Informatietechnologie 1

Cascading Style Sheets

Responsive web design - Positioning

Kristof Michiels

In deze presentatie

- Wat is Responsive web design? Waarom belangrijk?
- Vloeiende layouts
- Media queries
- Strategie en patronen
- Relatieve positioning
- Absolute positioning
- Fixed positioning

Responsive web design

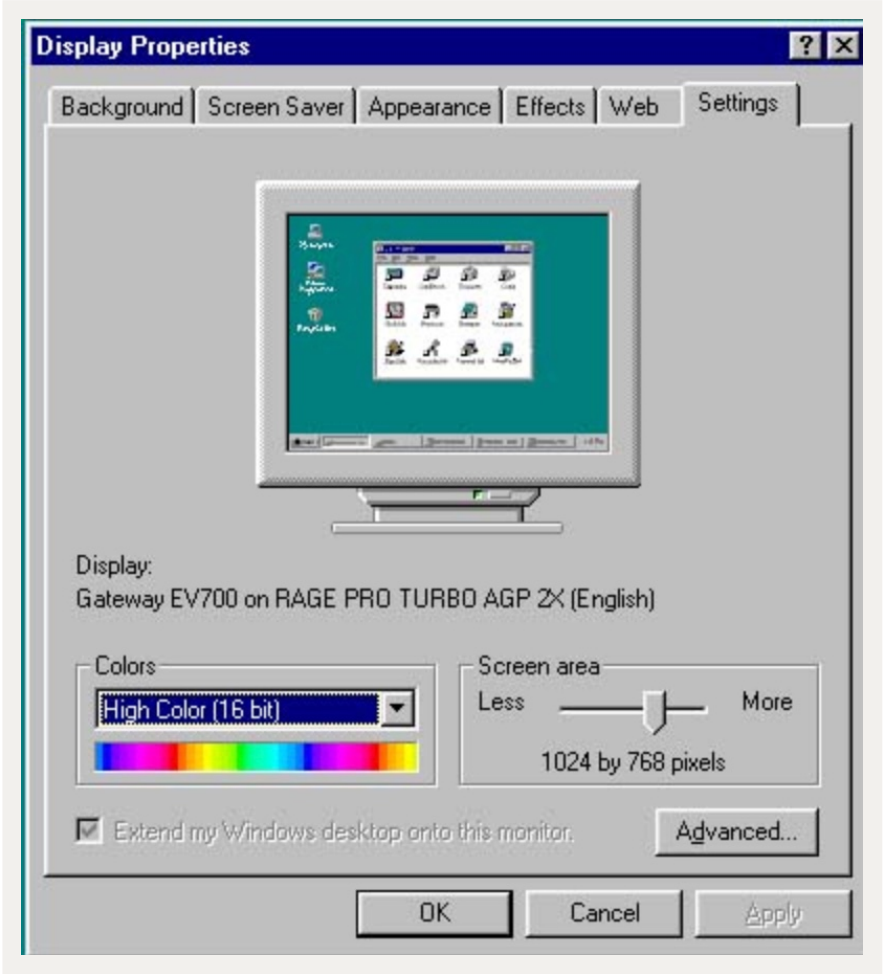
Wat is Responsive web design?

- Aanpak die toelaat dat een website goed weergegeven wordt op alle toestellen/schermafmetingen
- Alle toestellen krijgen zelfde html
- De CSS wordt gedeeltelijk aangepast aan de afmetingen van de viewport
- Responsive web sites werken goed op smartphones, tablet en desktop
- Belangrijk: ze zien er ook goed voortreffelijk uit op élke schermbreedte daar tussenin

