Leseprobe

Das Standardwerk für professionelle Webentwickler! Hier werden alle Ihre Fragen zu Responsive Webdesign beantwortet inkl. vieler Praxisbeispiele. In dieser Leseprobe zeigt Ihnen das Autorengespann, wie Sie ein starres Layout in ein flexibles umwandeln und welche Navigationskonzepte sich für responsive Webseiten anbieten.

Andrea Ertel, Kai Laborenz

Responsive Webdesign –
Konzepte, Techniken, Praxisbeispiele

524 Seiten, gebunden, 3. Auflage, Mai 2017
39,90 Euro, ISBN 978-3-8362-4578-4

www.rheinwerk-verlag.de/4395

Kapitel 2

Schnelleinstieg: Responsive Umsetzung eines fixen Layouts

»A great designer is the one who keeps moving stuff, even when everyone else leaves the room.«

Milton Glaser

Im Rahmen dieses Buches werden wir eine Beispiel-Website »Mobile First« aufbauen. Wir starten damit in Abschnitt 7.2, »Praxisbeispiel: Mobile First«, und demonstrieren daran in den entsprechenden Kapiteln alle möglichen Navigations- und Inhaltstypen. In diesem Kapitel möchten wir ein bisschen vorgreifen und werden mit den Grundlagen des ersten Kapitels eine kleine responsive Website aufbauen. Sie haben ja mit uns im vorherigen Kapitel bereits einen kurzen Blick auf die Umsetzung responsiver Raster geworfen. Nun zeigen wir Ihnen anhand eines Anwendungsbeispiels, wie Sie ein bestehendes festes Design mithilfe der drei Hauptzutaten des responsiven Designs (fluides Raster, flexible Inhalte und Media Queries) in ein anpassungsfähiges Layout umwandeln. Los geht's!

2.1 Die Ausgangslage: Ein grafischer Entwurf mit festen Abmessungen

Stellen Sie sich vor, Sie haben ein Layout von einem Kunden als Photoshop-Vorlage zur Umsetzung erhalten. Sie sehen ein ziemlich typisches dreispaltiges Layout mit einem Menü, einer Inhaltsspalte und einer Marginalie, basierend auf einem Raster, wie in Abbildung 2.1 zu sehen. Die Texte der Website sind noch nicht fertig, sodass Sie mit Blindtext arbeiten müssen, und auch das Logo ist noch in der Abstimmung, und an der Struktur der Website wird noch gearbeitet. Sicher ist aber, dass es mehrere Artikelteaser auf der Startseite geben soll, und die Farben stehen auch schon fest – eine nicht seltene Ausgangssituation.

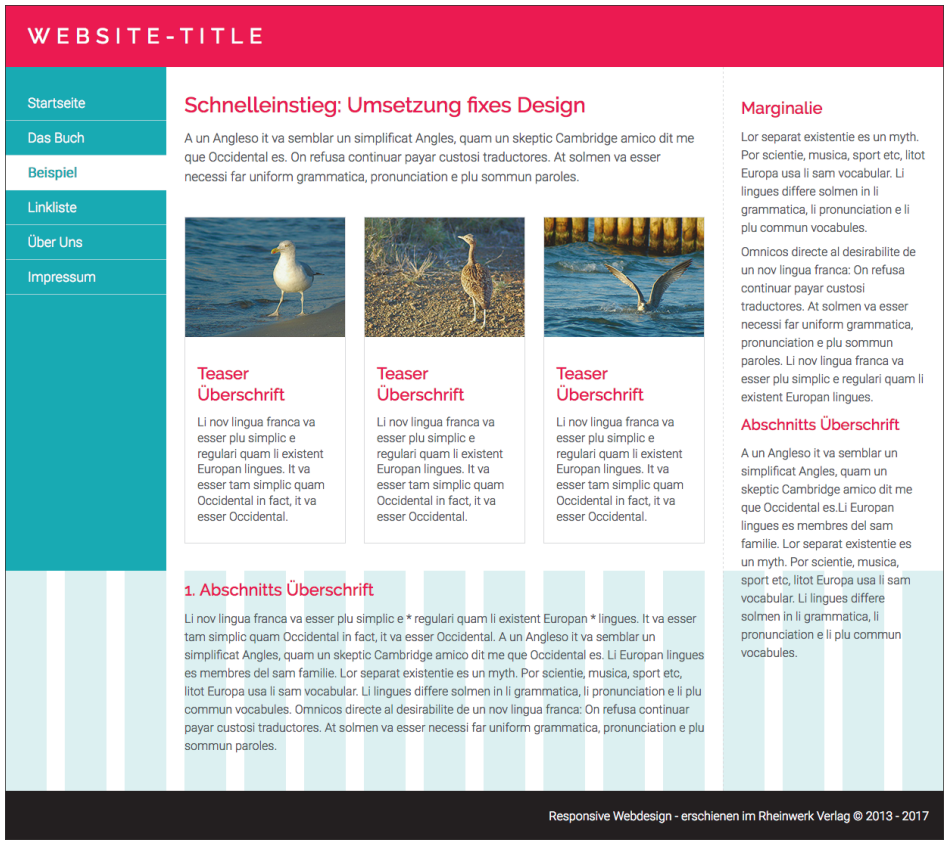

Abbildung 2.1 Die Layoutvorlage mit eingeblendetem Raster im unteren Drittel

Aus der Photoshop-Vorlage ergeben sich folgende Abmessungen, die Sie in Tabelle 2.1 sehen.

Gesamtbreite	1.240 Pixel, verteilt auf 16 Spalten à 55 Pixel mit einem Spaltenabstand von 24 Pixeln
Navigation `.main-nav`	213 Pixel (3 × 55 px + 2 × 24 px)
Hauptinhalt `.main-content`	735 Pixel (9 × 55 px + 10 × 24 px)
Marginalie `.aside`	292 Pixel (4 × 55 px + 3 × 24 px)
Hauptinhalt + Marginalie `.main-wrapper`	1.027 Pixel (13 × 55 px + 13 × 24 px)
Header und Footer gehen über die gesamte Breite.	

Tabelle 2.1 Abmessungen der Hauptbereiche nach der Designvorlage

Bei der Umsetzung verwenden wir die HTML5-Elemente `<header>`, `<main>`, `<aside>`, `<nav>` und `<footer>`. Einen Container `div.page-wrapper` legen wir um alle Inhaltselemente herum und bestimmen über ihn die (noch) fixe Gesamtbreite. `<header>`, `<nav>`, `div.main-wrapper` und `<footer>` liegen auf der gleichen Ebene im DOM. Der `div.main-wrapper` umschließt den eigentlichen Seiteninhalt in den HTML-Elementen `<main>` für den Hauptinhaltsbereich mit den Teasern und `<aside>` für die Marginalie.

Leicht gekürzt sieht der HTML-Code dann so aus:

```
<body>
   <div class="page-wrapper">
      <header class="header">
         <div class="logo"><a href="/">Website-Title</a></div>
      </header>
      <nav class="nav">
         <ul class="main-nav">
            <li><a href="#">Navi-Link</a></li>
            <li><a href="#">...</a></li>
            <li><a href="#">...</a></li>
         </ul>
      </nav>
      <div class="main-wrapper">
         <main class="main-content">
            <h1>Überschrift</h1>
            <p class="teasertext">...</p>
            <section class="teaser-articles">
               <article class="teaser-box">
                  <img src="img/dummy.jpg" alt="...">
                  <div class="box-inner">
                     <h2>Teaser Überschrift</h2>
                     <p>....</p>
                  </div>
               </article>
               <article class="teaser-box">...</article>
               <article class="teaser-box">...</article>
            </section>
            <section>
               <h2>Überschrift</h2>
               <p>...</p>
            </section>
         </main>
         <aside class="aside">
            <h2>Marginalie</h2>
            <p>...</p>
```

```
        </aside>
      </div>
      <footer class="footer">
        <p class="copy">...</p>
      </footer>
    </div>
</body>
```

Listing 2.1 Auszug aus dem HTML5-Quellcode für die Beispielseite

Im ersten Schritt verwenden wir hier bei der Umsetzung zur besseren Anschaulichkeit die festen Größen aus der Photoshop-Designvorlage. Der Aufbau des Layouts erfolgt mit den folgenden CSS-Anweisungen:

```
html {
  box-sizing: border-box;
}
.page-wrapper {
  margin: 0 auto;
  width: 1240px;
}
.header {
  padding: 1.5em 24px;
}
.main-nav {
  width: 213px;
  float: left;
}
.main-wrapper {
  width: 1027px;
  margin-left: 213px;
}
.main-content {
  float: left;
  width: 735px;
  padding: 2em 24px;
}
.aside {
  width: 292px;
  margin-left: 735px;
  padding: 30px 24px;
}
```



```
.footer {
  padding: 0.8em 24px;
}
```

Listing 2.2 Auszug CSS: Layoutgrundelemente (noch) mit Pixelmaßen

Der übergeordnete Container `.page-wrapper` hat eine feste Breite von 1.240 Pixeln. Der Container wird durch `margin: 0 auto;` horizontal zentriert (der Abstand nach oben und unten ist 0, nach rechts und links automatisch). Navigation `.main-nav` und Inhaltscontainer `.main-wrapper` werden nebeneinander gefloatet und mit festen Breiten versehen. Im Inhaltscontainer werden Hauptinhalt (`.main-content`) und die Marginalie (`.aside`) ebenfalls nebeneinander gefloatet.

Um uns das Rechnen etwas zu vereinfachen, haben wir das Boxmodell aller Elemente im HTML-Code vom Standardwert `box-sizing: content-box` auf `box-sizing: border-box` umgestellt. So müssen wir nicht lange `border`, `padding` und `width` addieren, um den tatsächlichen Platzbedarf eines Elements zu ermitteln, sondern können einfach die Breite angeben, die das Element insgesamt einnehmen soll. `border` und `padding` werden dann automatisch von `width` abgezogen. Auf diese sehr nützliche Eigenschaft gehen wir in dem Infokasten »›box-sizing: border-box;‹ for the Web« in Abschnitt 7.2.3 vertieft ein.

