**Czym jest RWD?**

RWD to skrót od nazwy Responsive Web Design. Jak sama nazwa wskazuje, ten termin odnosi się do dziedziny mającej na celu tworzenie stron maksymalnie elastycznych, reagujących na środowisko, w których się znajdują.  
Głównym założeniem jest dostosowanie wyświetlanych elementów to rozmiaru urządzenia bądź przeglądarki (bo przecież nawet jak mamy duży ekran, to możemy skalowac nasze okno).

**Kiedy stosujemy RWD?**

Można pomyśleć, że RWD jest już absolutnie wszędzie. Czy faktycznie zawsze tak to wygląda?  
Mimo, że termin istnieje w świadomości programistów aplikacji internetowych od lat to wielu z nich nadal nie do końca rozumie wszystkie dobre praktyki. Nawet dobrzy fachowcy mają czasem trudność ze stworzeniem prawdziwie responsywnych stron.  
Są jednak przypadki, w których RWD nie jest dla nas istotne. Jeżeli piszemy aplikację dedykowaną na ekrany komputerów, które mają wyświetlać bardzo dużą ilość danych (głównie statystycznych, tabele), niektórzy klienci świadomie rezygnują z tworzenia wersji na tablety czy telefony komórkowe.  
RWD jest jednak nieodłącznym elementem typowych stron internetowych, które mają trafić do szerokiego, bliżej niesprecyzowanego grona odbiorców.

**Jakie urządzenia powinniśmy brac pod uwagę?**

Tworząc aplikację/stronę reponsywną powinniśmy brać pod uwagę przynajmniej 3 najważniejsze *breakpointy*.

1. Mobile
2. Tablet
3. Laptop/PC

Na temat tego jakie są dokładne szerokości, dla których powinniśmy dostosować widoki jest wiele debat.  
Najpopularniejszy obecnie podział sugeruje następujące breakpointy:

1. Od 320px dd 576px (smartfony o stosunkowo niedużych ekranach)
2. Od 576px do 768px (duże smartfony i tablety)
3. Od 768px do 992px (tablety w widoku horyzontalnym, małe ekrany komputerów)
4. Powyżej 1200px (laptopy i desktopy)

Tworząc bardzo kompleksowe rozwiązanie można wziąć pod uwagę ekrany powyżej 1600px (duże monitory + telewizory). Trzeba jednak pamiętac o tym, że tak duże ekrany niekoniecznie są powodem do tego aby w pełni zagospodarować ich miejsce. Zbyt duża ilość elementów na stronie obniża ich czytelność, więc naszym celem mogłoby być ewentualnie powiększenie pewnych elementów, aby były bardziej widoczne z dużych odległości.

**AWD?**

Poza terminem RWD istnieje jeszcze jeden, zbliżony, ale różniący się pewnymi założeniami - Adaptive Web Design.  
W RWD mamy jeden layout, który zgodnie z dobrymi praktykami powinien być płynny (operujemy na wartościach proporcjonalnych, a nie sztywnych jak pixele). Do wiekszych zmian dochodzi podczas przełamywania breakpointów, ale dalej mamy do czynienia z tym samym designem.  
AWD to rozwiązanie, w którym tworzymy zauważalnie różne rozwiązania na poszczególne kategorie urządzeń. Wygląd strony na urządzenia mobilne może być serwowany pod całkiem inną domeną (często wygląda to tak: m.domena.pl) i być kompletnie osobną aplikacją. Nie manipulujemy już w tak subtelny sposób położeniem elementów jak w RWD.

**Co w takim razie jest lepsze?**

Jak zwykle, będziecie słyszeć te słowa wielokrotnie w swojej karierze - to zależy.  
  
Jeżeli jako klient bądź producent jakiegoś rozwiązania mamy świadomość, że na różnych urządzeniach będziemy trafiać do zauważalnie różnych grup społecznych, nasi odbiorcy będą mocno różni, to możemy pokusić się o wprowadzenie AWD.  
  
Dzięki danym statystycznym na temat ewentualnych gości na naszej stronie, możemy dopasować treści w nieco bardziej kompleksowy sposób, ale to rozwiązanie ma pewną ważną wadę, która sprawiła, że jest obecnie mniej popularne.  
  
Czas. Tworząc AWD musimy przygotować dużo więcej kodu - zarówno HTML, CSS jak i JavaScript, ponieważ będziemy manipulować bardzo zmiennymi elementami.  
  
Podczas gdy wiele stron może być responsywna *out of the box* dzięki gotowym narzędziom jak Bootstrap, w przypadku AWD potrzebowalibyśmy we wielu przypadkach nawet 300% nakładu takiej pracy.

**Techniki RWD**

Aby uzyskać efekt responsywnej strony internetowej możemy korzystać z dwóch najważniejszych technik:

* Fluid Layout
* Media Queries

**Fluid Layout**

We fluid layoucie chodzi o to abyśmy tworzyli elementy o możliwie płynnych kształtach.  
Dla przykładu, jeżeli zakładamy, że nasz container ma szerokość 768px i chcemy w nim zmieścić 2 elementy, na połowę tej szerokości, to możemy zrobić to na 2 sposoby:

.container {

width: 768px;

margin: 0 auto;

}

.box {

width: 384px;

}

.container {

width: 768px;

margin: 0 auto;

}

.box {

width: 50%;

}

W obu przykładach efekt będzie teoretycznie ten sam - jeżeli wrzucimy do containera 2 divy o klasie .box, to zajmą one równo po połowie.  
Co w przypadku kiedy container się zmniejszy np do rozmiaru 480px, dla urządzeń mobilnych?  
Wtedy będziemy musieli nadpisać również style .box, a przecież i tak chodziło o to żeby zajęły po 50% dostępnej przestrzeni. W drugim przykładzie będzie to działać w każdym możliwym scenariuszu.