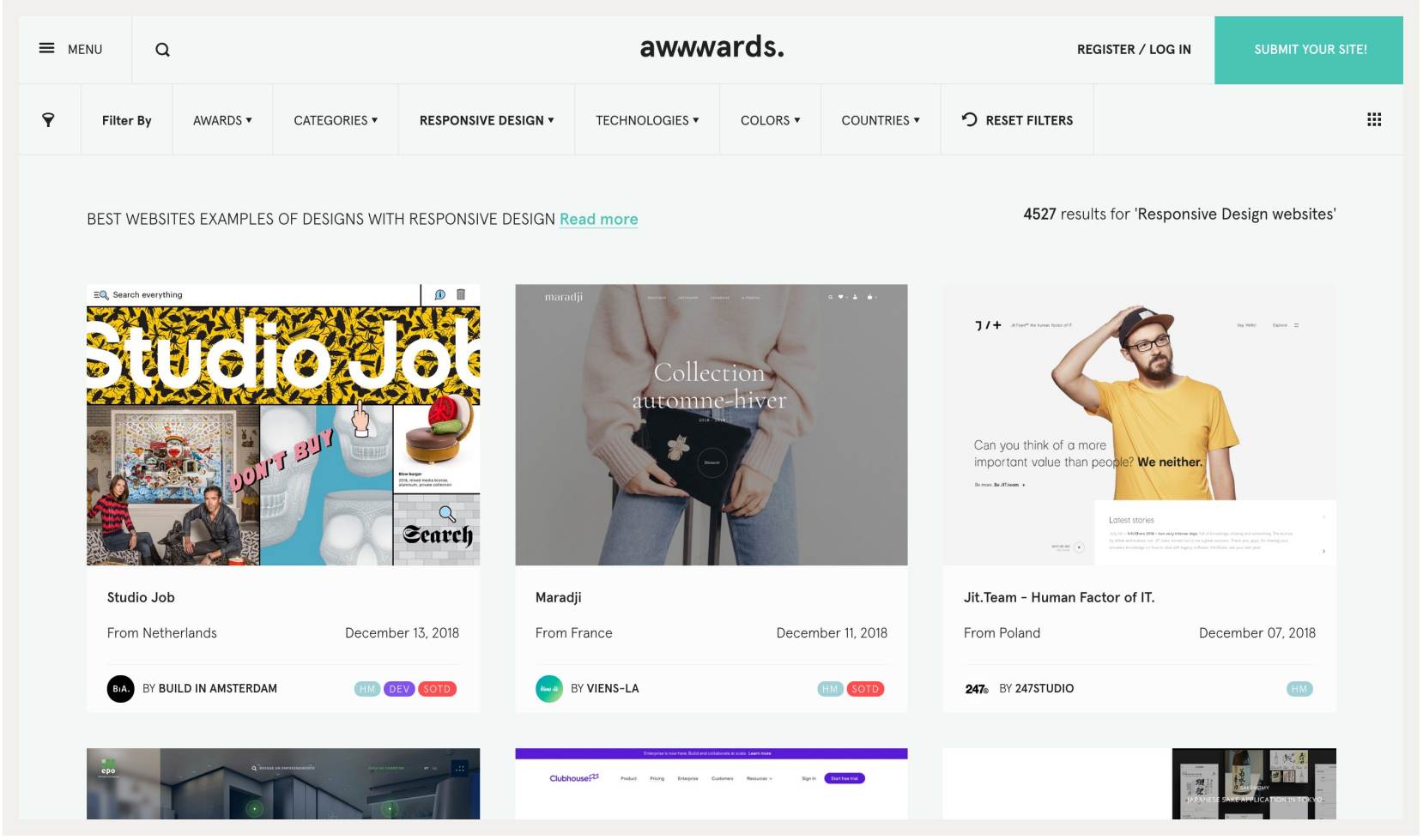
Waarom Responsive web design?

