Responsive Web Design

Inhalt

- Intro
- Der Viewport
- Media-Queries
- Testing

Setup

Für Übungen nutzen wir die folgende **CodeSandbox** als Startpunkt:

https://codesandbox.io/s/j611l

Die Übungen bauen immer aufeinander auf. Aber keine Angst! Für den Fall, dass bei einer Übung etwas nicht klappt, gibts bei jeder Übung einen Link zur CodeSandbox mit dem aktuellen Stand.

Intro

Man kann nicht wissen, von welchem Gerät aus ein Nutzer auf eine Website zugreift. Daher ist es wichtig, dass Websites responsiv sind, sodass sich die Website dem Gerät anpassen kann, um dem Nutzer immer eine bestmögliche User Experience zu bieten. Dies wird als Responsive Web Design (RWD) bezeichnet.

Source: https://www.mockupworld.co/wp-content/uploads/2019/03/free-apple-multi-devices-mockup-psd.jpg

Der Viewport

Damit eine Website responsiv wird, wird vor allem eines benötigt: Dem Browser muss mitgeteilt werden, dass die Website, auf die er zugreift, tatsächlich responsive ist. Hierfür gibt es einen `meta`-Tag, den man nutzen kann.

Beispiel

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

Verfügbare Attribute

Attribut	Werte	Beschreibung
`width`	Positiver Integer oder der Wert `device-width`	Definiert die Breite des Viewports in `px`
`height`	Positiver Integer oder der Wert `device-height`	Definiert die Höhe des Viewports in `px`
`initial-scale`	Positive Nummer zwischen `0.0` und `10.0`	Definiert das initiale Verhältnis zwischen Device und Viewport
`minimum-scale`	Positive Nummer zwischen `0.0` und `10.0`	Definiert den minimalen Zoom
`maximum-scale`	Positive Nummer zwischen `0.0` und `10.0`	Definiert den maximalen Zoom
`user-scalable`	`yes` (default) oder `no`	Bei `no` kann der User die Website nicht zoomen

Achtung Antipattern 🚫

- `user-scalable=no` anzugeben gilt als Antipattern, da Usern mit einer Sehschwäche die Möglichkeit genommen wird, die Website zu zoomen.
- `maximum-scale` sollte mindestens auf `5.0` (empfohlen von Google) gesetzt werden aus dem gleichen Grund.