Die weiteren Layoutstyles für Farben und Typografie lassen wir hier zunächst unberücksichtigt. Sie finden das gesamte Beispiel im Download-Paket auf der Website zum Buch im Verzeichnis *praxisbeispiele/kap02/bsp_02-01/*.

Das fertige Layout ist jetzt exakt 1.240 Pixel breit, auf größeren Screens wird die Website zentriert, auf kleineren Screens werden Inhalte abgeschnitten, und es erscheinen horizontale Scrollbalken.

2.2 Der erste Schritt: Feste Raster in flexible umrechnen

Um nun aus dem fixen Layout ein fluides zu machen, müssen Sie als Erstes die festen Pixelgrößen in eine relative Einheit umwandeln. In Abschnitt 1.3, »Flexible Raster – Gridsysteme«, haben wir Ihnen die Formel dafür bereits vorgestellt:

$$\text{Breite\_in\_Prozent} = \frac{\text{Breite\_in\_Pixel} \times 100}{\text{Breite\_des\_Elternelements\_in\_Pixel (Kontext)}}$$

Anstelle von *Breite des Elternelements* wird oft auch der Begriff *Kontext* verwendet.

Die Umrechnung der Weiten für die Navigation und den Container, der Hauptinhalt und Marginalie umspannt, erfolgt wie erwartet mit der Gesamtseitenbreite von 1.240 Pixeln als Bezugsmaß. Die errechneten Prozentwerte sind in Tabelle 2.2 gelistet.

Element	Werte in Pixeln (width/padding)	Werte in Prozent (width/padding)
.main-nav	213 px/24 px	17,177419355 %/ 1,935483871 %
.main-wrapper	1.027 px/24 px	82,822580645 %/ 1,935483871 %

Tabelle 2.2 Umrechnung der Pixelwerte in Prozente mit der gesamten Seitenbreite von 1.240 Pixeln als Bezugsgröße

Etwas trickreicher ist die Umrechnung für .main-content und .aside, da diese Elemente innerhalb des div.main-wrapper liegen und dieser darum mit seinen 1.027 Pixeln die Bezugsgröße für die Umrechnung ihrer Weiten bildet (siehe Tabelle 2.3).

Element	Werte in Pixeln (width/padding)	Werte in Prozent (width/padding)
.main-content	735 px/24 px	71,567672833 %/ 2,3369036027 %
.aside	292 px/24 px	28,432327167 %/ 2,3369036027 %

Tabelle 2.3 Umrechnung der Pixelwerte in Prozente mit ».main-wrapper« als Bezugsgröße (1.027 Pixel)

Die krummen Prozentwerte haben Sie ja schon in Kapitel 1, »Denken in flexiblen Strukturen«, kennengelernt. Übernehmen Sie diese in ihrer vollen Länge in Ihre CSS-Anweisungen; nur so lassen sich eventuelle Rundungsfehler beim Nachbau von Pixellayouts vermeiden. Je mehr Größen aufaddiert werden, desto größer ist die Gefahr, dass Rundungsfehler in dem einen oder anderen Browser die Darstellung Ihres Layouts beeinträchtigen. Es gibt einen interessanten englischsprachigen Artikel zum Thema »Rundungsfehler« von John Albin Wilkins (*www.palantir.net/blog/responsive-design-s-dirty-little-secret*), in dem er unter anderem auch eine Methode erklärt, die er als »container-relative floats« bezeichnet und die in dem von ihm entwickelten Framework für das responsive Zen Grids (*http://zengrids.com*) verwendet wird.

CSS-Boxmodells: border-box

Gerade für die Verwendung von prozentualen Werten mit vielen Nachkommastellen, aber auch für den Einsatz von unterschiedlichen Einheiten für die Abstände (px

oder em) und die Spalten (%) im Layoutraster ist die im letzten Abschnitt erwähnte Veränderung des CSS-Boxmodells von box-sizing: content-box auf box-sizing: border-box sehr hilfreich.

Um nun in unserem Beispiel die Änderungen umzusetzen, ersetzen Sie die Pixelwerte mit den errechneten Prozentwerten aus Tabelle 2.2 und Tabelle 2.3. Für die Elemente aus Listing 2.2 sieht das jetzt so aus:

```
html {
   box-sizing: border-box;
}
.page-wrapper {
   margin: 0 auto;
}
.header {
   padding: 1.5em 1.935483871%;
}
.main-nav {
   width: 17.177419355%;
   float: left;
}
.main-wrapper {
   width: 82.822580645%;
   margin-left: 17.177419355%;
}
.main-content {
   float: left;
   width: 71.567672833%;
   padding: 2em 2.3369036027%;
}
.aside {
   width: 28.432327167%;
   margin-left: 71.567672833%;
   padding: 2em 2.3369036027%;
}
.footer {
   padding: 0.8em 1.935483871%;
}
```

Listing 2.3 Auszug CSS: Umstellung der festen Werte auf die errechneten Prozentwerte

Für den umgebenden Container `.page-wrapper` wird die Breitenangabe komplett entfernt – es soll ja ein fluides Raster entstehen, das sich jeder Bildschirmbreite anpasst (siehe Abbildung 2.2).

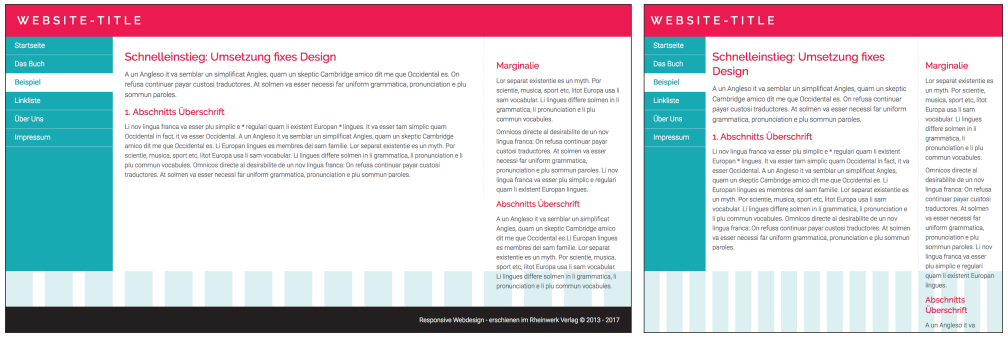

Abbildung 2.2 Fluides Layout in allen Screengrößen (mit eingeblendetem Raster im unteren Bereich)

2.3 Der zweite Schritt zu mehr Flexibilität: Anpassungsfähige Inhalte

Als Nächstes müssen Sie sich um die Inhalte im fluiden Layout kümmern. Für Text ist erst einmal wenig zu tun; normalerweise fließt er einfach in das neue Raster, wie in Abbildung 2.2 zu sehen. Eine Herausforderung im responsiven Webdesign sind jedoch alle Elemente, die grundsätzlich eine feste Größe haben, wie beispielsweise Bilder, Videos oder Inline-Frames. Im Detail erfahren Sie mehr über die Einbindung unterschiedlicher Inhaltselemente in responsive Webseiten in Kapitel 10, »Flexible Bildelemente«, und Kapitel 11, »Mehr flexible Inhalte«.

Schauen wir uns trotzdem hier schon einmal an, wie wir die Teaserboxen mit Bild und Text aus dem Entwurf (siehe Abbildung 2.1) realisieren können:

```
<section class="teaser-articles">
   <article class="teaser-box">
      <img src="img/dummy.jpg" alt="...">
      <div class="box-inner">
         <h2>Teaser Überschrift</h2>
         <p>....</p>
      </div>
   </article>
   <article class="teaser-box">...</article>
   <article class="teaser-box">...</article>
</section>
```

Listing 2.4 HTML-Auszug: Teaserboxen

Für jede Teaserbox verwenden wir hier ein `<article>`-Tag, in dem als Erstes ein Bild zugeordnet wird und darunter in einem `div.box-inner` die textlichen Inhalte folgen. Die drei `article.teaserbox` werden umgeben von einer `section.teaser-articles`, und mittels der CSS-Eigenschaft `flex` teilen wir jeder Teaserbox ein Drittel des Inhaltsbereiches zu. (Ein ausführliches Flex-Beispiel finden Sie in Abschnitt 7.7, »Flexbox-Layout«.)

```
.section.teaser-articles {
   display: flex;
}
.teaser-box {
   flex: 1 33.333333%;
   margin: 0 2.3369036027% 2em 0;
   border: 1px solid #ddd;
   background-color: #fff;
   overflow: hidden;
}
.teaser-box:nth-of-type(3n+3) {
   margin-right: 0;
}
.teaser-box img {
   width: 100%;
   height: auto;
}
.box-inner {
   padding: 1em;
}
```

Listing 2.5 CSS-Auszug: Teaserboxen

Auch hier verwenden wir für das `margin-right` der `.teaser-box` den in Tabelle 2.3 berechneten Prozentwert (2,3369036027 %) für die 24 Pixel als Abstand im Raster. Beim dritten Teaser wird die rechte Margin auf 0 gesetzt. Die Bilder spannen sich durch das `width: 100%` über die gesamte Teaserbreite, und über das `.box-inner` rücken wir den Text gleichmäßig vom Rand der Box nach innen.

2.3.1 Exkurs: Flexible Bilder

Wir haben dem Thema der flexiblen Bilder und Bildelemente ja das ganze Kapitel 10 gewidmet, aber damit verständlich wird, warum wir die Styles für `.teaser-box img` in Listing 2.5 so gewählt haben, hier ein kurzer Exkurs:

Grafikdateien beziehen ihre Größeneigenschaften zunächst aus sich selbst heraus. Ein Bild hat immer eine bestimmte Breite und Höhe, die sich einfach aus der Anzahl

seiner Pixel ergibt. Wenn Sie also ein Bild ohne weitere Angaben in eine HTML-Seite einsetzen, wird das Bild immer in seiner tatsächlichen Größe abgebildet:

```
<img src="beispielbild.png">
```

In HTML ist es dann möglich, dem Element `<img>` Angaben zu Breite und Höhe hinzuzufügen:

```
<img src="beispielbild.png" width="400" height="300">
```

Diese Angaben wiederum lassen sich mittels CSS überschreiben:

```
img {
   width: 200px;
   height: 150px;
}
```

Dass Bilder im responsiven Layout im Gegensatz zu Text erst einmal nicht flexibel sind, liegt genau an diesen Größendefinitionen. Letztendlich muss der Browser Informationen bekommen, wie viel Platz er zum Rendern der Bilder in einer Webseite frei halten muss. Mit den CSS-Anweisungen

```
img {
   width: 100%;
   height: auto;
}
```

legen Sie fest, dass sich die Bildgröße immer an den verfügbaren Platz, den der umgebende Container frei hält, anpasst.

