Responsive Design: Neue Weblayout-Strategien für 2024

Jonas Hellwig am 1. Juni 2024
Lesezeit 7 Minuten

Das Konzept des Responsive Designs hat sich über die Jahre hinweg stark weiterentwickelt. Ursprünglich lag der Fokus darauf, Weblayouts an verschiedene Displaygrößen anzupassen. Heutzutage umfasst modernes Responsive Webdesign jedoch die Anpassung an eine Vielzahl von Kriterien.

In der Regel werden Projekte heute nach dem Mobile-First-Ansatz umgesetzt. Das User Interface ist modular aufgebaut und mindestens strukturell in Layouts und Komponenten unterteilt. In einigen Fällen wird sogar ein detaillierterer Ansatz wie Atomic Design verfolgt. Layouts und Komponenten funktionieren eigenständig und verfügen über individuelle Breakpoints, die auf Media Queries bzw. Container Queries basieren. Zudem wird häufig fluides Type-Scaling angewendet, idealerweise in Verbindung mit einer daraus abgeleiteten Spacing-Hierarchie. Als Kür werden die Einstellungen in Browsern und Betriebssystemen berücksichtigt, um sicherzustellen, dass das Layout entsprechend reagiert.

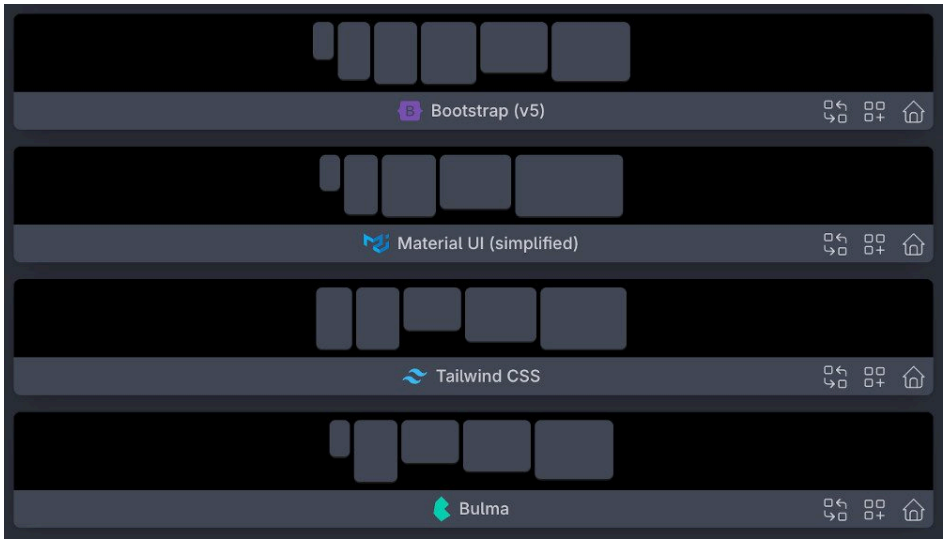
Insgesamt passt sich das Design nicht mehr nur an die Geräte an, sondern auch viel stärker an die Inhalte und die Bedürfnisse der Benutzer. In einigen Bereichen überschneidet sich das Konzept des Responsive Designs daher mit Themen wie Barrierefreiheit oder nachhaltigem Webdesign.

Lassen Sie uns gemeinsam einen Blick auf die Entwicklung und die aktuellen Best Practices werfen.

Der Browser als Leinwand

HTML war schon immer flexibel, sodass Texte und andere Inhalte sich der Größe des Displays anpassen konnten. Um jedoch ein wirklich flexibles Layout zu erreichen, fehlten variable Medieninhalte, Breakpoints und flexible Raster, die durch Ethan Marcottes Definition von Responsive Webdesign (RWD) im Jahr 2010 festgelegt wurden.