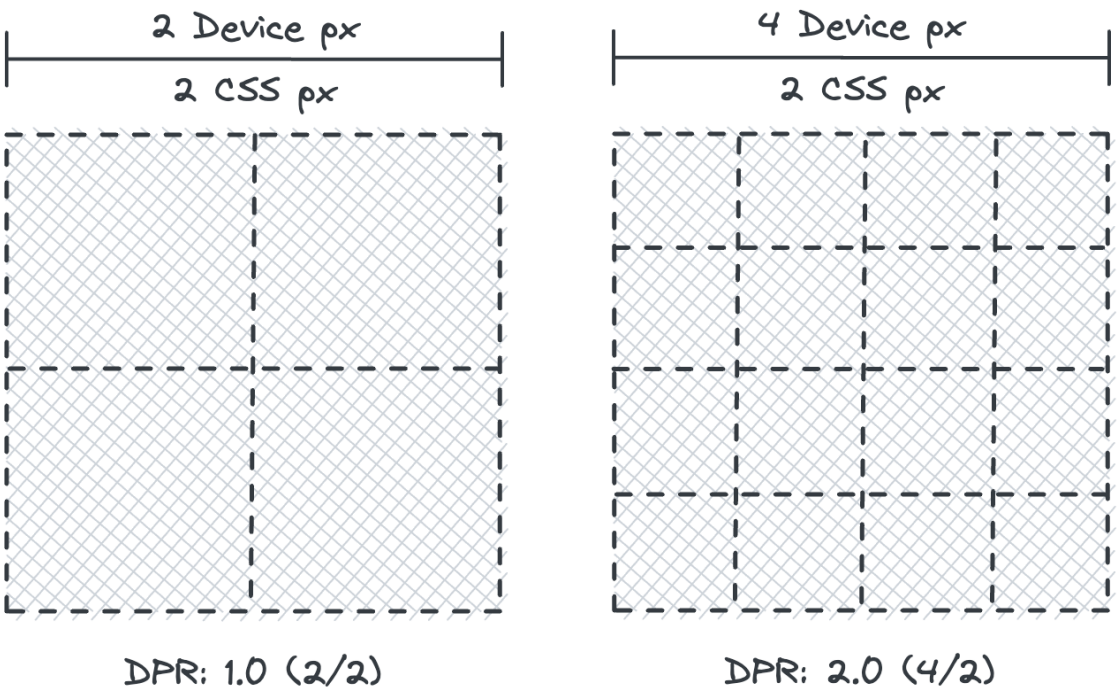
Beispiel mit HTML-Grundgerüst

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Beispiel mit HTML-Grundgerüst</title>
  </head>
  <body>
    <h1>Hello World</h1>
  </body>
</html>
```

Demo 🤯

- Viewport Metatag

Einschub: CSS Pixel und Device Pixel 👀



Mittlerweile haben fast alle mobilen Geräte ein Pixelratio von mehr als 1.0. Das bedeutet grundsätzlich, dass 1 CSS-Pixel 'mehr' ist als nur 1 Device-Pixel.
Ein iPhone 12 Pro hat eine native Auflösung von `1170px` x `2532px`, besitzt jedoch ein Pixelratio von 3.0. Das Betriebssystem legt dann das Pixelratio fest, anhand dessen die CSS-Pixel berechnet werden, und diese Information wird an den Browser weitergegeben. Im Browser haben wir daher 'nur' eine Breite von `390px` zur Verfügung. **Grundsätzlich beziehen sich alle Units/Einheiten, die eine Distanz beschreiben, auf die CSS-Pixel, nicht auf die Device-Pixel**.

Rausfinden des Pixelratio

Um herauszufinden welches DPR euer aktuelles Device hat, könnt ihr folgendes in der Dev-Console eingeben:

```
window.devicePixelRatio
```

Hilfreiche Links

- Understanding the Difference Between CSS Resolution and Device Resolution
- Retina Display Media Query
- Github Gist examples

Media-Queries

Vorwort

Grundsätzlich gibt es fürs RWD zwei Grundprinzipien:

1. Im Layout sollte vermehrt mit `%` gearbeitet werden
2. Durch **Media-Queries** ermöglichen wir, dass CSS-Deklarationen nur dann angewendet werden, wenn bestimmte Kriterien erfüllt sind.

Syntax

Ein Media Query beinnt mit einem `@media`, auch 'at-rule' genannt. Danach folgen ein optionaler `media-type` und 0-n `media-feature`.

```
@media [media-type] [and [(media-feature)]] {
  /*...*/
}
```

Beispiel

```
@media screen and (min-width: 600px) {
  /*...*/
}
```

Was dies genau heisst ist: "Wenn wir uns auf einem `screen` befinden, und dieser mindestens `600px` breit ist, wende das folgende CSS an"

`media-type` und `media-feature` sind beide optional, das heisst, dass die folgenden Media-Queries beide valid sind.