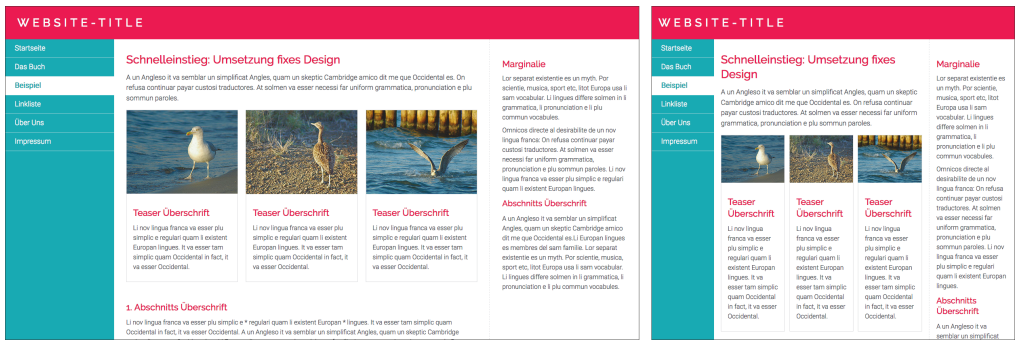

Abbildung 2.3 Fluides Layout jetzt auch mit flexiblen Teaserboxen

Das Ergebnis sehen Sie in Abbildung 2.3. Das Layout passt sich jetzt in alle Screengrößen ein. Die Abbildung lässt aber auch erkennen, dass wir sowohl bei sehr großen Bildschirmen als auch bei schmalen noch nachbessern müssen.

2.4 Der dritte Schritt: Layouts mit Media Queries umschalten

Jedes flexible Layout stößt irgendwann an seine Grenzen, entweder wenn die verfügbare Bildschirmbreite nicht mehr genug Platz für die Inhalte lässt oder aber wenn der Raum zu groß wird. Da die Lesbarkeit von Text ab einer Länge von 60 bis 70 Zeichen deutlich schlechter wird, ist es keine gute Idee, Layouts endlos auszubreiten. Hier kommen jetzt die Media Queries ins Spiel. Die Media Queries sind ein so wichtiges und komplexes Thema, dass wir ihnen ebenfalls ein komplettes Kapitel in diesem Buch gewidmet haben. Seien Sie gespannt auf Kapitel 3, »Die Schlüsseltechnologie Media Queries«.

Um unser Beispiel aber abzurunden und Ihnen eine Idee zu vermitteln, wie leicht sich Layoutänderungen, bezogen auf die Bildschirmgröße, umsetzen lassen, nehmen wir hier schon mal ein paar Regeln vorweg.

2.4.1 Exkurs: Media Queries

Ein Media Query ist eine Mediaabfrage, an das die Ausführung von CSS-Anweisungen geknüpft wird, wenn bestimmte Bedingungen zutreffen. Die am häufigsten verwendete Abfrage ist die nach der Viewport-Weite (Ausgabeweite) für die Darstellung von flexiblen Layouts.

Da die drei Teaser in unserem Beispiel unter einer Seitenbreite von 1.000 Pixeln nebeneinander dargestellt recht gedrängt wirken, ändern wir für kleinere Screens die Anordnung und stellen die Teaser untereinander, wobei Bild und Text innerhalb jedes Teasers nebeneinander angeordnet werden.

Wir unterteilen die CSS-Anweisungen für die Teaser in drei Blöcke:

```
.teaser-box {
   background-color: #fff;
   border: 1px solid #ddd;
}
.box-inner {
   padding: 1em;
}
```

Listing 2.6 CSS für alle Teaser

Im ersten Block stehen die Anweisungen, die das Layout aller Teaser betreffen, in unserem Fall die Hintergrundfarbe, die Rahmen und die Einrückung der Texte in den Boxen.

```
@media screen and (max-width: 999px) {
   .teaser-box {
      overflow: auto;
      width: 100%;
      margin-bottom: 1em;
   }
   .teaser-box img {
      width: 45%;
      float: left;
      margin: 1em;
   }
}
```

Listing 2.7 CSS für Teaser mit Bild und Text nebeneinander

Der zweite Block beinhaltet unser erstes Media Query und wirkt nur auf Bildschirmgrößen kleiner als 1.000 Pixel (also maximal 999 Pixel). Hier setzen wir eine kleinere Bildgröße (45 %) und lassen den Text um die Bilder fließen.

```
@media screen and (min-width: 1000px) {
   .section.teaser-articles {
      display: flex
   }
   .teaser-box {
      flex: 1 33.333333%;
      margin: 0 2.3369036027% 2em 0;
   }
   .teaser-box:nth-of-type(3n+3) {
      margin-right: 0;
   }
   .teaser-box p:last-of-type {
      margin-bottom: 0;
   }
   .teaser-box img {
      width: 100%;
   }
}
```

Listing 2.8 CSS für die drei Teaser für große Bildschirme

Im dritten Block haben wir die Anweisungen für die Teaser auf großen Screens, die Sie schon aus Listing 2.5 kennen, ebenfalls in ein Media Query gesteckt, das ab einer Bildschirmgröße von 1.000 Pixeln (also minimal 1.000 Pixel) wirkt.

Mittels der Zuordnung einer maximalen Weite im `.page-wrapper` verhindern Sie zusätzlich, dass die Website über diese Breite hinaus skaliert wird:

```
.page-wrapper {
   max-width: 1400px;
}
```

Das Ergebnis unserer Layoutstufen mittels Media Queries sehen Sie in Abbildung 2.4.

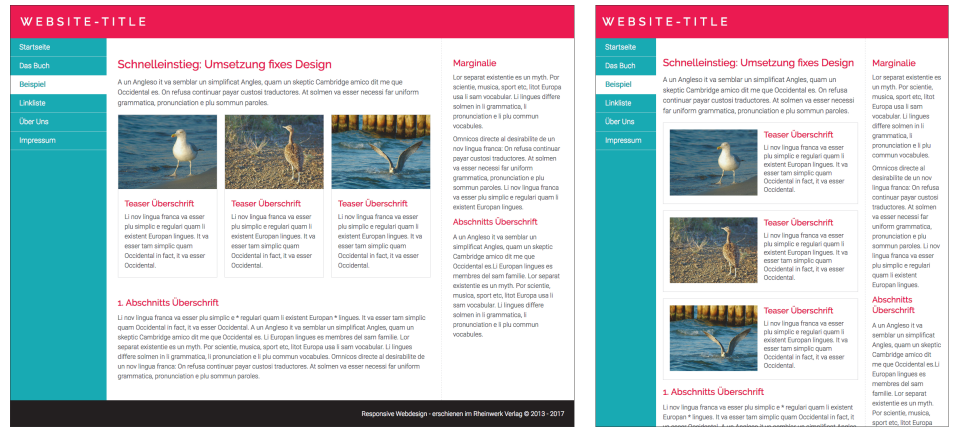

Abbildung 2.4 Das erste responsive Layout mit Weitenbegrenzung auf großen Screens und unterschiedlichen Teaserboxen je nach Viewport – »Hello World!«

Natürlich ist das hier nur ein Anfang und noch kein vollständiges responsives Layout. Als Nächstes bräuchte unsere Website auch eine Layoutstufe, die auf wirklich kleinen Screens gut funktioniert, und dann wartet da noch viel Feinarbeit: Anpassungen von Abständen und Schriftgrößen und jede Menge Testen. Vielleicht haben Sie ja Lust, anhand der vorbereiteten Beispiele ein wenig zu experimentieren. Sie finden die Beispieldateien unter *praxisbeispiele/kap02/bsp_02-02/*.

2.5 Zusammenfassung

In diesem Kapitel haben wir mit Ihnen ein erstes responsives Layout umgesetzt. Mithilfe der drei Zutaten (fluides Raster, anpassungsfähige Inhalte und Media Queries) ist aus einem festen statischen Layout ein anpassungsfähiges entstanden, das nach oben begrenzt und nach unten gestalterisch angepasst wurde. Sie haben erfahren, wie Sie ein flexibles Raster anlegen, Bildinhalte mit einer einfachen CSS-Anweisung responsiv machen können und mit Media Queries unterschiedliche Layouts je nach Größe des Anzeigefensters erzeugen.

Im nächsten Kapitel beschreiben wir die Media Queries, also die Schlüsseltechnologie des *Responsive Webdesigns*, im Detail.

Kapitel 9
Navigationskonzepte

»It doesn't matter how many times I have to click,
as long as each click is a mindless, unambiguous choice.«
Steve Krug

Die Navigation ist einer der wichtigsten funktionalen Bereiche einer Website. Was haben Besucher von interessanten Inhalten im schönsten Design, wenn diese durch eine undurchschaubare Navigation nur schwer gefunden werden? Frustrierte Besucher verlassen eine Website schon nach wenigen Klicks wieder, wenn sie das Gefühl haben, ihrem Ziel nicht nähergekommen zu sein.

Eine Navigation muss sich dem Besucher einfach erschließen. Er braucht eine grobe Übersicht über die gesamte Website, die feiner wird, je weiter er in der Ebenenhierarchie hinabsteigt. Unerwartete Seitensprünge sollten genauso vermieden werden wie tote Enden, von denen es kein Vor und Zurück mehr über die eigene Seitennavigation gibt.

In diesem Kapitel zeigen wir Ihnen, worauf Sie bei responsiven Websites besonders achten müssen, und stellen einige Navigationskonzepte (Patterns) vor. Anhand des Praxisbeispiels erstellen Sie selbst eine flexible Navigation.