- Mobiele revolutie (iPhone vanaf 2007) zorgde voor een veelheid aan schermen
- Vandaag heeft de smartphone de desktop voorbij gestoken qua marktaandeel
- e-commerce-winkels zien meer klanten binnenkomen via smartphone en tablet dan via desktop
- Voor moderne web designers als ons is "responsive design" gewoon web design. We hebben geen keuze
- Het is de standaard manier geworden om de realiteit van de verschillende toestellen te omarmen

The old days: 1024x768px

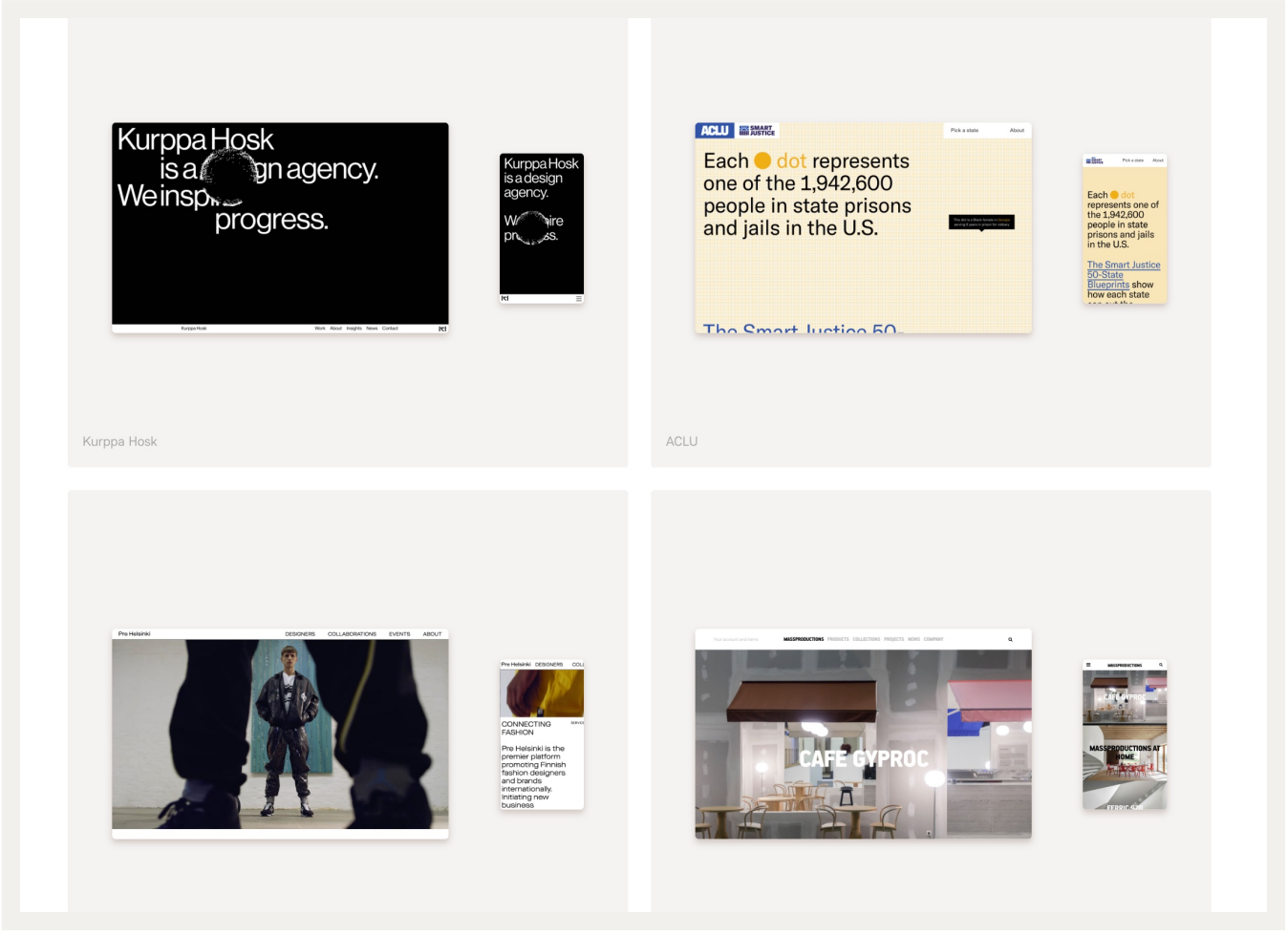


Awards of design, creativity & innovation on the i'net

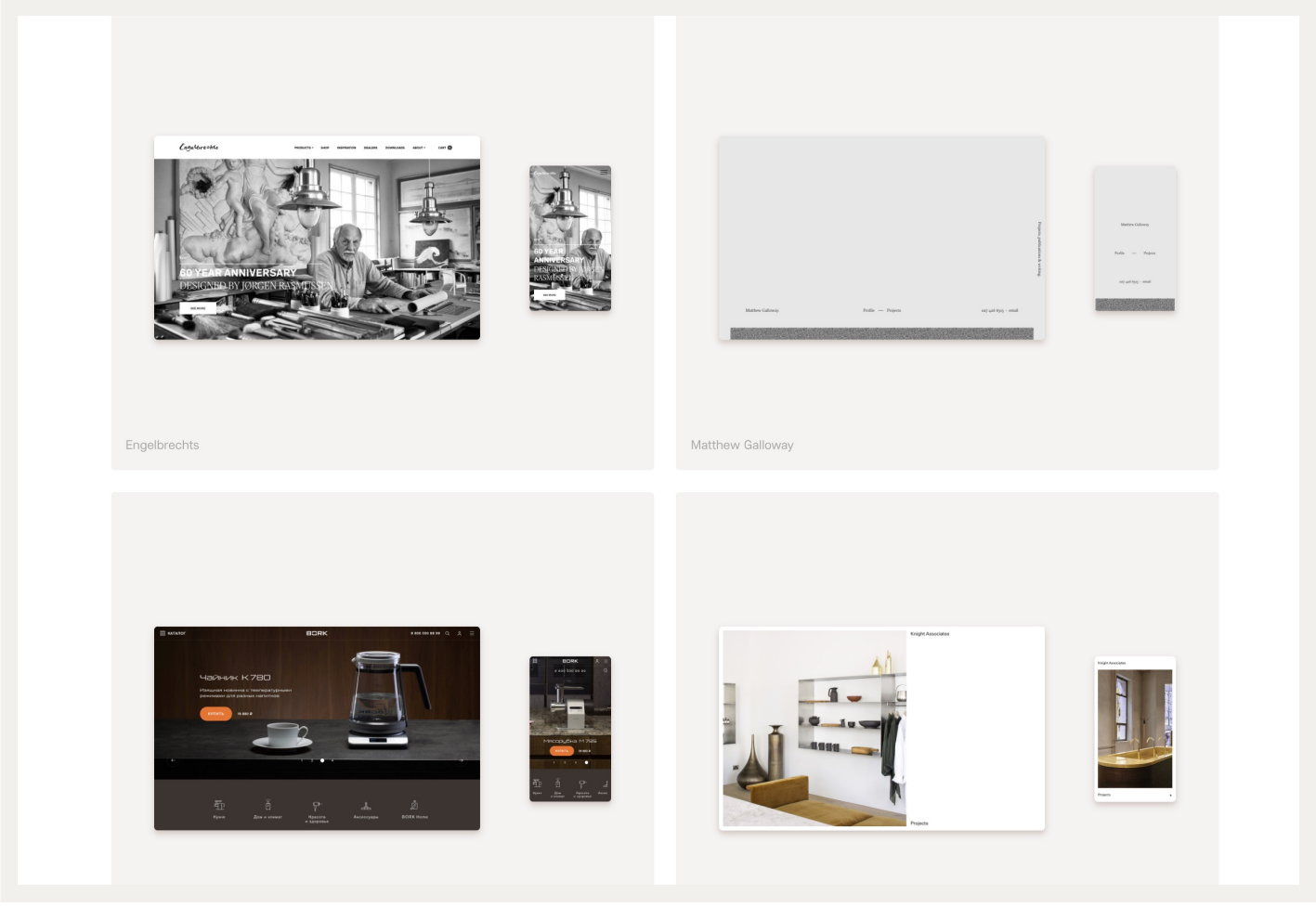


https://www.awwwards.com/

Responsive Web design



Responsive Web design (2)