```
@media screen {
  /*...*/
}
```

```
@media (min-width: 600px) {
  /*...*/
}
```

Media Types

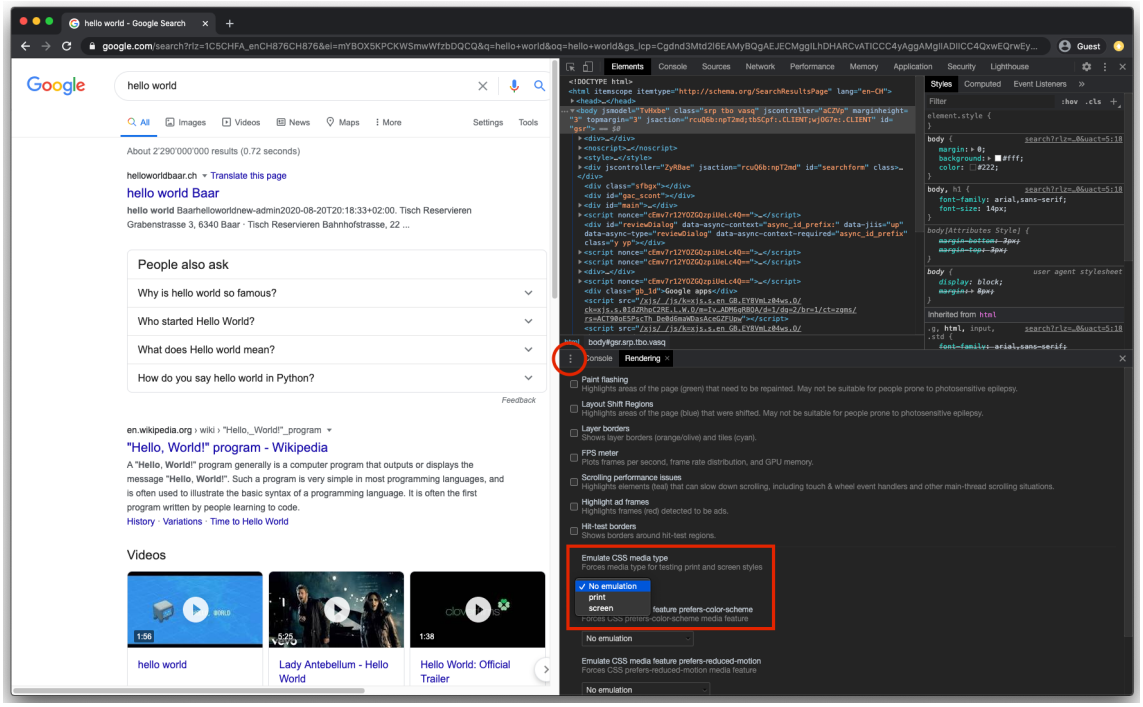
Der Media Type wird genutzt um das Medium zu beschreiben wie die Website konsumiert wird:

- `all` → Default value
- `screen` → Wird angewendet, wenn das Medium einen Bildschirm hat *(normal)*
- `print` → Wird angewendet, wenn die Website gedruckt wird

Weitere media Types in der offiziellen Dokumentation (werden aber nie genutzt).

Einschub: Media Type emulieren 👀

Der Media Type kann emuliert werden, so könnt ihr anhand der Devtool schauen, wie die Website z.B. beim Druck aussehen würde.



Media Features

Fürs Responsive Web Design wird vor allem das *dimension* Feature genutzt.

`width` und `height`

Es können die `width` (genauer Wert), `min-width` und `max-width`, und die `height` (genauer Wert), `min-height` und `max-height` genutzt werden.
Am häufigsten werden jedoch `min-*` und `max-*` verwendet.

Beispiele

```
@media (min-width: 600px) {
  /*...*/
}

@media (max-height: 1000px) {
  /*...*/
}
```

Best Practises ✅

Alle *dimension* Features unterstützen die regulären CSS Units wie `px`, `em`, usw. Es wird empfohlen, dass innerhalb von Media-Queries `em` genutzt wird.

Hilfreiche Links

- Media Query Spezifikation (official)
- Media Features Spezifikation (official)
- `@media` MDN

Logische Operatoren

Der `and` Operator

Man kann den `and` Operator zwischen verschiedenen `media-type` und `media-feature` nutzen, um diese beliebig zu verbinden.

Beispiele

```
@media screen and (min-width: 600px) {
  /*...*/
}

@media (min-width: 400px) and (max-width: 800px) {
  /*...*/
}

@media screen and print and (min-width: 800px) and (max-width: 1200px) {
  /*...*/
}
```

Der oder Operator: `,` (ein Komma)

Der oder Operator wird genutzt um mehrere Media-Queries zu deklarieren, die das gleiche CSS anwenden. Jeder vom `,` getrennte Wert ist dabei ein eigenständiger Media Query und sobald ein Media Query davon zutrifft, wird das CSS angewendet.

Beispiele

```
@media screen, print {
  /*...*/
}

@media screen and (min-width: 1200px), (max-width: 800px) {
  /*...*/
}

@media screen and (max-width: 400px),
  (min-width: 800px) and (max-width: 1200px) {
  /*...*/
}
```

Der `not` Operator

Mit dem `not` Operator kann man einen **ganzen** Media Query negieren. Solange dieser nicht zutrifft, wird das CSS angewendet. Mit diesem Operator muss ein `media-type` angegeben werden, damit dieser valid ist.

Beispiel

```
@media not screen {
  /*...*/
```

```
}

@media not all and (max-width: 800px) {
  /*...*/
}
```

Nesting

Media-Queries können auch ineinander verschachtelt werden.

Beispiel

```
/* Anstatt */
@media (min-width: 800px) and (max-width: 1200px) {
  /*...*/
}