9.1 Was macht eine Navigation benutzerfreundlich?

Um eine gut bedienbare Navigation zu erstellen, müssen Sie viele Faktoren berücksichtigen; wir können das Thema hier nur streifen. Wichtige Voraussetzungen für eine gute Navigation sind:

- Semantisches Markup:
  - Menüpunkte sind Aufzählungen und deshalb am besten in einer ungeordneten Liste untergebracht.
  - Eine veränderte Auszeichnung aller Links beim Überfahren mit der Maus oder beim »Durch-Tabben« mit der Tabulatortaste verdeutlicht, welche Links klickbar sind.
  - Verwenden Sie das semantische HTML5-Element `<nav>...</nav>` als Container für Navigationen auf der Seite.

- Verständlichkeit:
  - Die inhaltliche Struktur der Navigation sollte logisch aufgebaut sein, sodass die Besucher verstehen, welche Möglichkeiten diese Website ihnen bietet und wo sie sich befinden. Eine zusätzliche Pfadnavigation (Breadcrumbs) zeigt dem Besucher den Weg an, den er zu der Seite gegangen ist, auf der er sich jetzt befindet.
  - Der aktive Menüpunkt sollte hervorgehoben und nicht verlinkt sein. Die Möglichkeit, die aktuelle Seite über das Menü noch einmal neu zu laden, ist unnötig und verwirrend. Generell sind Links auf die aktuelle Seite problematisch, da beim Klicken keine Veränderung stattfindet. Ein Neuaufruf der aktuellen Seite wirkt daher wie ein Fehler.
- Zugänglichkeit:
  - Die Navigation muss auch ausschließlich per Tastatur bedienbar sein.
  - Farben und Schriften müssen einen ausreichenden Kontrast aufweisen, um auch für Menschen mit Sehschwäche erkennbar zu sein.
  - Verwenden Sie WAI-ARIA-Rollen `<nav role="navigation">` zur Auszeichnung des Elements, z. B. für Screenreader.
  - Aktive Elemente für Zeigegeräte und Touchbedienung müssen Minimalabmessungen haben, um vernünftig »getroffen« zu werden. Dabei kommt es nicht nur darauf an, dass man das Element treffen kann, sondern auch auf das subjektive Empfinden des Nutzers (»Das ist so klein …«).
  - Eine Navigation mit mehreren Ebenen sollte beim Ausklappen der Unterebenen nicht aus dem Fokus des Benutzers rutschen.
- Robuste Struktur:
  - Um ein zu schnelles oder unbeabsichtigtes Auf- und Zuklappen eines Dropdown-Menüs zu vermeiden, sollten Verzögerungen per JavaScript eingebaut werden, die das Auf- und Zuklappen auslösen. Dadurch klappt ein Menü auch bei einem versehentlichen Überfahren mit der Maus nicht sofort auf, und wenn die Maus einmal kurz von der aktiven Fläche rutscht, ist es nicht gleich wieder geschlossen.
  - Die Navigation muss auch ohne JavaScript funktionieren. Falls das JavaScript aus irgendeinem Grund nicht geladen werden kann, muss es ein funktionales Fallback geben.

9.2 Benutzerfreundliche Navigation für mobile Geräte

Für das Responsive Design kommen jetzt noch einige Komponenten für eine gute Bedienbarkeit auf einem kleinen Touchscreen hinzu. Worin unterscheidet sich nun

eine Navigation auf mobilen Geräten von der auf dem Desktop, und was sind die wichtigsten Punkte, die Sie beachten sollten?

9.2.1 Freier Blick auf die Website

Die Navigation sollte nicht den Blick auf das Wesentliche versperren. Hier gilt *Content First*. Wenn die Navigation immer im sichtbaren Bereich liegt und einen Großteil (oder mehr) des Bildschirms ausfüllt, geht das auf Kosten der eigentlichen Inhalte, die dadurch in den Hintergrund gedrängt werden. Nach dem Klick auf einen Menülink ändert sich im schlimmsten Fall nichts im sichtbaren Bereich des kleinen Screens. Der Nutzer bekommt so keinerlei Feedback über die Auswirkungen seiner Aktion. Nur sehr neugierige Nutzer (oder verzweifelt Suchende) scrollen wahrscheinlich nach unten, in der Hoffnung, das Gesuchte zu finden. Aber auch Besucher, die den neuen Inhalt finden, werden es negativ beurteilen, wenn sie bei jedem Navigieren aus dem Menübereich herausscrollen müssen.

Darum besteht bei allen Menüs, die aus so vielen Menüpunkten bestehen, dass sich diese nicht mehr in ein oder zwei Zeilen im Header zusammenfassen lassen, ein spezieller Handlungsbedarf. Das gilt auch für Menüs mit mehreren Ebenen. Es haben sich mittlerweile viele Umsetzungsvarianten für die Darstellung von komplexen Menüs auf kleinen Screens etabliert. Wir stellen Ihnen etwas später in diesem Kapitel einige davon vor.

9.2.2 Ausreichend große Klickflächen für die Touchbedienung

Für die Bedienung mit den Fingern ist es wichtig, dass klickbare Elemente ausreichend groß sind. Buttons oder Links sollten nicht zu dicht nebeneinanderliegen, um zu vermeiden, dass der Benutzer versehentlich den Link daneben anklickt. Ausführlicher haben wir das Thema ja schon in Abschnitt 5.3.3, »Size matters: Ziele für Touchevents«, besprochen.

9.2.3 Umgang mit Menüs mit mehreren Ebenen

Ein anderer Punkt, der schon beim Konzept der Website berücksichtigt werden muss, ist die Zugänglichkeit einer zweiten und vielleicht sogar dritten Menüebene über einen Touchscreen. Eine Dropdown-Navigation, die per Maus-Hovereffekt oder mit dem Fokuseffekt der Tastatur gut funktioniert, ist mittels Touchscreen nicht mehr bedienbar – es gibt kein Hover auf Touchscreens (siehe auch Abschnitt 5.3.4, »Es gibt kein Hover auf Hawaii, und ein Klick ist kein Touch«). Hier müssen Sie für die mobile Navigation gänzlich umdenken. Der Touchklick auf einen Menüpunkt muss dann erst einmal sein Untermenü aufklappen, und erst der zweite Klick auf denselben Menüpunkt darf dann den Menüpunktlink selbst auslösen. Ein Konzept für ver-

schachtelte Menüs könnte beinhalten, dem Menüpunkt, auf den eine Unternavigation folgt, selbst keinen Link (also keine eigenen Seite) zu geben und ihn nur über das Auf- und Zuklappen des Untermenüs zu steuern. Aber auch Menüs, bei denen der übergeordnete Menüpunkt selbst zu einer Seite führt, lassen sich mit JavaScript-Helfern wie DoubleTabToGo.js oder Flexnav umsetzen, wie wir sie Ihnen in Abschnitt 9.9, »Multilevel-Menüs«, vorstellen.

9.2.4 Navigationstypen für mobile Geräte mit Touchscreen

Es gibt ganz unterschiedliche Ansätze für die Navigation auf mobilen Geräten. Findige Entwickler haben diverse JavaScript-Plug-ins für unterschiedliche Lösungen gebaut und diese über Github oder ihre eigenen Projektseiten veröffentlicht und dadurch uns und Ihnen zur Verfügung gestellt. Die Implementierung der skriptbasierten Plug-ins ist in der Regel auch ohne tiefere JavaScript-Kenntnisse sehr einfach. Die unterschiedlichen Varianten haben ihre Vor- und Nachteile, und für welchen Navigationstyp Sie sich letztendlich entscheiden, sollten Sie von den Gegebenheiten Ihres Projekts abhängig machen:

- Gibt es viele Menüpunkte oder sehr lange Bezeichnungen?
- Hat Ihr Menü mehrere Ebenen?
- Gibt es mehrere Menügruppen wie Zielgruppen-, Haupt- und Servicenavigationen?

Wichtig für alle Navigationstypen, ...

... die auf JavaScript basieren, ist ein funktionales Fallback: Auch wenn die Skripte nicht geladen oder verarbeitet werden können, sollte eine Navigation durch die Website möglich sein.

Schauen wir uns einmal verschiedene Menütypen an.

9.3 Wenige Menüpunkte am oberen Rand

Die denkbar einfachste Variante eines Menüs besteht aus nur wenigen kurzen Menüpunkten, die am oberen Rand neben oder unter dem Logo angeordnet sind, somit kaum extra Platz des Screens beanspruchen und das Augenmerk des Besuchers auf den Inhalt der Seite lenken. Das ist eine gute Ausgangssituation für ein benutzerfreundliches Menü. Es ist bei jedem Seitenwechsel im sichtbaren Bereich. Der Nutzer weiß immer, wo er sich befindet, und der Inhalt einer Seite liegt immer im Fokus. Diese Variante macht kaum zusätzliche Arbeit und ist auf allen Geräten voll funktionsfähig.

Das kompakte Menü am oberen Rand der Website von Trent Walton (siehe Abbildung 9.1) nimmt auch auf kleinen Screens keinen zusätzlichen Platz ein.

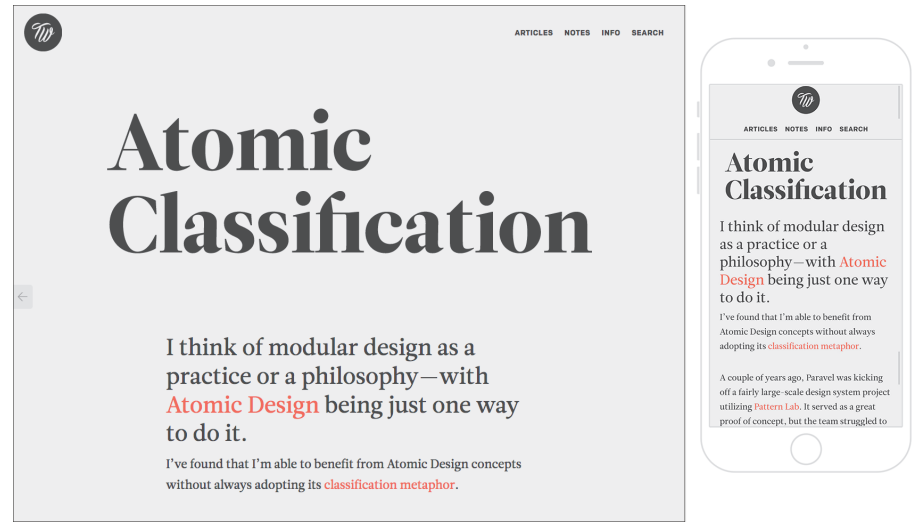

Abbildung 9.1 Das kompakte Menü am oberen Rand der Website von Trent Walton nimmt auch auf kleinen Screens keinen zusätzlichen Platz ein (http://trentwalton.com).