Kurz nach der Einführung des RWD-Konzepts empfanden viele Designer den Browser als schwer kontrollierbares Medium. Als Reaktion darauf wurden vertraute Arbeitsweisen aus dem Printdesign auf das Web übertragen: Die verschiedenen Seitenarten (Startseite, Textseite usw.) wurden speziell für unterschiedliche Geräteformate (Smartphone, Tablet, Desktop usw.) gestaltet, was einen erheblichen Aufwand bedeutete. Mithilfe flexibler Gestaltungsraster wurden diese Formate in Spalten und Zeilen unterteilt und die Inhalte in diesem Raster platziert.



Übersicht über Standard-Viewports und Breakpoints verschiedener populärer Frameworks im Polypane-Browser

Diese Herangehensweise führte dazu, dass das Layout eher für verschiedene beliebte Geräte optimiert wurde. Es entstanden globale Breakpoints für Webseiten, die den Abmessungen der am häufigsten verwendeten Geräte entsprachen. Diese globalen Breakpoints wurden durch populäre Frameworks wie Bootstrap weit verbreitet.

```
/* Standard-Stile für Desktop-Layout */
.container {
        max-width: 960px;
        margin: 0 auto;
        padding: 1.5em;
}

/* Media Query für Tablet-Geräte mit einer maximalen Breite von 767px */
@media only screen and (max-width: 767px) {
        .container {
        padding: 1em;
        }
}

/* Media Query für Smartphone-Geräte mit einer maximalen Breite von 479px */
@media only screen and (max-width: 479px) {
        .container {
        padding: 0.5em;
        }
}
```

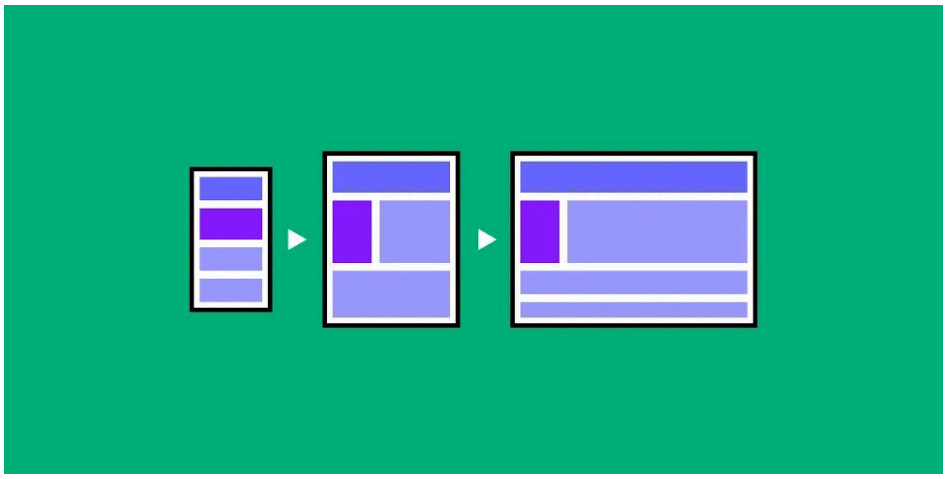
RWD_1.txt hosted with ❤ by GitHub view raw

Veralteter technischer Ansatz mit Desktop-First geschriebenen Media Queries und Pixelangaben

Mobile First

Mit der Einführung des iPhones und kurz darauf des Responsive Designs verlagerte sich das Webdesign deutlich von großen Desktop-Bildschirmen auf mobile Geräte. Anfangs stand die Desktop-Ansicht einer Webseite im Mittelpunkt, und die Version für kleine Displays wie Smartphones wurde davon abgeleitet. Diese Herangehensweise wurde sowohl im Design als auch im Code als „Desktop-First“ bezeichnet.

Allerdings wies dieser Ansatz einige Nachteile auf. Oft war es schwierig, alle Inhalte der Desktop-Version ordentlich auf dem Smartphone unterzubringen. Dies führte dazu, dass Inhalte auf dem Smartphone ausgelassen wurden, was letztendlich bedeutete, dass diese Inhalte für Smartphone-Benutzer nicht zugänglich waren. Auch die Reihenfolge von Inhalten auf kleinen Geräten bereitete oft Probleme.



