Kapitel 16 - Teil 4

Responsives Webdesign

Inhalte in diesem Kapitel

- Media Queries verstehen und einsetzen
- Anpassungsfähige Layouts umsetzen
- Formulare und Navigationen responsiv gestalten

Responsiv – Das Wesentliche

- Webseite muss auf allen Geräten funktionieren
- Herangehensweise nennt man responsives Webdesign (RWD)
- Responsiv = antwortend, reagierend = Webdesign reagiert auf Geräte und Viewports
- Grundprinzip:
    - Es gibt 1 HTML-Basis der Webseite und Anpassungen werden über CSS durchgeführt => Media Queries (Medienabfragen)
    - Artikel -> Ethan Marcotte: https://alistapart.com/article/responsive-web-design
      Komponenten, die dazugehören:
        - Flüssiges Layout => Angaben in Prozent
        - Flüssige Bilder
        - Media Queries für Anpassungen
    - Kann aber auch responsive Layouts ohne Media Queries geben!!

Alternativen zum responsiven Webdesign

- Früher separate Webseiten für unterschiedliche Geräteklassen -> oft über Subdomains gesteuert
- Probleme:
    - Aufwand Erstellung und Pflege sehr hoch
    - Häufig abgespeckte Version für Smartphones = nicht alle Inhalte verfügbar
    - Kommen ständig neue Geräteklassen dazu, die man berücksichtigen muss
- Unterschied zwischen responsives und adaptives Layout
    - Responsive:
        - Flüssige Layouts mit Media Queries = Prozentangaben
        - Es existieren unendlich viele Zwischenstufen des Layouts da Breite in Prozent festgelegt ist
    - Adaptiv:
        - Feste Layouts mit Media Queries = Pixel, em oder rem
        - Genau definierte Anzahl an Layouts
- Beispiele: /kapitel_16/adaptive-vs-responsive.html

Bestandteile des responsiven Webdesigns

- Zutaten für responsive Webseiten

Viewport-Meta-Angabe

- Smartphones verkleinern die Webseite so, dass sie komplett auf den Bildschirm passen
- Für responsiv Webseiten stört das Verhalten da passendes Layout vorbereitet ist
- Deaktivierung über:

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

- Mit Google Chrome in Device-Mode anschauen!! Dieser berücksichtigt Viewport-Meta Angaben
  Beispiele: /kapitel_16/
  - mit-viewport-meta.html
  - ohne-viewport-meta.html
- mit `user-scalable=no` oder `maximum-scale` => maximale Skalierung beschränken => Besucher können Seite auf Smartphones nicht vergrößern (unvorteilhaft für Usability)

Klassische Medienangaben

- Media Queries = entscheidende Zutat vom responsiven Webdesign
- Unterschiedliche CSS-Angaben für unterschiedliche Viewportgrößen bzw. bestimmte Formatierungen nur bei ausgewählten Ausgabegeräten
- @media print
  - Gesonderte Formatierung für die Druckausgabe

```
@media print { nav { display: none; } }
```

- Print Stylesheet

```
<link rel="stylesheet" href="druck.css" media="print">
```

  - Alternative für Formatierung für Druckausgaben
- Tipp für Druckausgaben: Linkadresse anzeigen über CSS

```
a[href]::after { content: „ [„ attr(href) „]"; }
```

- Weitere Tipps für Stylesheet für Ausdrucke: https://wiki.selfhtml.org/wiki/CSS/Tutorials/Print-CSS

Media Queries für responsive Webseiten

```
@media screen and (min-width: 560px) { . . . }
```

- Formatierungen in der geschweiften Klammer gelten nur wenn es sich um einen Bildschirm handelt und der Viewport eine Mindestbreite von 560px aufweist

```
@media screen and (min-width: 780px) { . . . }
```

- Formatierungen in der geschweiften Klammer gelten nur wenn es sich um einen Bildschirm handelt und der Viewport eine Mindestbreite von 780px aufweist
- Beispiele: /kapitel_16/mediaqueries_einstieg.html
- **Media Queries gehen von klein bis zu größer. Beim Definieren daher auf die Reihenfolge achten!!!**
- In den Media Queries nur das schreiben, was ab dieser Viewportgröße anders sein soll
- Wenn Angaben für kleine Viewports zuerst stehen => *Mobile-First-Ansatz*
- Gegenteil: *Desktop-First-Ansatz* -> zuerst Layout für große Viewports definieren und dann Anpassungen für kleine Screens
- Heute meist: *Content-First-Ansatz* = Content (Inhalt) ist entscheidende Komponente, Aufbau Media Queries nebensächlich -> Content in allen Viewports muss funktionieren

Breakpoints – woher nehmen oder stehlen?

