex` oraz `align-items: center;` i `justify-content: center;`. To wyśrodkuje nagłówek w obu osiach.
4. Czas na galerię: dodaj sekcję z 6 obrazkami o wymiarach 400 x 400 pikseli. Jak zapewne zobaczysz, będą one wybiegać poza obręb kontenera galerii. Aby temu zapobiec, stwórz tam kolejny flexbox i dodaj mu właściwość: `flex-wrap: wrap`. W ten sposób zostaną one przeniesione do kolejnego wiersza.
5. Zbudujmy teraz karty. Stwórz kolejną sekcję-kontener będącą flexboksem. Dodaj tam więcej elementów (np. 10). Niech każdy z nich ma przynajmniej 200px szerokości i zawiera w sobie krótki tekst. Ponownie dodaj zawijanie wiersza za pomocą właściwości `flex-wrap`. Następnie wyśrodkuj poszczególne elementy, dodając do kontenera właściwość: `justify-content:center;`. Sprawdź teraz, jak zachowuje się sekcja w mniejszym oknie przeglądarki!
6. Elementy również mogą być kontenerami. Wyśrodkuj w nich tekst za pomocą reguły `text-align`. Dodaj im także `display:flex` oraz odpowiednio właściwości `justify-content` i `align-items` równe `center`.
7. Czas na stopkę. Dodaj w niej 3 sekcje, jedną zawierającą zdanie o zastrzeżeniu praw, jedną z nazwą firmy oraz jedną z adresem. Dodaj do środkowej właściwość `flex-grow:2`, by rozciągnąć ją w stosunku do pozostałych.

Jest to jedynie zadanie praktyczne, mające na celu zaznajomienie z tego typu rozwiązaniem. Efekt końcowy powinien wyglądać mniej więcej w ten sposób.

Podgląd zadania

Przejdź do projektu ✔

Regulamin

Polityka prywatności