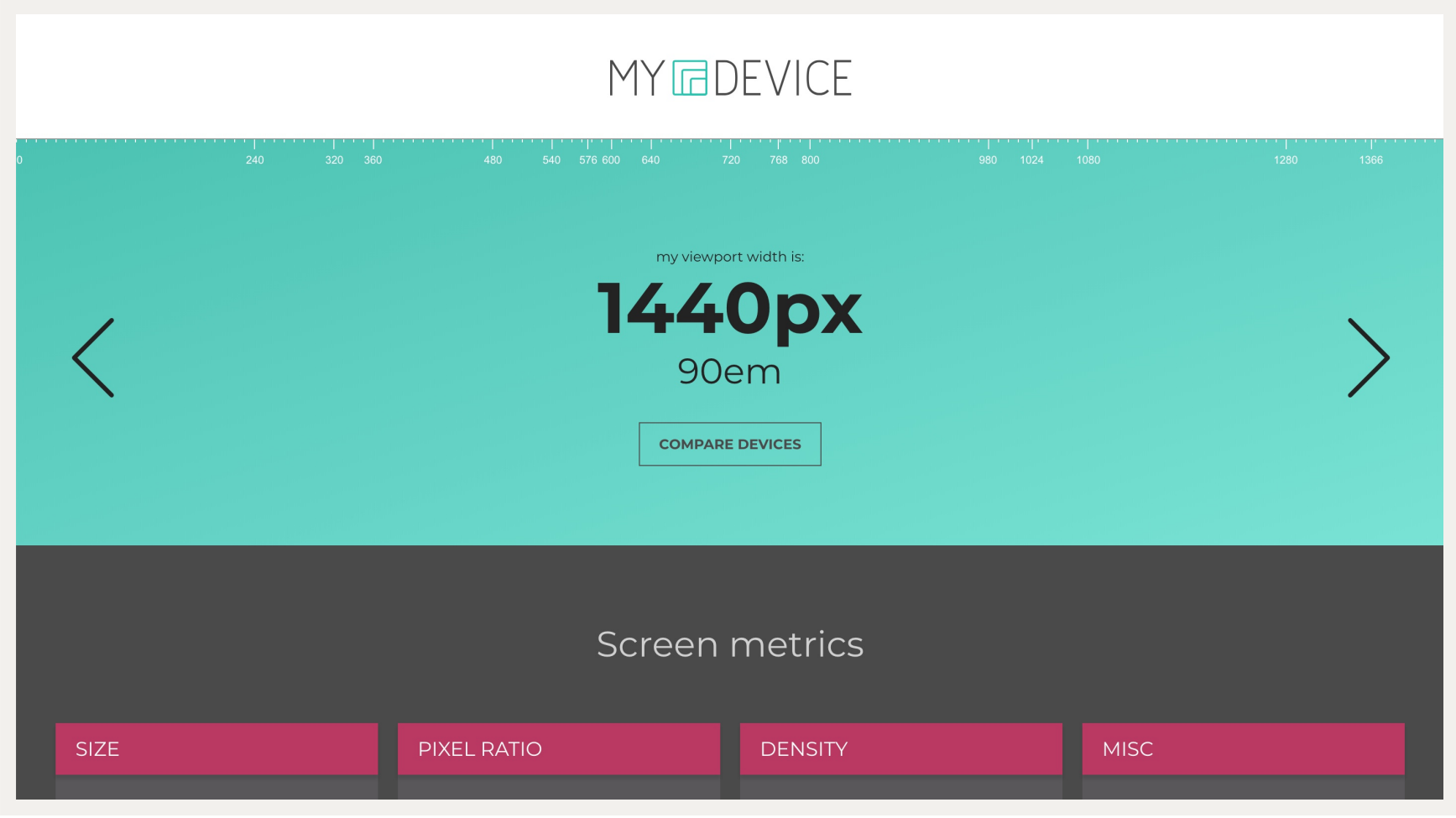
Responsiviteit: hoe doen we het?