- Breakpoints = Stelle an denen es Layout- bzw. Formatierungsveränderungen gibt
- Auf Viewportgrößen kein Verlass da ständig neue Gräte auf den Markt kommen
- Gibt keine „am weitesten verbreiteten Viewportgrößen“
- Breakpoints wählen, wie sie zu Inhalt und Layout passen
- Orientierung an gängigen CSS Frameworks möglich (z. B.: Bootstrap *kommt noch genauer*)
    - `min-width: 576px` (kleine Geräte)
    - `min-width: 768px` (mittlere Geräte u Tablets)
    - `min-width: 992px` (große Geräte wie Desktops)
    - `min-width: 1200px` (extrem große Geräte)

Mehr Abfragen

- wichtigsten Angaben bei Media Queries
  - `width` => normalerweise `min-width` oder `max-width`
  - `height` => normalerweise `min-height` oder `max-height`
  - `orientation` => `landscape` für Querformat und `portrait` für Hochformat
  - `hover` => ob Gerät `hover` versteht; Werte dafür: `hover`, `any-hover` oder `none`
  - `pointer` => Eingabegrät wie Maus/Touch. Mögliche Werte: `fine` (z.B. Maus), `coarse` (ungenau, z.B.: Touch- oder Gestensteuerung), `none`: nur Tastatureingabe
  - `resolution` => Auflösung, normalerweise `max-resolution` oder `min-resolution`

```
@media screen and (min-resolutin: 300dpi) { }
```

  - Kombinieren

```
@media (min-width: 700px) and (orientation: landscape) { }
```

Responsive Layouts

Einfaches responsives Layoutbeispiel

- Beispiel Kapitel 2 durch besprechen

Dreispaltiges responsives Layout mit Flexbox

- Beispiel rwd_flexbox.html -> Quellcode durchbesprechen
  - Darstellung große Viewports wird main zu Flexcontainer und order ist anders als auf kleinen Screens
  - 1em entspricht ca 16px
- Besonderheit:
  - Drei Spalten
  - Große Viewports: Navigation links vom Inhalt
    bei einspaltigen Layout nach dem Inhalt
    Reihenfolge Elemente im Quellcode entspricht der bei kleinen Viewports = wichtig das Reihenfolge im Quellcode mit der optimalen Reihenfolge übereinstimmt (Screenreader)

Responsives Layout mit Gridlayout

- Beispiel: rwd_gridlayout.html
- Große Viewport dreispaltig
  mittlere Viewport zweispaltig
  kleine Viewport einspaltig
- Gridlayout und Flexbox sehr ähnlich
- Gridlayout mehr Optionen bei Aufteilung, da Elemente frei auf Raster verteilt werden können und sich beliebige Raster definieren lassen

Komponenten auf responsiv trimmen

- Bilder, Formulare und Navigation benötigen Sonderbehandlung bei responsiven Webdesign

Bilder im responsiven Webdesign

- Einfachste Möglichkeit Bilder responsive zu machen

```
max-width: 100%;
height: auto; (optional)
```

- Bilder maximal so groß wie der Bereich, in dem sie sich befinden

Responsives Formular

- Beispiel: rwd_formular.html
- Formular als Liste
  - `label` und `input` in einem Listenelement
  - für kleine Displays: automatisch untereinander da Blockelemente
  - für große Screens => Media Query für `display: flex;`
  - Boxmodell wird gewechselt damit `width` die Gesamtbreite bezeichnet
  - Für `html` wird Schriftgröße und Art definiert
  - Formular nimmt 90% der Breite ein und wird mit `margin: auto;` zentriert
  - Listenelemente:
    - `padding-left` auf 0 gesetzt
    - Abstand nach unten
    - Aufzählungszeichen werden entfernt
  - Beschriftung
    - Innenabstand versehen u zu Blockelemente für kleine Viewports
  - Formularfelder `input` und `textarea`
    - Bisschen Innenabstand, Rand, volle Breite, abgerundete Ecken
    - Mit `font: inherit;` Elemente erben die bei `html` definierte Schrift
    - Transition gesetzt für Übergang auf Pseudoklassen `:focus` => für leichte Animation der gesetzten CSS Anweisungen
    - Bei `:focus`: mit `outline` Markierung entfernt
  - Button eigenes Styling
    - Innenabstände, Standardrahmen wird entfernt, Farbe bestimmt
  - Für große Screens innerhalb der Media Queries
    - Flex Anweisung für `li`-Elemente => `label` und `input/textarea` nebeneinander
    - Für `label` optimale Größe über flex definiert: Beschriftung dürfen größer werden aber nicht kleiner mit Optimalbreite 25%
    - Für `input/textarea` gleich nur Optimalbreite 75%
    - Button mit `margin-left: auto;` rechts angeordnet

Navigation responsiv machen

- Beispiel: rwd_navigation.html
- Elemente der Navigation werden auf kleinen Viewports untereinander dargestellt => keine elegante Lösung

HTML, CSS & JavaScript-Code für Klappnavigation

- Beispiel: rwd_navigation_klapp.html
- Button
  - Innerhalb des `nav`-Elements
  - Klasse: `navbutton` (für CSS)
  - `aria-expanded="false"` => kennzeichnet das zugehörige Element als geschlossen
  - `aria-controls="navi-liste"` => Bezug zu dem Element hergestellt, das mit Button verknüpft ist
- Ungeordnete Liste
  - `id="navi-liste"`
- `aria`-Attribut gehört zu Accessible Rich Internet Applications (aria) = Standard von W3C => bessere Zugänglichkeit von Webanwendungen für Nutzern von Screenreadern
- JavaScript
  - Hamburger-Icon wird in eine Konstante `hamburger` abgespeichert
  - Klick ruft Funktion `klappNav` auf
  - In Funktion wird Attribut `aria-expanded` auf `true/false` gesetzt => steuert ob Menü angezeigt wird oder nicht
- CSS
  - Button in großen Viewports ausblenden
  - Kleine Viewport
    - Menüpunkte untereinander
    - Button über `position: absolute;` positioniert (geht auch anders)
    - Button `display` Anweisung wird gesetzt auf block
    - Menü wird standardmäßig ausgebelendet und nur bei aria-`expanded="true"` angezeigt