/* Kann dies auch so geschrieben werden */
@media (min-width: 800px) {
  @media (max-width: 1200px) {
    /*...*/
  }
}
```

> ***Note:*** *Das Verschachteln von Media-Queries wird grundsätzlich nicht genutzt, wird aber von Browsern unterstützt*

Demo 🤯

- Nesting Media Queries

Media-Queries im HTML

In einem `<link>` können Media-Queries ebenfalls verwendet werden.

Beispiel

```
<link href="style.css" rel="stylesheet" media="screen and (min-width: 400px)" />
```

Achtung Antipattern 🚫

Dies ist zwar möglich, sollte aber trotzdem nicht verwendet werden. Jede verlinkte CSS-Datei wird heruntergeladen, egal ob der Media Query zutrifft oder nicht. Der Media Query wird dabei erst evaluiert, wenn die Datei heruntergeladen ist.

Demo 🤯

- Link Media Download (Downloads können per Dev-Tools überprüft werden)

Practice 🔥

Öffne diese **CodeSandbox** als Startpunkt.

- [ ] Unter `30em` Breite, sollten alle Container untereinander sein

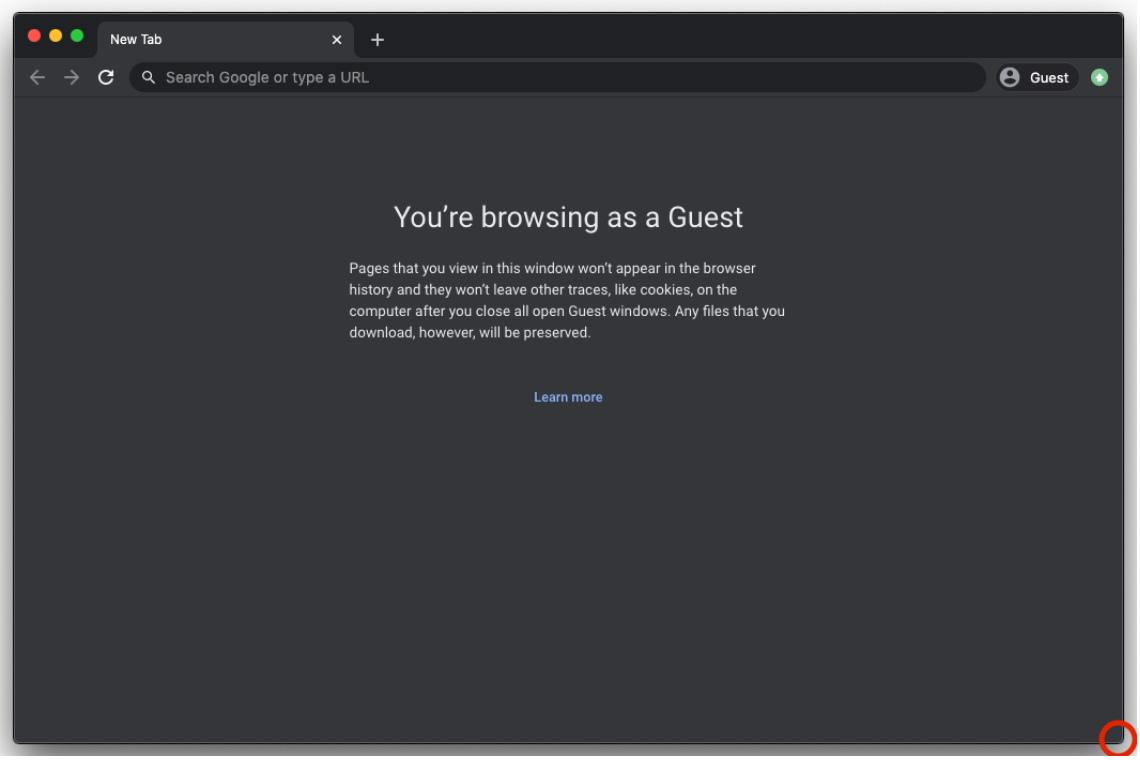
- [ ] Ab `30em` und unter `50em` Breite, sollten die Navigation und die Sidebar jeweils `20%` Breite des Elternelements einnehmen
- [ ] Ab `50em` Breite sollten di Navigation und die Sidebar jeweils `15%` Breite des Elternelementes einnehmen
- [ ] Der `.wrapper` sollte horizontal zentriert sein, sobald dieser seine `max-width` erreicht hat
- [ ] Strukturiere dein CSS, sodass nur `(min-width: xxx)` media Queries genutzt werden