Schematische Darstellung des Mobile-First-Ansatzes (Quelle: UXPin)

Der Mobile-First-Ansatz löste diese Probleme, indem zunächst die Smartphone-Version einer Webseite entwickelt und daraus dann die Desktop-Version abgeleitet wurde. Dadurch wurden von Anfang an Prioritäten gesetzt. Zusätzlich wurden einige mobile Gestaltungslösungen, wie beispielsweise das Hamburger-Menü, von der mobilen Ansicht in die Desktop-Ansicht übernommen.

Auch im Code wurde eine Mobile-First-Entwicklung angewendet. Zu Beginn des Responsive Webdesigns folgte der CSS-Code noch dem Desktop-First-Ansatz: Die Stile für die Desktop-Version standen ganz oben im CSS-Dokument, und erst weiter unten wurden innerhalb von Media Queries die Stile für kleinere Displays überschrieben.

Allerdings offenbarten sich auch hier einige Nachteile. Die Smartphone-Version zeigte die meisten Inhalte untereinander an und erstreckte sich über die volle Breite des Displays. Da die meisten Layoutelemente in HTML Block-Elemente sind und automatisch über die volle Breite laufen und untereinander angezeigt werden, konnte die Smartphone-Version oft mit sehr wenig Code erstellt werden. Das Smartphone-Layout entsprach gewissermaßen dem HTML-Standard.

Beim Desktop-First-Coding-Ansatz wurde dieser Standard zu Beginn des CSS-Dokuments überschrieben und dann weiter unten im Code mit eigenen Stilen wiederhergestellt. Insgesamt wurde dadurch viel mehr Code geschrieben – und geladen – als notwendig war. Das Dokument wurde unnötig komplex.

Aus diesem Grund wurde der Code umgedreht und auch im CSS-Dokument mit der kleinen (mobilen) Ansicht begonnen. Mit Media Queries wurden dann komplexere und größere Layouts ergänzt. Diese Vorgehensweise entsprach stärker dem HTML/CSS-Standard und führte zu weniger Code und kürzeren Ladezeiten.

```
@media only screen and (min-width: 48em) {
        .container {
        max-width: 50rem;
        }
```

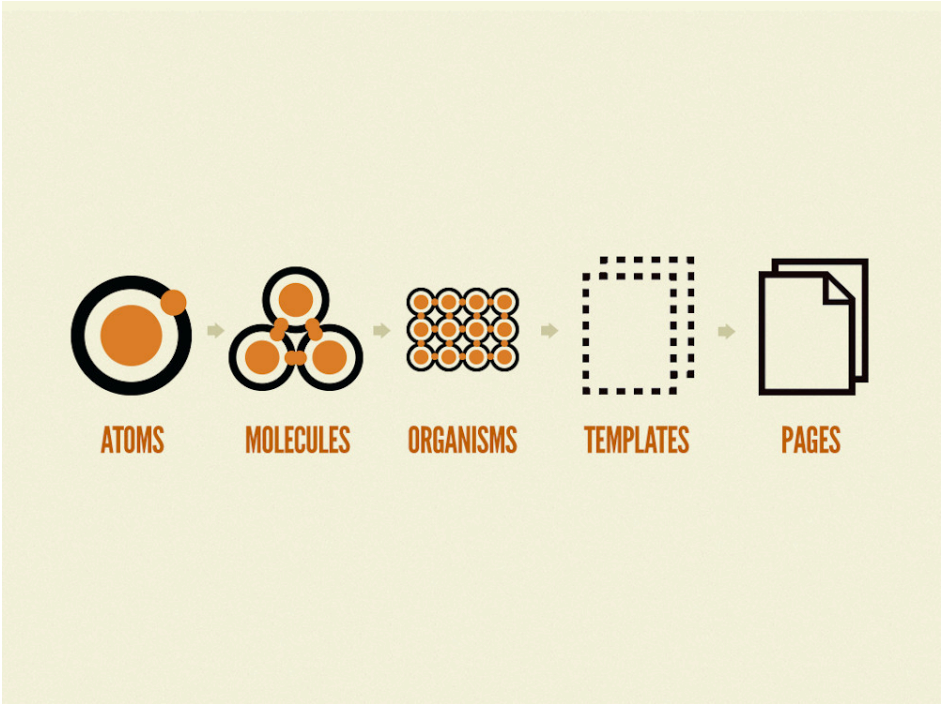
RWD_2 hosted with ❤ by GitHub view raw

Der Mobile-First-Ansatz macht es nicht notwendig, dem Container eine Breite von 100% zuzuweisen, da das der Standard für Block-Elemente ist. Erst sobald abgewichen werden soll, wird ein Media Query eingesetzt. Zudem werden hier alle Breitenangaben in flexible rem-Einheiten umgerechnet.