9.3.1 Praxisbeispiel: Mininavigation – wenige Menüpunkte am oberen Rand

Kehren wir zurück zu dem Stand des Praxisbeispiels aus Kapitel 7, »Responsive Layout-Patterns«. Dort hatten Sie eine Basisversion und drei weitere Layouts für unterschiedliche Viewports angelegt. Die Navigation in der Basisversion ist so angelegt, dass jeder Menüpunkt über die volle Gerätebreite ausgedehnt wird. Weitere Menüpunkte reihen sich ein und vergrößern das Menü in der Höhe (siehe Abbildung 9.2).

Abbildung 9.2 Ausgangssituation Praxisbeispiel für alle folgenden Beispiele: Der Platz, den das Menü hier auf dem Smartphone einnimmt, fehlt dem Inhalt.

Ausgangssituation für alle Beispiele in diesem Kapitel

Die Informationen über das Markup des Praxisbeispiels finden Sie abgedruckt in Kapitel 7, »Responsive Layout-Patterns«, und im Download-Paket sowie auf der Website zum Buch unter *praxisbeispiele/kap09/bsp_09-00/*. Auf diesem Code bauen alle weiteren Beispiele dieses Kapitels auf, wenn nichts anderes in den einzelnen Abschnitten vermerkt ist.

Gehen wir in diesem Beispiel davon aus, dass das Menü überschaubar ist und nur drei Menüpunkte hat, bietet es sich an, schon in der Basisversion das Menüstyling der Tablet-Version zu verwenden. Auf kleinen Screens breiten sich die drei Menüpunkte gleichmäßig über den gesamten Viewport aus, ab 640 Pixeln Breite floaten die drei Menüpunkte mit gleichbleibendem Abstand (`padding`) nebeneinander (siehe Abbildung 9.3).

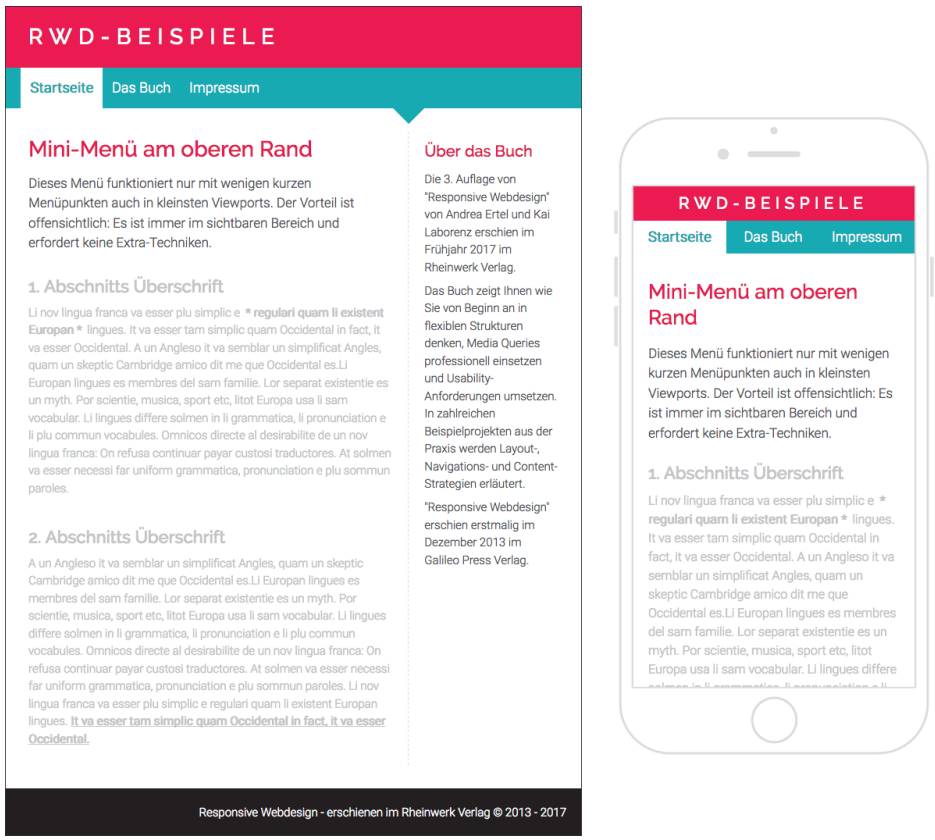

Abbildung 9.3 Wenige Menüpunkte finden Platz am oberen Rand.

Für die Ausdehnung der Navigation über die gesamte Breite verwenden wir ein *Flexbox-Layout*. Setzen Sie die ungeordnete Liste (<ul>) (`.main-nav`) dafür auf `display: flex` und die einzelnen Listenpunkte auf `flex: 1`. Zusammen mit dem `text-align: center` und einem `padding` in der Höhe reicht das schon. Abweichend von der Ausgangsdatei unseres Praxisbeispiels werden folgende Styles für die Navigation in der Basisversion gesetzt:


```
.main-nav {
   margin: 0;
}
.main-nav a:link {
   text-decoration: none;
}
.main-nav > li > a,
.main-nav > li > b {
   display: block;
   font-size: 1.8rem;
}
```

Das Flexbox-Layout begrenzen wir auf die Basisversion und verwenden hierfür ein `max-width`-Media Query bis 639 Pixel (39,938 em).

```
@media screen and (max-width: 39.938em) {
   .main-nav {
      display: flex;
   }
   .main-nav > li {
      flex: 1;
      text-align: center;
   }
   .main-nav > li > a,
   .main-nav > li > b {
      padding: 0.4em 0;
   }
}
```

In dem Media Query ab 640 Pixeln Weite (40 em) für die Tablets werden die einzelnen Listenpunkte nebeneinander geflotatet (`float: left`) und die seitlichen Abstände entsprechend angepasst:

```
@media screen and (min-width: 40em) and (max-width: 63.938em) {
   .main-nav {
      display: block;
      padding: 0 2rem;
      overflow: hidden;
   }
   .main-nav > li {
      float: left;
```

```
      display: block;
   }
   .main-nav > li > a,
   .main-nav > li > b {
      padding: 0.7em 1.2rem;
   }
}
```

Listing 9.1 Auszug aus der Datei layout.css für die Mininavigation

Dieses Praxisbeispiel finden Sie im Download-Paket unter *praxisbeispiele/kap09/bsp_09-01*.

9.4 Lange Menüs kompakt anordnen

Besteht das Menü nicht nur aus wenigen Punkten, kann man durch geschickte Gruppierung der Menüpunkte neben- und untereinander versuchen, die Navigation möglichst kompakt zu halten, um dem Inhalt noch genügend Raum zu geben. Das Regent College (siehe Abbildung 9.4) arbeitet mit einer Kombination aus kompakter, aber übersichtlicher Zusammenfassung der oberen Menüpunkte und einem zusätzlichen ausklappbaren Menü darunter.

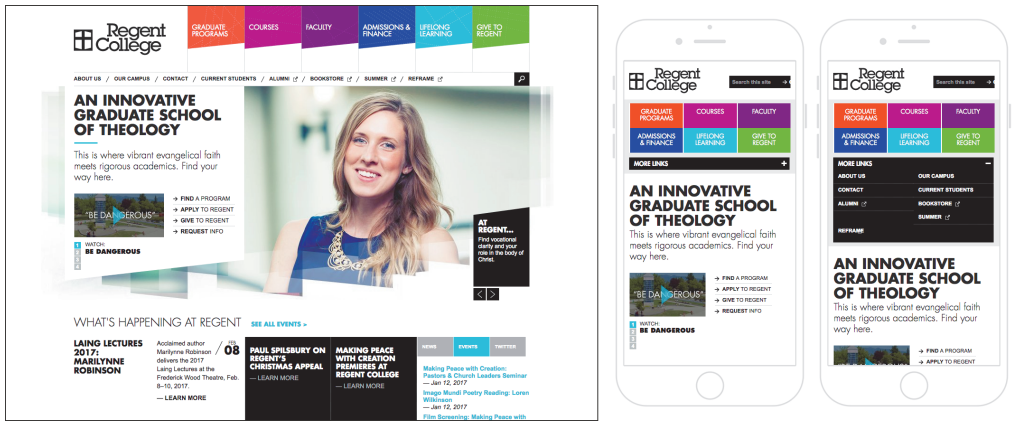

Abbildung 9.4 Kompakte Menüanordnung auf kleinen Screens beim Regent College (www.regent-college.edu)

Eine andere Variante, bei der ein Menü auch auf schmaleren Screens immer nur eine Zeile einnimmt, ist das sogenannte *Priority-Menü*. Hier sind immer nur so viele Menüpunkte sichtbar, wie nebeneinander Platz finden, alle anderen werden versteckt und erst nach Klick auf den danebenstehenden Menübutton eingeblendet (siehe Abbildung 9.5). Sie sehen, warum dieser Menütyp Priority-Menü genannt wird.

Hier muss abgewogen werden, welche Navigationspunkte so wichtig sind, dass sie immer sichtbar sein sollten, und bei welchen man auf schmaleren Screens in Kauf nehmen will, sie erst nach einer Nutzeraktion einzublenden.