- Flexibele grid: geen statische breedte, wel relatieve waarden (% of vw) die de browser automatisch vullen
- Flexibele afbeeldingen: die zich automatisch aanpassen aan de beschikbare ruimte
- Media queries: zorgen ervoor dat bepaalde CSS regels bvb pas vanaf bepaalde breedte worden toegepast

De Viewport instellen

- Om standaard (dus niet-responsive) websites weer te kunnen geven op kleine schermen gaan mobiele browsers de pagina uitzetten op een canvas die de viewport wordt genoemd
- De viewport wordt dan verkleind zodat het past op het kleine scherm
- We noemen de echte breedte van het scherm de device width
- bvb bij iPhone wordt een site standaard weergegeven met een breedte van 980px die dan verkleind wordt weergegeven tot 320 a 414px (afhankelijk van model)
- Mydevice.io (zie volgende slide) toont je al deze waarden

mydevice.io



https://www.mydevice.io/

Een belangrijk meta-element

```
<meta name="viewport" content="width=device-width,initial-scale=1">
```

- Dit meta-element komt in de head van de html-pagina en vertelt de mobiele browser dat de schermbreedte effectief deze van het toestel dient te zijn en dat het zoomlevel standaard op 1 (100%) moet staan
- Als nu de effectieve schermbreedte van een smartphone 320px is dan zal je webpagina effectief ook op 320px breedte worden weergegeven
- We gaan vanaf nu deze meta-tag steeds toevoegen aan onze webpagina's
- Het is een cruciale eerste stap naar een responsive web design

Flexibele grids (vloeiende layouts)

- We zagen reeds dat als je breedtes zet in %, vw's of fr je een soort elasticiteit krijgt
- De elementen krijgen hierdoor een soort elasticiteit: ze krimpen of zetten uit naarmate er minder of meer schermbreedte beschikbaar is
- Deze elasticiteit is exact wat we nodig hebben om een hele reeks viewports te kunnen bedienen
- Ze vormen mee de basis van responsive web design (al is het mogelijk om responsive te werken met widths in rem gecombineerd met media queries)

```
body {
    width: 80vw;
    margin: 5rem auto;
}
```

Afbeeldingen flexibel maken

- Met onderstaande eenvoudige regel kan je ervoor zorgen dat afbeeldingen altijd de breedte van hun container aannemen
- Is de plaats beperkter, dan ook de breedte van de afbeelding
- Hiermee maken we dus de afbeeldingen responsive
- Je kan op deze manier zelfs de width en height van de afbeelding weglaten

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

Media queries

- De elasticiteit van onze vloeiende widths brengt ons al een heel eind maar soms kunnen onze teksten in bepaalde situaties te lang worden
- We zeggen dan dat ons design "breekt"
- We zullen dus voor verschillende breedtes ook aanpassingen moeten kunnen maken in onze css
- Dat doen we met media queries

Media queries

- We kunnen met media queries in alle vrijheid aanpassingen maken aan onze designers
- Een voorbeeld hier is het opdelen van content in meerdere kolom naarmate de schermbreedte toeneemt
- Je schrijft ze zoals op onderstaand voorbeeld. Let op dat je jezelf niet verliest in de vele accolades. Met correcte en heldere insprongen werken helpt

```
body {
    background-color: tomato;}

@media (min-width: 80rem) {
        body {
            background-color: limegreen;}
}
```