Modulares Design & Atomic Design

Die traditionelle Herangehensweise mit Seitentypen (Templates) erwies sich zunehmend als unhandlich und ineffizient. Als Reaktion begannen Webdesigner, globale Webseiten-Elemente wie Header und Footer separat zu betrachten.

Das Konzept des Atomic Designs hat sich als grundlegendes Denkmodell für die Organisation von Weblayouts etabliert. Seitdem setzen sich einzelne Seitentypen modular aus Komponenten zusammen. Diese Veränderung führte dazu, dass Breakpoints verstärkt für jede Komponente individuell geplant wurden, anstatt global für die gesamte Webseite basierend auf den Seitentypen.



Visualisierung des Atomic Design-Prinzips von Brad Frost

Um den modularen Ansatz im CSS-Code umzusetzen, wurden zunächst separate CSS-Dateien für Komponenten und Templates erstellt. Eine Herausforderung bestand jedoch darin, dass die Technik zur Erzeugung von Breakpoints (Media Queries) auf die gesamte Breite des Browserfensters reagierte.

Viele Komponenten erstrecken sich jedoch nur über einen Teil des Layouts und sollten daher nicht auf die gesamte Browserfensterbreite, sondern auf den verfügbaren Platz im Layout reagieren. Um dies zu erreichen, mussten im CSS-Code lange Zeit Tricks angewendet werden mussten, bis Anfang 2023 die passende Technik der sog. Container Queries verfügbar war.

HTML

```
    <div class="container">
            <div class="card">
            …
            </div>
    </div>
```

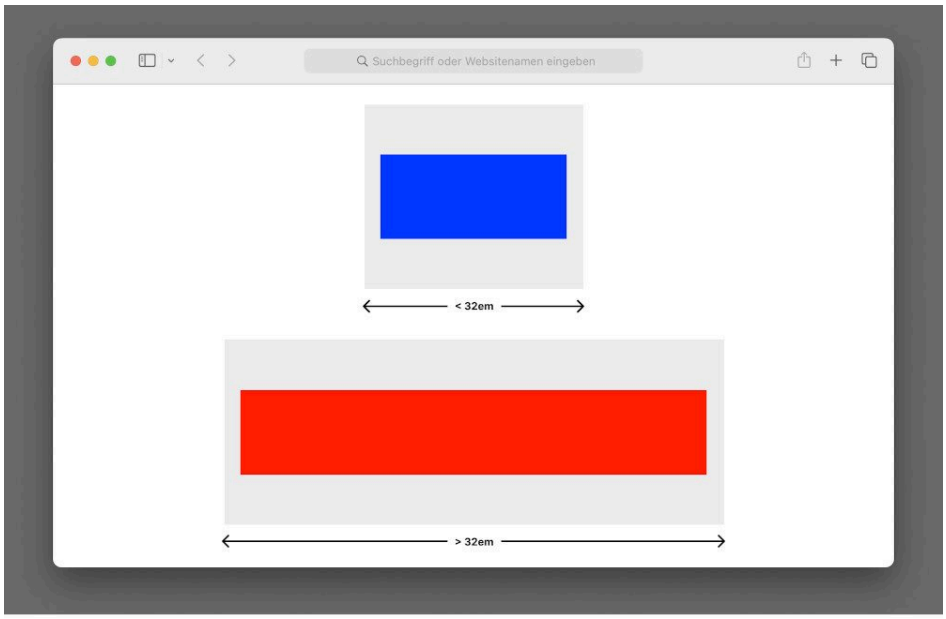
RWD_3 hosted with ❤️ by GitHub view raw

CSS

```
/* Definiere den Container-Typ für die Klasse .container */
.container {
   container-type: inline-size;
}

/* Definiere Standard-Stile für die Klasse .card */
.card {
  background: blue;
}

/* Verwende Container-Queries, um die Hintergrundfarbe der Klasse .card basierend auf der Conta
@container (min-width: 32em) {
        .card {
        background: red;
        }
}
```