**Media Queries**

Fluid Layout sprawia, że nasza strona jest elastyczna dzięki wartościom proporcjonalnym. Skalują się dynamicznie i dają nam już pewien przedsmak tego jak strona powinna wyglądąć. Dochodzi jednak do takich sytuacji, w których będziemy musieli w większym stopniu manipulować naszymi stylami w taki sposób, aby dostosowały się do zmieniającego rozmiaru okna.

**Zastosowanie Media Queries i Fluid Layout**

Aby stosować *breakpointy*, używamy parametru CSS'owego @media screen.  
Po słowie kluczowym *and* możemy w nawiasach podać informację dla jakich minimalnych bądź maksymalnych rozdzielczości chcemy stosować zmiany w stylach.

.container {

width: 960px;

margin: 0 auto;

}

.box {

width: 50%;

}

@**media** screen and (max-width: 960px){

.container {

width: 768px;

}

}

We wskazanym wyżej przykładzie mówimy przeglądarce: tutaj definiuję podstawowe style moich elementów *ale* jeżeli rozdzielczość będzie wpisywać się w podaną przeze mnie w Media Query, zmień je proszę na takie jak zostały w nim zadeklarowane.  
To znaczy, że tylko parametr width zostanie nadpisany dla .containera i jeżeli nasz ekran ma mniej niż 960px szerokości, to chcemy aby kontener zajmował 768px;

**Mobile First**

Przeglądarka czyta kod podobnie jak człowiek czyta książkę - od góry na dół. Dlatego kod, który znajdzie się wyżej od pozostałych, zostanie szybciej zinterpretowany i wyświetlony.  
Na urządzeniach mobilnych, mimo ich szybkiego rozwoju, mamy ograniczoną moc obliczeniową, a dodatkowym *wąskim gardłem* jest fakt, że ich użytkownicy często korzystają z sieci komórkowych, które w zależności od lokalizacji, mogą działać szybciej lub wolniej.  
Kluczowym elementem jest czas ładowania takiej strony. Jeżeli trwa zbyt długo, użytkownik opuści naszą stronę i poszuka innej, która działa sprawniej.

**Zastosowanie Mobile First**

Zgodnie z omówioną wyżej zasadą, zależeć nam będzie na tym aby style na urządzenia ładowały się od *najmniejszych rozdzielczości*.  
Najprawdopodobniej telefon o małym ekranie będzie stosunkowo wolniejszy od swoich "większych braci" dlatego chcemy aby przeglądarka serwowała ich style priorytetowo.  
Wróćmy do naszego przykładu powyżej:

.container {

width: 960px;

margin: 0 auto;

}

.box {

width: 50%;

}

@**media** screen and (max-width: 960px){

.container {

width: 768px;

}

}

Czy on aby na pewno wpisuje się w taktykę Mobile First? Nie, jest odwrotnie.  
Aby zrobić z niego przykład zastosowania Mobile First, nasze style dla *mniejszego* urządzenia powinny być pierwsze, a dopiero potem w Media Query przypisywać te dla większych.

.container {

width: 768px;

margin: 0 auto;

}

.box {

width: 50%;

}

@**media** screen and (min-width: 1200px){

.container {

width: 960px;

}

}

Jak wyglądałby poprawny scenariusz dla kontenera przygotowanego na najważniejsze *breakpointy*?

.container {

width: 100%;

margin: 0 auto;

}

@**media** screen and (min-width: 768px){

.container {

width: 480px;

}

}

@**media** screen and (min-width: 960px){

.container {

width: 768px;

}

}

@**media** screen and (min-width: 1200px){

.container {

width: 960px;

}

}

Dla najmniejszych urządzeń container nie potrzebuje mieć marginesów z lewej i prawej strony, bo i tak mamy małą przestrzeń do zagospodarowania - dlatego domyślna szerokość to 100% ekranu.  
W kolejnych @media definiujemy jak ma zmieniać się jego rozmiar wraz ze *wzrostem* rozdzielczości - o to w tym wszystkim chodzi :)

**Zadanie domowe - zrób, nim przejdziesz dalej!**

Zadanie nr 1 - Spraw aby przygotowana przez Ciebie strona miała zielone tło, ale zmieniła je 3 razy: dla rozdzielczosci powyżej 480px ma być różowe, dla rozdzielczosci powyżej 768px ma zmienić się na niebieskie, a powyżej 960px niech będzie żółte.  
  
Zadanie nr 2: Stwórz <div> o klasie *container*, który ma 100% szerokości i właściwość css margin: 0 auto; i wraz ze zmieniającą się szerokością, zmieniaj jego rozmiar następująco: - powyżej 768px niech jego szerokość wyniesie 720px - powyżej 960px niech jego szerokość ma 920px - powyżej 1200px niech ma szerokość 960px