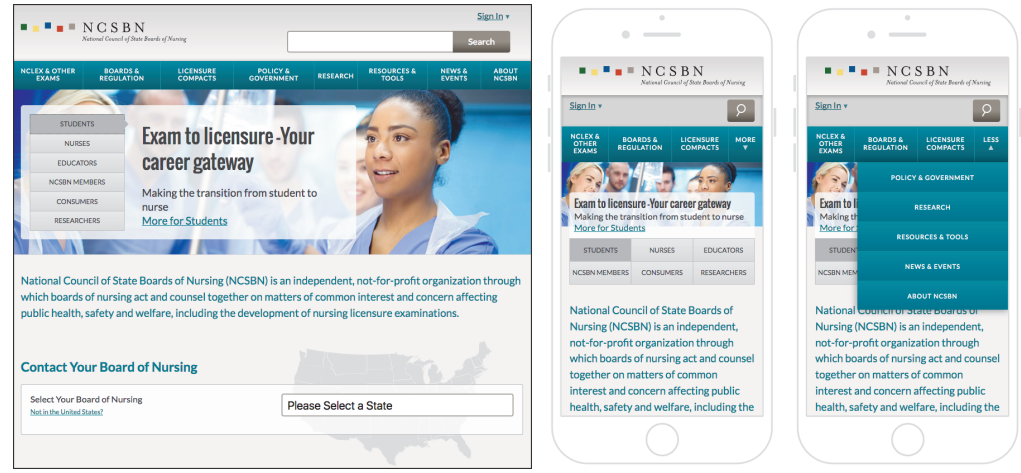

Abbildung 9.5 Die Website des National Council of State Boards of Nursing (www.ncsbn.org) verwendet ein Priority-Menü.

9.4.1 Praxisbeispiel: Priority-Menü

Wir haben das Priority-Menü einmal in unser Praxisbeispiel integriert. Für die Umsetzung stehen Ihnen unterschiedliche Plug-ins zur Verfügung, es gibt z. B. die *Priority-Navigation* von Gijs Rogé (*https://github.com/gijsroge/priority-navigation*) und das Plug-in *GreedyNav* von Luke Jackson (*https://github.com/lukejacksonn/Greedy-Nav*). Wir verwenden in unserem Beispiel das einfache jQuery-Plug-in GreedyNav und binden jQuery und die Datei *greedynav.js* wieder am Ende unserer HTML-Datei ein. Jetzt muss noch das HTML der Navigation etwas angepasst werden.

```
<nav class="nav greedy" role="navigation">
   <ul class="main-nav links">
      <li><a href="#">Startseite</a></li>
      <li><a href="#">...</a></li>
      <li><a href="#">...</a></li>
      ...
   </ul>
   <button aria-controls="menu">Mehr</button>
   <ul class="hidden-links hidden" aria-hidden="true"></ul>
</nav>
```

Listing 9.2 Die Navigation für GreedyNav bekommt eine extra Klasse ».greedy«, einen Button hinter der UL und eine weitere UL, die die versteckten Listenpunkte enthält.

Das <nav>-Element bekommt die Klasse .greedy zugewiesen, damit das JavaScript greift, und hinter der ul.main-nav.links fügen wir einen Button ein. Per JavaScript werden die Navigationspunkte, die nicht in den sichtbaren Bereich passen, in die versteckte neue ul.hidden-links hinter den Button verschoben. Der Klick auf den Button blendet dann die ausgeblendeten Navigationspunkte wieder ein, wie in Abbildung 9.6 zu sehen.

Nun müssen natürlich noch Button und Dropdown-Menü unserem Layout angepasst werden. Das <nav>-Element bekommt ein position: relative, damit wir den Button darin absolut positionieren können.

```
.nav {
   position: relative;
   padding-right: 100px;
}
.nav button {
   position: absolute;
   top: 0;
   height: 100%;
   right: 0;
   padding: 0 2rem;
   border: 0;
   outline: none;
   background-color: #f57c00;
   color: #fff;
   cursor: pointer;
}
.nav button::before {
   content: '';
   position: absolute;
   border-style: solid;
   border-color: #f57c00 transparent;
   width: 0;
   top: 13px;
   left: -15px;
   bottom: auto;
   border-width: 15px 15px 15px 0px;
   border-color: transparent #f57c00;
}
.nav button::after {
   content: attr(count);
   position: absolute;
   width: 3rem;
```



```
   height: 3rem;
   left: -0.5rem;
   top: 1.8rem;
   text-align: center;
}
```

Listing 9.3 CSS-Auszug: Styles des Priority-Menü-Buttons mit Zähler für die versteckten Menüelemente

Das button::before links neben dem Button stylen wir wie einen Pfeil (oder ein Dreieck), der auf das Menü zeigt, und im button::after wird der Wert des count-Attributs, das per JavaScript in den Button geschrieben wird, als content sichtbar gemacht und auf dem Button positioniert.

Der nächste Absatz beschreibt die Liste der versteckten Menüelemente, die als Dropdown-Menü aufgeklappt werden.

```
.nav .hidden-links {
   position: absolute;
   right: 0;
   top: 100%;
   background-color: #f8b255;
}
.nav .hidden-links li {
   display: block;
}
.nav .hidden-links li a {
   padding: 0.5em 2rem;
   display: block;
   text-decoration: none;
   color: #fff;
}
.nav .hidden-links li a:hover {
   color: #fff;
}
.nav .hidden {
   visibility: hidden;
}
```

Listing 9.4 CSS-Auszug: Liste der versteckten Menüelemente

Den vollständigen Quellcode dieses Beispiels finden Sie im Download-Paket unter *praxisbeispiele/kap09/bsp_09-02/*.

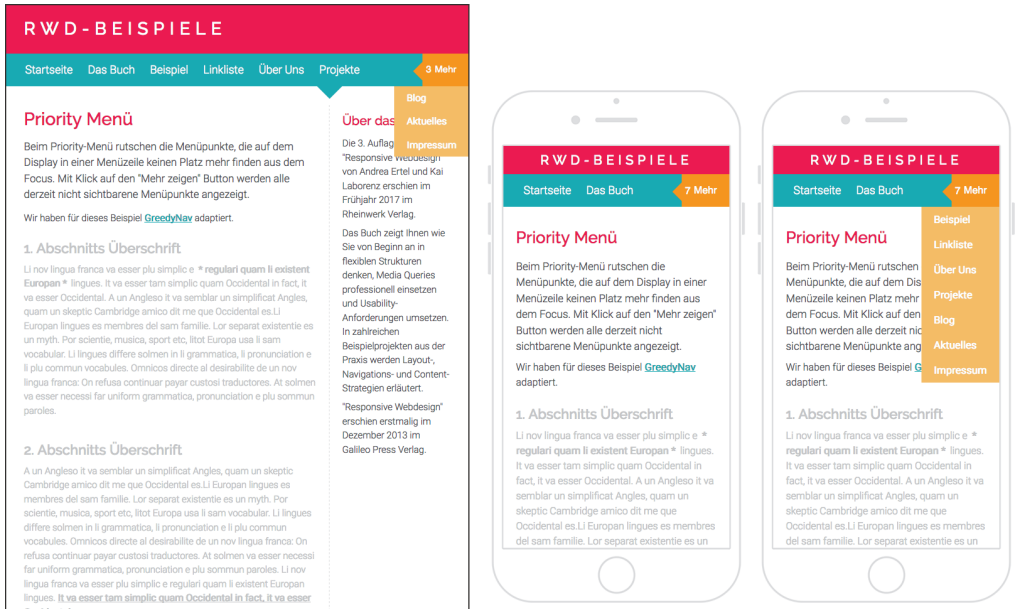

Abbildung 9.6 Beim Priority-Menü werden immer nur so viele Menüpunkte dargestellt, wie auf dem Screen Platz finden, und die anderen über einen Menübutton als Dropdown-Liste eingeblendet.

9.5 Select-Menü

Eine seltenere Lösung zur Darstellung des Menüs in kleinen Viewports ist die Umwandlung des Menümarkups mittels JavaScript in ein Select-Menü. Menüpunkte lassen sich so verstecken und trotzdem leicht erreichen. Damit diese Menüs von den Website-Besuchern auch als solche erkannt werden, ist für diese Art der Navigation auf jeden Fall eine sehr gute Kennzeichnung notwendig. Das Stylen von `<select>`-Elementen ist nur in einem begrenzten Maße möglich, was eine Integration in das Design erschweren kann. Abbildung 9.7 zeigt die Website von Retreats4geeks. Das Menü, das auf der Desktopversion am Baum angepinnt ist, hängt in der Smartphone-Version in einer Select-Box an einem Ast. Auf mobilen Geräten werden Select-Menüs durch betriebssystemspezifische Elemente ersetzt.

Abbildung 9.7 Bei Retreats4geeks rutscht das Menü für kleine Viewports in eine Select-Box (http://retreats4geeks.com).

9.5.1 Praxisbeispiel: TinyNav – Select-Menü

Zur Erstellung eines solchen Select-Menüs hat Viljami Salminen ein kleines jQuery-Plug-in namens TinyNav (*http://tinynav.viljamis.com*) veröffentlicht. Sie finden das folgende Beispiel im Download-Paket auf der Website zum Buch unter *praxisbeispiele/kap09/bsp_09-03/*. Um die Navigationsleiste des Praxisbeispiels in kleinen Viewports als Select-Menü darzustellen, haben wir die JavaScript-Datei von *tinynav.js* unter der jQuery-Einbindung vor dem schließenden `<body>`-Tag untergebracht (siehe Listing 9.5). Zur Initialisierung haben wir die Funktion `tinyNav()` an die ID `#navMain` unseres Hauptmenüs geknüpft. Mit dem Setzen von `header: 'Menu'` wird ein zusätzliches `<option>`-Tag `Menu` zur Kennzeichnung der Navigation erstellt.

```
<script src="js/jquery.js"></script>
<script src="js/tinynav.js"></script>
<script>
   $(function () {
      $('#navMain').tinyNav({
         header: 'Menü'
      });
   });
</script>
```

Listing 9.5 JavaScript-Einbindung für das Select-Menü TinyNav in unserem Praxisbeispiel

Das Select-Menü, das nun im Basislayout statt der Navigationsliste dargestellt wird, schieben wir nach rechts (per `float`). Die normale Navigation `#navMain` setzen wir für kleine Screens auf `display: none`. Für größere Viewports drehen wir die Ausgabe um. Jetzt wird das Select-Menü versteckt und die normale Navigation wieder sichtbar (siehe Abbildung 9.8).