RWD_4 hosted with ❤️ by GitHub view raw

Container Query am Beispiel einer Card. Unterhalb von 32em Darstellungsfläche ist die Card blau, wenn sie mehr Platz hat, wird sie rot. Der Umschließende Container hat den Ttyp »Inline Size« erhalten, was normalerweise bedeutet, dass er auf die Breite reagiert.



Visualisierung des Beispiels mit zwei Containern unterschiedlicher Größe sowie enthaltener Card. Der Browser ändert sich in der Größe nicht.

Type-Scaling

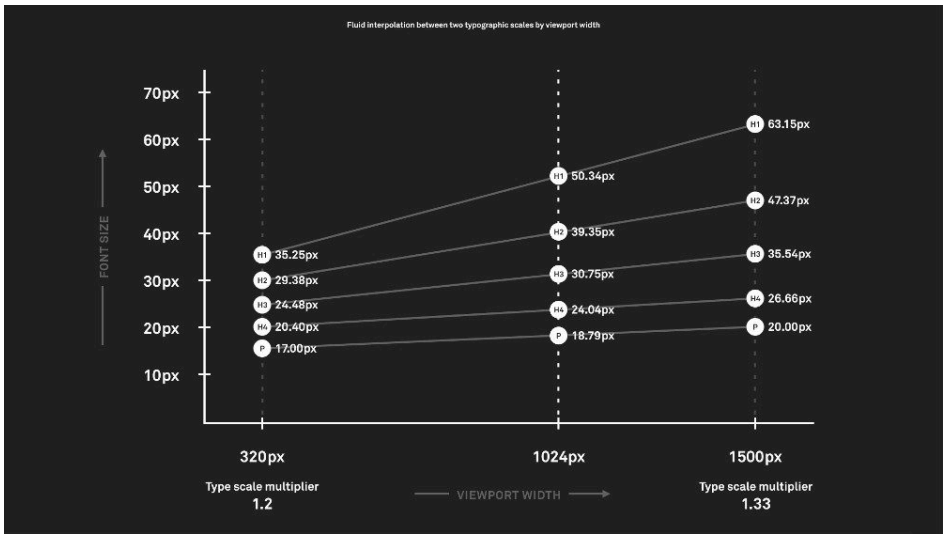
Die Schriftgröße einer responsiven Webseite ist auf dem Smartphone in der Regel kleiner als auf dem Desktop-Computer. Bis etwa 2020 wurden mit Hilfe von Media Queries verschiedene Breakpoints für die Basisschrift festgelegt, und die Schriftgröße wurde an bestimmten Stellen schrittweise erhöht. Alternativ wurden CSS-Hacks mit *calc()* oder ähnlichen Methoden verwendet.

Seit 2020 wird die CSS-Technik *clamp()* genutzt, die es ermöglicht, die Schriftgröße stufenlos zwischen einem Minimalwert und einem Maximalwert zu skalieren. Dabei wird die Schriftgröße in der Regel in vw (Viewport Width) oder moderner in vi (Viewport Inline Size) definiert.

```
body {
        font-size: clamp(1rem, 2vi, 1.5rem); /* Schriftgröße skaliert zwischen 1rem und 1.5rem
```

RWD_5 hosted with ❤️ by GitHub view raw

Einfaches Beispiel für fluides Font-Scaling auf Basis von clamp().



Visualisierung der stufenlosen Schriftskalierung auf der Website von Utopia.

Spacing

Da die meisten Webprojekte Textinhalte enthalten, ist es wichtig, dass die Höhe und Breite der Layoutelemente flexibel sind. Viel bedeutender als exakte Höhen und Breiten sind die Abstände innerhalb (Padding) und zwischen den Elementen (Margin, Gap).