Media types

- Mogelijke waarden: all, print, screen, speech
- Kunnen toegevoegd worden aan de media query
- Indien niet aanwezig wordt uitgegaan dat het om een screen type MQ gaat
- Je kan dit dus rustig weglaten maar weet wat het betekent als je er eentje ziet

```
@media screen and (min-width: 80rem) {
        body {
            background-color: limegreen;}
}

@media print {
        ...
}
```

Media feature queries

- We gaan hiermee testen op bepaalde voorwaardes
- Mogelijke features: width, height, orientation (portrait, landscape), aspect-ratio (bvb 16/9), ...
- Wij gaan steeds min-width gebruiken: omdat we mobile-first gaan werken

```
body {
    background-color: tomato;}

@media (min-width: 80rem) {
        body {
            background-color: limegreen;}
}
```

Hoe media queries gebruiken?

- Wij gaan onze media queries steeds in onze externe stylesheet schrijven
- We schrijven ze steeds onderaan de pagina
- Op deze manier kunnen we eerst de algemene stijlregels schrijven
- Wordt vervolgens voldaan aan de criteria van de media query dan wordt extra CSS toegepast
- Eén css bestand kan meerdere media queries bevatten
- Herinner je dat we in onze stylesheet ook een @supports-query zullen hebben die nagaat of de browser van de bezoeker grid layout ondersteunt

Hoe media queries gebruiken?

```
body {
    background-color: tomato;}

@media (min-width: 80rem) {
        body {
            background-color: limegreen;}
}

@media (min-width: 120rem) {
        body {
            background-color: purple;}
}

@media (min-width: 140rem) {
        body {
            background-color: magenta;}
}
...
```

Mobile first

- Wij gaan altijd beginnen met designen voor het kleinste scherm
- Mobile first is de naam van deze design strategie
- Je begint hierbij eerst na te denken over de mobiele versie van je webpagina
- Gaandeweg breid je uit naar de bredere schermversies
- Designvoordeel: je concentreert je eerst op de kleinste schermen. Er is minder ruimte dus de essentiële features van je applicatie komen naar de voorgrond
- Voordeel voor ons is ook dat dit goed samengaat met onze strategie voor browsers die grid layout niet ondersteunen. We geven ze dan een veredelde mobiele versie van onze site

Breekpunten kiezen

- Een breekpunt is een punt waarop je design "breekt", m.a.w. niet meer mooi wordt weergegeven in de browser
- We gaan dit dan verhelpen door een media query te plaatsen met daarin css die voor een oplossing zorgt
- Breekpunten kunnen per website anders liggen. Jij kiest ze door het scherm zelf breder uit te rekken tot het design "breekt". Je hebt dan zelf een breekpunt gevonden en kunt hier ingrijpen met nieuwe css

Rem-gebaseerde breekpunten

- We zetten onze breekpunten met een min-width media query en werken steeds met rem-waarden

```
body {
    background-color: tomato;}

@media (min-width: 80rem) {
        body {
            background-color: limegreen;}
}
```

Onze strategie

```
body {max-width:100rem;}

@supports (grid-area: auto) {

    body {max-width:none;}

    @media (min-width: 80rem) {
        main {
            display: grid;
            grid-template-columns: 1fr 1fr;}
    }
}
```

Positioning

Gepositioneerde elementen

- Is een techniek om html-elementen te (ver-)plaatsen op een webpagina
- Wij gebruiken het niet als algemene layouttechniek (wel: grid layout)
- Vroeger werd deze techniek hiervoor wel gebruikt (samen met floating)
- Wel zeer nuttig voor plaatsingswerk binnen de grid
- En voor creatieve extra's: bvb het voor/achter elkaar plaatsen van elementen