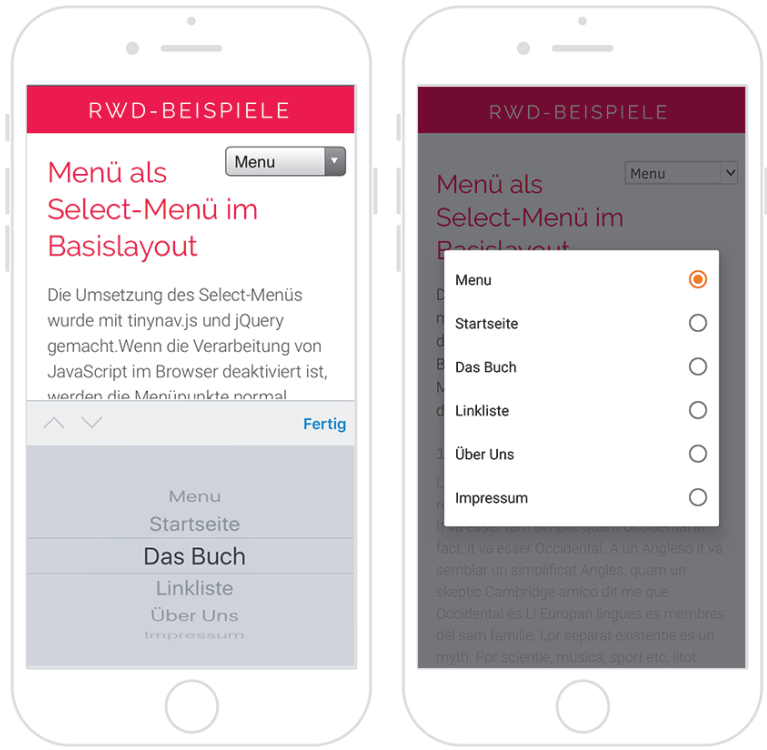

Abbildung 9.8 Praxisbeispiel: Das Menü wird im kleinsten Viewport in einer Auswahlbox (Select-Menü) angezeigt. Das Layout ist dabei abhängig vom Betriebssystem der Geräte (links iOS, rechts Android).

```
.tinynav {
    display: block;
    float: right;
}
#navMain {
    display: none;
}
@media screen and (min-width: 40em)  {
    .tinynav {
        display: none;
    }
```



```
    #navMain {
        display: block;
    }
}
```

Listing 9.6 Ein- und Ausblenden sowie Positionierung des Menüs

Das funktioniert so weit ganz gut, aber wenn Sie das JavaScript in Ihrem Browser deaktivieren, werden Sie feststellen, dass es in der Basisversion gar keine Navigation mehr gibt.

Ergänzung eines Fallbacks, falls JavaScript nicht zur Verfügung steht

Der Container `#navMain` darf also nur versteckt werden, wenn der Browser JavaScript verarbeiten kann. Um das zu prüfen, haben wir eine Klasse `.js-off` in das `<html>`-Tag gesetzt und tauschen sie per JavaScript gegen eine andere (nämlich `.js-on`) aus. Diese Klasse verwenden wir dann zum Aus- und Wiedereinblenden des Menüs.

```
<html class="js-off">
```

Listing 9.7 Ergänzung im <html>-Tag, um zu prüfen, ob JavaScript aktiv ist

Nun folgt das JavaScript für den Austausch der Klasse `.js-off` durch `.js-on` im `<html>`-Tag:

```
<script src="../js/jquery.js"></script>
<script src="../js/tinynav.js"></script>
<script>
   $(function () {
      // Wenn JS aktiv remove "js-off" und setze "js-on"
      $('html').addClass('js-on');
      $('html').removeClass('js-off');
      $('#navMain').tinyNav({
         header: 'Menü'
      });
   });
</script>
```

Listing 9.8 JavaScript-Erkennung und Einbindung für das Select-Menü TinyNav in unserem Praxisbeispiel

Nur wenn die Klasse `.js-on #navMain` vorhanden ist, wird das Menü in der Basisversion aus- und für größere Viewports wieder eingeblendet.

```
.tinynav {
    display: block;
    float: right;
}
.js-on #navMain {
    display: none;
}
@media screen and (min-width: 40em) {
    .tinynav {
        display: none;
    }
    .js-on #navMain {
        display: block;
    }
}
```

Listing 9.9 Ein- und Ausblenden sowie Positionierung des Menüs (mit Fallback)

Wenn Sie jetzt in Ihrem Browser JavaScript deaktivieren, wird das Menü als Liste untereinander dargestellt. Das verbraucht zwar ziemlich viel Platz, aber der Nutzer kann sich nun trotz allem durch die Seite bewegen.

JavaScript-Weiche über CSS-Klassen mit und ohne jQuery

Im Beispiel eben haben wir die jQuery-Funktionen `addClass()` und `removeClass()` für die JavaScript-Erkennung verwendet, da wir hier für das Plug-in *tinyNav.js* jQuery sowieso einsetzen.

Das geht natürlich auch schlanker mit reinem JavaScript. Um unterschiedliche Styles anzuwenden, je nachdem, ob JavaScript aktiv ist oder nicht, führen Sie die folgenden Schritte aus:

Fügen Sie dieses kleine JavaScript direkt nach dem `<title>` in den Seitenkopf ein:

```
<script>
   document.documentElement.className += "js";
</script>
```

Wenn JavaScript aktiv ist, wird dadurch eine Klasse "`.js`" für das Element `<html>` (also `<html class="js">`) gesetzt.

Mit dieser Klasse (`.js`) können Sie andere Selektoren kombinieren und dadurch unterschiedliche Eigenschaften zuweisen, je nachdem, ob JavaScript aktiv ist oder nicht.

```
<script>
   .etwas { /* wenn kein JS verfügbar */ }
```



```
   .js .etwas { /* wenn JS verfügbar */  }
</script>
```

Diese simple Art der JavaScript-Erkennung und Fallback-Erstellung ist auch für alle anderen Menübeispiele wichtig. (Wenn Sie Modernizr einsetzen, haben Sie die Erkennung schon mit an Bord.)

9.6 Navigation per Anker am Ende des Seiteninhalts

Ein längeres Menü kann für kleine Viewports an den unteren Seitenrand verschoben werden. Über einen Menülink im oberen Bereich springt man dann zu einem Anker am Ende der Seite, um dort zu navigieren. Von Vorteil ist, dass es sich hierbei um eine Lösung ohne zusätzliche Skripte handelt, die auf allen Systemen sicher funktioniert. Das Menü versperrt bei einem Seitenwechsel nicht die Sicht auf den eigentlichen Inhalt, da man wieder am Kopf der neuen Seite landet.

Für die Nutzer kann es erst einmal verwirrend sein, dass die Navigation am Ende der Seite steht. Der Menüankerlink von oben ist ganz wichtig. Hilfreich, um unnötiges Scrollen zu vermeiden, ist zusätzlich ein »Nach-oben«-Link aus dem Menübereich zurück. Momentum beispielsweise hat dieses Navigationskonzept gewählt, obwohl sie nur wenige Menüpunkte unterbringen mussten (siehe Abbildung 9.9).

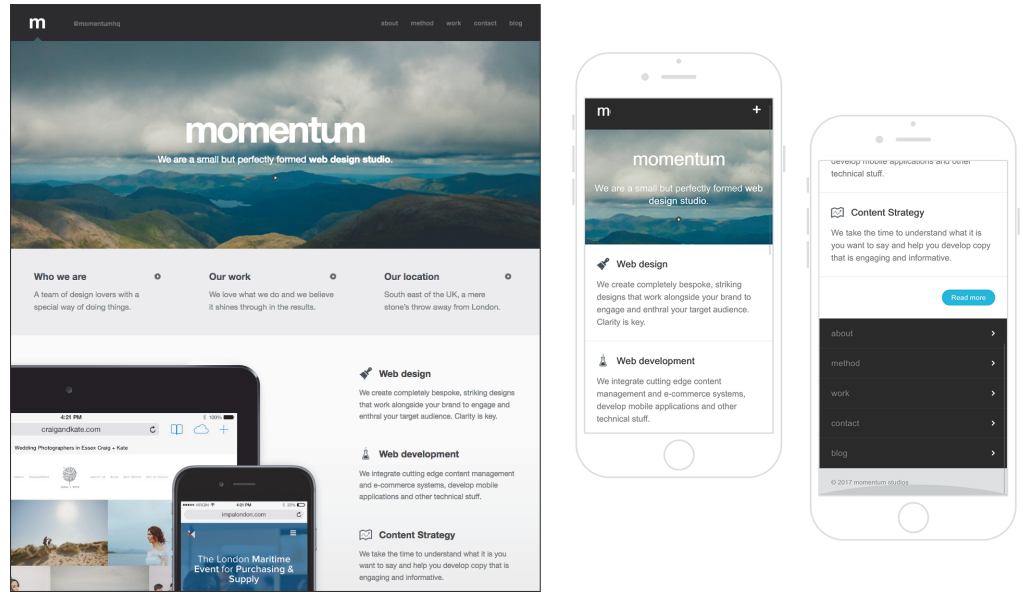

Abbildung 9.9 Bei Momentum rutscht die Navigation auf kleinen Screens ans Ende der Seite (http://builtwithmomentum.com).