Die meisten professionellen Webseiten verwenden heute eine sogenannte Spacing-Hierarchie. Dabei handelt es sich um eine Palette von vordefinierten Größen, die an verschiedenen Stellen im Layout

mithilfe von CSS-Variablen genutzt werden. Diese Hierarchie basiert in der Regel auf einem 8-Punkt-Raster und wird in den CSS-Einheiten *em* oder *rem* festgelegt. Wenn später die Basisschriftgröße geändert wird, passen sich die Abstände proportional an.

Wenn die Schrift mithilfe von *clamp()* stufenlos flexibel gestaltet ist, reagieren auch alle Abstände stufenlos – ein äußerst schöner Effekt.

HTML

```
<button class="button">Klick mich</button>

  <p class="large-paragraph">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sit
```

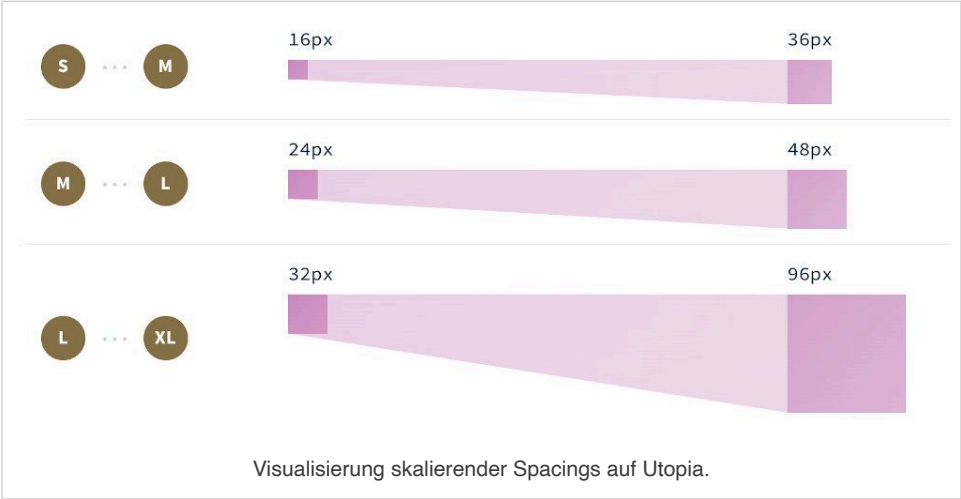
RWD_6 hosted with ❤️ by GitHub view raw

CSS

```
/*
   Definiere globale CSS-Variablen im :root-Selektor, um die Basis-Schriftgröße und Größen des
   Diese Variablen können dann in verschiedenen Teilen des Stylesheets wiederverwendet werden.
*/
:root {
  /* Definiere die Basis-Schriftgröße */
  --base-font-size: 1em;

  /* Definiere die Größen des 8-Punkt-Rasters in em */
  --spacing-0: 0rem;
  --spacing-1: 0.25em;
  --spacing-2: 0.5em;
  --spacing-3: 0.75em;
  --spacing-4: 1em;
  --spacing-5: 1.25em;
  --spacing-6: 1.5em;
  --spacing-7: 1.75em;
  --spacing-8: 2em;
}

/* Setze die Schriftgröße des body-Tags zwischen 1em und 1.5em, skaliert basierend auf der Vier
body {
  font-size: clamp(1em, 2vi, 1.5em);
}

/* Beispielstil für einen Button */
.button {
  display: inline-block;
  padding: var(--spacing-4) var(--spacing-6); /* Verwende das 8-Punkt-Raster für den Innenabsta
  font-size: var(--base-font-size); /* Verwende die Basis-Schriftgröße */
  background-color: #007bff; /* Standard-Hintergrundfarbe */
  color: #ffffff; /* Standard-Schriftfarbe */
  border: none;
  border-radius: 4px;
}

/* Beispielstil für einen größeren Absatz mit mehr Abstand */
.large-paragraph {
  margin-block-end: var(--spacing-8); /* Verwende das 8-Punkt-Raster für den Abstand zum nächst
}
```

RWD_7 hosted with ❤️ by GitHub view raw

Das Online-Projekt Utopia bietet Generatoren zur Erstellung fluider Type- und Spacing-Hierarchien. Es erklärt das Prinzip detailliert und stellt ein Figma-Plugin sowie eine SCSS-Integration bereit.