Elementen positioneren

- Doen we met de CSS eigenschap position
- Alle elementen hebben een standaard waarde: static
- Een waarde anders dan static gebruiken en het element wordt een gepositioneerd element
- De opties zijn hier: relative, absolute, fixed en static
- Gepositioneerde elementen kunnen de eigenschappen top, left, bottom en right gebruiken voor plaatsing

```
div.mijndiv {
    position: relative;
    left: 15rem;
    top: 5rem;
}
```

Relatief positioneren

- Is de minst ingrijpende vorm
- Het element blijft in de normale flow van het webdocument staan (dwz: de elementen eronder schuiven niet naar boven)
- Met top, left, bottom en right (met telkens lengte eenheden als waarde) verplaats je het element vanaf zijn niet-gepositioneerde startpositie

```
div.mijndiv {
    position: relative; /* doet op zich niets */
    left: 15rem;    /* hiermee verplaats je het element 15rem naar rechts */
    top: 5rem;    /* hiermee verplaats je het element 5rem naar onder */
}
```

Absoluut positioneren

- De meest ingrijpende vorm
- Neemt een element uit zijn normale flow (dwz: de elementen eronder schuiven wel naar boven)
- Maar je krijgt veel meer flexibiliteit: je zal het element positioneren vanaf de rand van de browser

```
<div class="mijndiv">
     <h2>Nieuwe CSS eigenschap ontdekt!</h2>
     <h3>Door: Kristof Michiels</h3>
     <p>Gisteren werd bij opgravingen in Egypte een nieuwe CSS eigenschap ontdekt</p>
</div>
```

```
h3 {
     position: absolute;
     right: 1rem; /* de h3 komt op 1rem links van de rechterrand van het scherm */
     top: 1rem; /* komt op 1rem van de bovenrand van het scherm */}
```

Absoluut positioneren

- Een uiterst belangrijke strategie bij absoluut positioneren
- Je kan de browser toelaten de positie van een element niet uit te rekenen vanaf de schermrand, maar wel vanaf een "dichter" liggend omkaderend element
- Je zet dan gewoon een position:relative op dat omkaderend element naar keuze
- Gevolg: met top, left, bottom en right reken je nu vanaf dit omkaderend element

```
div.mijndiv {
    position: relative;} /* position:relative op het omkaderend element */

h3 {
     position: absolute;
     right: 1rem; /* wordt nu berekend vanaf de div.mijndiv */
     top: 1rem; /* wordt nu berekend vanaf de div.mijndiv */}
```

Elementen verankeren met fixed positioning

- Hiermee zet je een element op een vaste plaats
- De op die manier geplaatste elementen scrollen niet mee naar boven maar blijven op dezelfde plaats staan
- Haalt het element zoals bij position:absolute uit de normale flow

```
div {
     position: fixed;
     right: 1rem;
     top: 1rem;
}
```

Elementen voor/achter elkaar plaatsen met z-index

- Met positioneren kan je elementen over elkaar schuiven
- Ingrijpen op de overlappen en de volgorde (welk element is zichtbaar, welk niet) doe je met z-index
- Geen z-index of gelijke index = de overlapping wordt bepaald door plaatsing binnen de DOM

```
<main>
     <img class="klasse1" src="afb1.jpg" alt="Afbeelding 1">
     <img class="klasse2" src="afb2.jpg" alt="Afbeelding 2">
</main>
```

```
    .klasse1, .klasse2 {
        position: relative;
        z-index: 1;}
```

Elementen voor/achter elkaar plaatsen met z-index

- Het element met de hoogtste z-index waarde is bovenaan zichtbaar
- Belangrijk: z-index werkt enkel met gepositioneerde elementen (gebruik relative als het je enkel om de z-index te doen is)
- z-index: kan 0, 1, 2, 10, 100 of zelfs 9999 zijn

```
.klasse1, .klasse2 {
    position: relative;
    z-index: 1;}

.klasse2 {
    z-index: 10;
}
```

IT1 - Responsive Web design - Positioning

Vragen?

kristof.michiels01@ap.be