Zeit: ~ 15 min

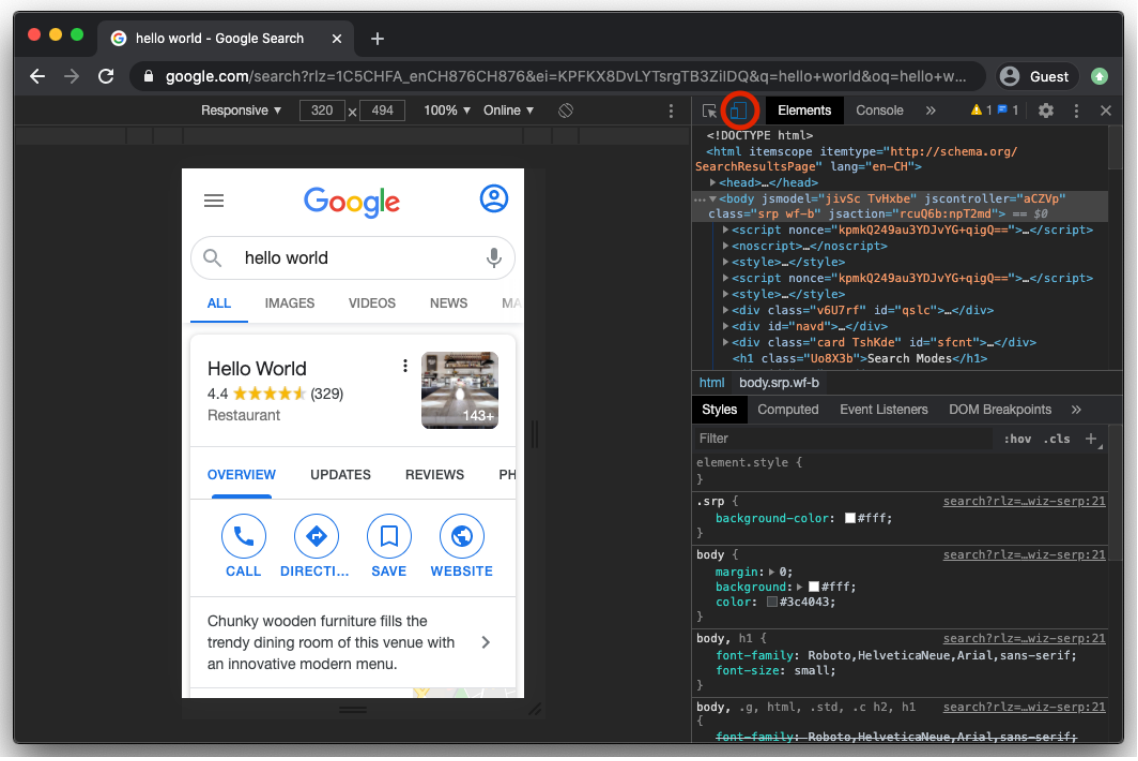
Solution: https://codesandbox.io/s/xfy1p

Testing

Die einfachste Form des Testens von responsiven Websites ist es, den Browser kleiner und größer zu machen.



Die effektivste Form des Testens von responsiven Websites erfolgt jedoch mithilfe der Entwickler-Tools. Dort kann man den Device-Modus aktivieren, um verschiedene Szenarien zu emulieren. Benutzerdefinierte `width` - und `height` -Werte stellen dabei lediglich die Grundlage dar. Man kann damit ein Gerät emulieren, die Pixeldichte verändern und noch vieles mehr.



Beispiel: Chrome Dev-Tools

Practice 🔥

Öffne diese **CodeSandbox** als Startpunkt.

- [ ] Simuliere ein iPhone10
- [ ] Simuliere mit den Dev-Tools ein Pixelratio von mindestens 3.0, damit der Hintergrund der Website grün wird
- [ ] Schalte ein, dass alle Media-Queries angezeigt werden

Zeit: ~ 5 min

Hilfreiche Links

- What is Mobile First Design?
- Complete Guide to Responsive Images!
- Accessible, Simple, Responsive Tables
- Responsive web design tricks and tips

Einschub: Feature-Queries 👀

Mit `@supports` kann man prüfen, ob bestimmte Deklarationen unterstützt werden.

Beispiele

```
@supports (display: grid) {
  .grid-container {
    /* display: grid; can be used here */
```

```
    }
}
```

Hilfreiche Links

- Feature Queries Spezifikation (official)