Visualisierung skalierender Spacings auf Utopia.

Intrinsische Techniken

Responsive Design wird in vielen Bereichen von außen gesteuert. Das Layout ändert sich immer dann, wenn durch Designer oder Geräte festgelegte Punkte (sog. Breakpoints) erreicht sind.

Intrinsic Design hingegen funktioniert anders. Hier gestaltet sich das Layout teilweise von selbst, wobei die Inhalte maßgeblich dafür verantwortlich sind, wann eine gestalterische Änderung erfolgt. Intrinsische Techniken gewinnen im Webdesign zunehmend an Bedeutung, da sie robuste Lösungen für Projekte mit vielen oder sich stark verändernden Inhalten bieten.

Intrinsische CSS Grids

Die Gestaltung von CSS Grids mit Hilfe von *auto-fit/fill* ist eine der ältesten intrinsischen Techniken. Dabei wird nicht die Anzahl der Spalten festgelegt, sondern die maximale Breite einer Spalte. Die Anzahl der Spalten im Grid ergibt sich dann automatisch je nach verfügbarem Platz.

https://gist.github.com/oliverlindberg/a115bd197b8389db647325b5bdd085d1

min/max-content

Die CSS-Einheit *min-* bzw. *max-content* bezieht sich auf die Inhalte eines Elements. Mit *min-content* kann beispielsweise festgelegt werden, dass ein Objekt nicht schmaler werden darf als sein enthaltener Inhalt.

```
.element {
  width: min-content; /* Das Element wird nicht schmaler als sein Inhalt */
}
```

RWD_9 hosted with ❤️ by GitHub view raw

:has()

Mit der CSS Pseudoklasse *:has()*, kann geprüft werden, ob ein Element andere Elemente, Klassen o.ä. beinhaltet. Wenn das der Fall ist, kann in Abhängigkeit dazu eine Layoutänderung angestoßen werden. Eine Teaser-Komponente könnte beispielsweise zweispaltig gestaltet werden, wenn ein Bild enthalten ist.

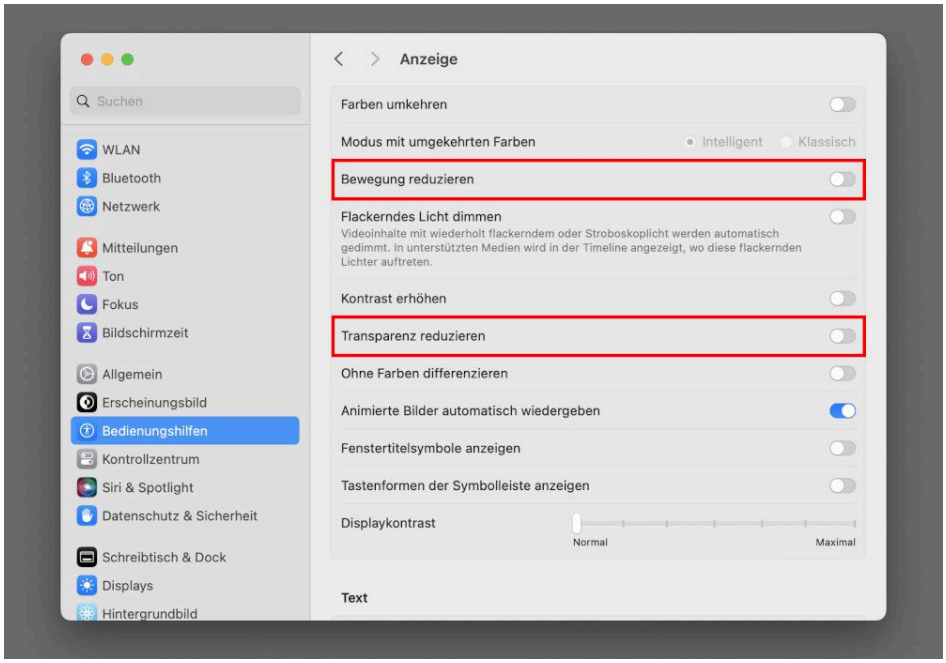
```
.teaser:has(img) {
  /* Wenn ein Bild im Teaser vorhanden ist, ändere das Layout auf zweispaltig */
  grid-template-columns: 1fr 1fr;
}
```

RWD_10 hosted with ❤️ by GitHub view raw

Reaktion auf benutzerdefinierte Einstellungen

Zu guter Letzt reagiert modernes Responsive Design auch auf die Einstellungen, die im Browser oder im Betriebssystem vorgenommen werden. Dabei überlappen sich einige dieser Einstellungen mit den Anforderungen zur Barrierefreiheit. Es bietet sich auch an, die Philosophie des Progressive Enhancement zu verfolgen, indem eine solide Basislösung umgesetzt und dann bei Bedarf anspruchsvollere Techniken hinzugefügt werden, sofern das Feature vom Browser unterstützt und/oder vom Benutzer gewünscht wird.

Die meisten Optionen können mithilfe spezieller Media Queries überprüft werden. Abhängig davon, ob die Einstellung aktiv ist oder nicht, kann der Code entsprechend darauf reagieren.



Anzeigeeinstellungen des Apple Betriebssystems. Rot hervorgehoben sind Einstellungen, die mit Media Queries leicht angefragt werden können.

Reduzierte Transparenz

In den Bedienungshilfen des Betriebssystems kann festgelegt werden, dass Transparenzen nicht gewünscht sind.

```
@media (prefers-reduced-transparency: no-preference) {
  /* Keine Präferenz für Transparenzen, daher wenden wir die Transparenz an */
  .element {
        opacity: .5; /* Transparenz anwenden */
  }
}
```

RWD_11 hosted with ❤️ by GitHub view raw

Reduzierte Bewegung

Ebenfalls in den Bedienungshilfen findet sich eine Einstellung zum Abschalten von Animationen.

```
@media (prefers-reduced-motion: no-preference) {
  /* Keine spezifische Präferenz für Bewegungen, standardmäßig Animation ausführen */
  .element {
        animation: slide-in 1s ease-in-out forwards; /* Führe die Animation aus */
  }
}
```

RWD_12 hosted with ❤️ by GitHub view raw

Dark Mode

Sofern die Website einen sog. Dark Mode gestaltet hat, kann mittels Media Query auf den Farbmodus des Betriebssystems reagiert werden. Das macht u.a. Sinn, wenn die Webseite automatisch den im Betriebssystem verwendeten Modus übernehmen soll.

```
@media (prefers-color-scheme: dark) {
  body {
        background-color: #333; /* Dunkler Hintergrund im Dark Mode */
        color: #fff; /* Helle Schriftfarbe im Dark Mode */
  }
}
```

RWD_13 hosted with ❤️ by GitHub view raw

Responsive Design: Neue Weblayout-Strategien für 2024 – Zusammenfassung

Abschließend lässt sich festhalten, dass Responsive Design heute ein breites Spektrum an Techniken und konzeptionellen Ansätzen umfasst. In den 14 Jahren seit ihrer Entstehung ist diese Technik erwachsen geworden, und zahlreiche bewährte Verfahren haben sich entwickelt.

Moderne Design-Tools wie Figma, Webflow und andere sind zwar in der Lage, Responsive Design wesentlich besser zu gestalten als ältere Tools. Dennoch bleiben viele der beschriebenen Techniken weiterhin dem CSS-Code vorbehalten. Solange diese Tools keinen weiteren großen Schritt in Richtung Browser machen, bleibt erstklassiges Responsive Design eine Disziplin des Frontend-Entwicklungsprozesses.

Titelmotiv des Artikels Responsive Design: Photo by Edho Pratama on Unsplash