9.6.1 Praxisbeispiel: Footer-Navigation mit Anker

Um das Prinzip der Footer-Navigation zu veranschaulichen, haben wir diese einfache Variante in unser Praxisbeispiel integriert. Sie finden sie im Download-Paket unter *praxisbeispiele/kap09/bsp_09-04/*. Damit dieses Beispiel funktioniert, müssen Sie unser bisher verwendetes HTML-Markup leicht verändern. Wir brauchen einen zusätzlichen Container (`div.content-wrapper`) um den inhaltlichen Bereich herum, unter dem das `<nav>`-Element in der Basisversion angezeigt werden soll:

```
<div class="content-wrapper">
   <header class="header" role="banner">
      <div class="logo">...</div>
      <a class="toggle-nav" href="#navMain">Menü</a>
   </header>
   <nav class="nav">
      <ul id="navMain" class="main-nav" role="navigation">
         <li><a href="#">...</a></li>
         <li><a href="#">...</a></li>
         <li id="back-to-top">
            <a href="#top" title="Zum Seitenanfang"> -- ⇧ -- </a>
         </li>
      </ul>
   </nav>
   <div class="main-wrapper">...</div>
</div> <!-- div.content-wrapper - Ende -->
<footer class="footer">...</footer>
```

Listing 9.10 Auszug aus der HTML-Datei mit dem zusätzlichen »div.content-wrapper« für die Positionierung der Navigation vor dem Footer

Dem Container `div.content-wrapper` geben Sie ein `display: table` und dem `.nav` ein `display: table-footer-group`, und schon schiebt sich die Navigation an das Ende des Containers. Über den Menülink aus dem Header heraus springt der Nutzer zu einem Anker in das Menü am Ende der Seite, über einen »Nach-oben«-Link kann er dann wieder an den Anfang der Seite navigieren. Der »Nach-oben«-Link im Menü `#back-to-top` wird in der Basisversion nach rechts eingerückt und für Viewports ab 40 em wieder ausgeblendet.

```
.content-wrapper {
   display: table;
}
.nav {
   display: table-footer-group;
}
```



```
#back-to-top {
   text-align: right;
}
@media screen and (min-width: 40em) {
   .content-wrapper {
      display: block;
   }
   .nav {
      display: block;
   }
   #back-to-top {
      display: none;
   }
}
```

Listing 9.11 CSS-Anpassungen für das Footer-Menü mit Anker

Das CSS für das Menü selbst wird unverändert beibehalten. Was jetzt noch fehlt, ist die Auszeichnung für den Menülink in der Kopfzeile (siehe Listing 9.12):

```
a.toggle-nav {
   position: absolute;
   top: 0.4rem;
   right: 0.5rem;
   display: inline-block;
   padding: 0.357em;
   font-weight: 300;
   border-radius: 0.2em;
   text-decoration: none;
   cursor: pointer;
}
a.toggle-nav:link,
a.toggle-nav:visited,
a.toggle-nav:active,
a.toggle-nav:focus {
   background: #189ca4;
   border: 1px solid #189ca4;
   color: #fff;
}
a.toggle-nav:hover {
   background: #96cfbf;
   border: 1px solid #96cfbf;
   color: #fff;
}
```

```
@media screen and (min-width: 40em) {
   a.toggle-nav {
      display: none;
   }
}
@media screen and (max-width: 23em) {
   .header {
      text-align: left;
   }
}
```

Listing 9.12 CSS-Auszeichnungen für den Menülink oben rechts (gilt für alle Beispiele)

Der Menülink wird per `position: absolute` rechts oben positioniert (siehe Abbildung 9.10). Für Bildschirme ab 40 em wird der Menülink nicht mehr angezeigt (das Tablet-Layout zeigt ja die Navigation am oberen Rand), und für besonders schmale Viewports bekommt der `.header` ein `text-align: left`, damit daneben mehr Platz für den Menülink bleibt.

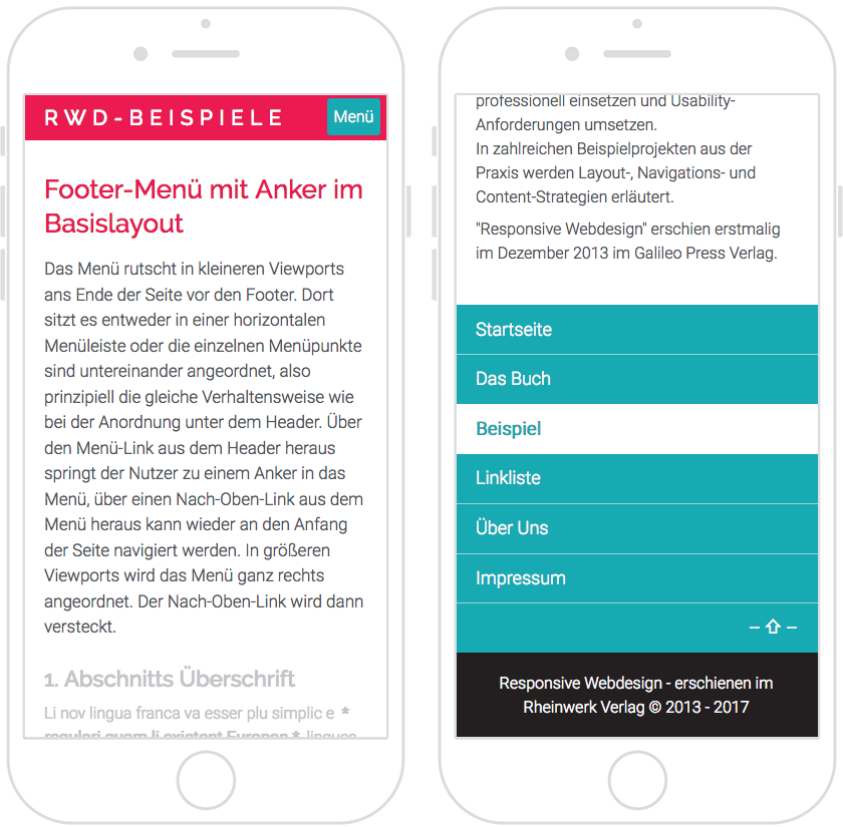

Abbildung 9.10 Nach Klick auf den Menübutton landet man im Menü am Ende der Seite.

Bei den bislang beschriebenen Varianten ist das Menü jederzeit irgendwo auf der Seite sichtbar und zugänglich. Ein anderer Ansatz ist, die Navigation aus dem sichtbaren Bereich zu entfernen und sie erst nach einer Aktion durch den Benutzer wieder einzublenden.

9.7 Toggle-Menü

Das Toggle-Menü (*to toggle* = »umschalten«) hat die größte Verbreitung bei der mobilen Navigation. Durch das Drücken auf einen Menülink wird das Menü sichtbar. Dabei kann es sich oberhalb der Website oder unter der Logoleiste aufschieben, oder es legt sich auf die Webseite, ohne sich einen Bereich freizuschieben. Nach Auswahl eines Menüpunktes wird die neue Seite geladen, und das Menü ist wieder ausgeblendet.

9.7.1 Praxisbeispiel: Toggle-Menü mit dem Plug-in Responsive-Nav

Für die Umsetzung des Toggle-Menüs in diesem Beispiel haben wir uns für ein kleines, leichtgewichtiges JavaScript-Plug-in namens Responsive Nav (*http://responsive-nav.com*) entschieden. Mit dem Plug-in des finnischen Entwicklers Viljami Salminen ist der Aufbau einer »togglebaren« Navigation ganz einfach. Das Plug-in *Responsive Nav* können Sie direkt von *https://github.com/viljamis/responsive-nav.js* oder von der oben genannten Projekt-Website herunterladen. Dort werden auch mehrere Beispiele für den Einsatz gezeigt.

Als Ausgangssituation für unser Praxisbeispiel können Sie wieder */kap9/01start-nav/* verwenden. Das Menü ist auf kleinen Viewports also erst einmal ausgeblendet; somit haben die Nutzer freie Sicht auf die Inhalte und können das Menü bei Bedarf anzeigen lassen. Die Position rechts oben für den Menübutton hat sich inzwischen bei mobilen Sites, die diese Art der Navigation nutzen, weitgehend eingebürgert. Ein beschrifteter Button sorgt für mehr Verständlichkeit, als ein Button, der nur durch ein Icon gekennzeichnet ist (siehe auch dazu Abschnitt 6.1.3, »Verständlichkeit«; siehe Abbildung 9.11).

Die Einbindung des Menülinks im HTML-Beispiel erfolgt hier durch das Plug-in selbst und entfällt deshalb im Markup.

```
<header class="header" role="banner">
   <div class="logo">...</div>
</header>
<nav id="nav" class="nav" role="navigation">
   <ul id="navMain" class="main-nav">
```

```
      <li><a href="#">Link 1</a></li>
      <li><a href="#">Link 2</a></li>
      <li>...</li>
      ...
   </ul>
</nav>
```

Listing 9.13 Auszug aus dem HTML-Code für das Responsive-Nav-Beispiel

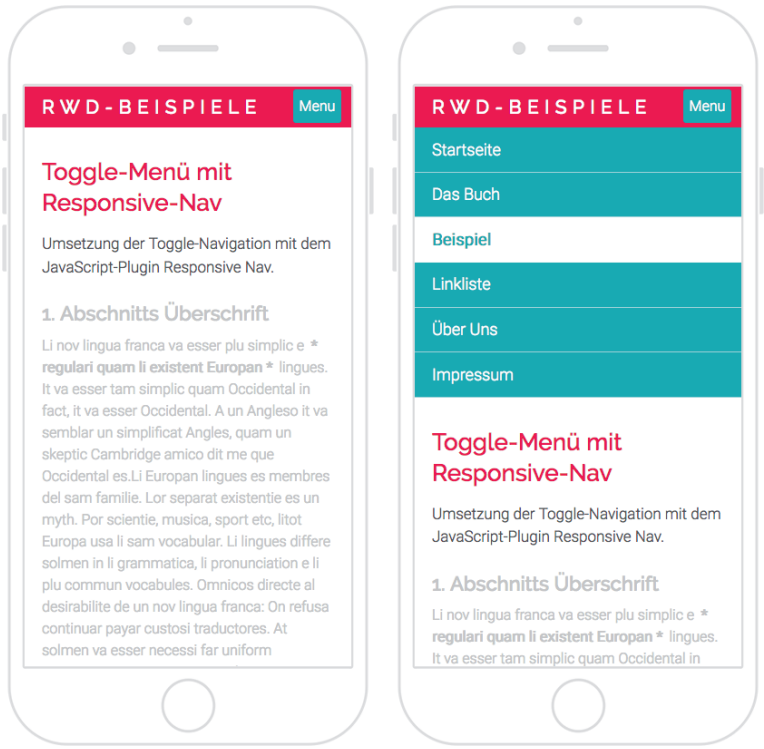

Abbildung 9.11 Praxisbeispiel: Toggle-Navigation mit Responsive Nav

Die JavaScript-Datei *responsive-nav.min.js* binden Sie vor dem schließenden `<body>`-Tag ein. Die JavaScript-Konfiguration haben wir so angepasst, dass der Anker zum Umschalten vor dem Menü eingebunden wird. Die Navigation soll darunter ausgefahren werden, wie auch in den Beispielen vorher.

```
<script src="js/responsive-nav.min.js"></script>
<script>
   var navigation = responsiveNav("#nav", {
      insert: "before"
```



```
   });
</script>
```

Listing 9.14 JavaScript-Einbindung und Konfiguration für Responsive Nav vor dem schließenden <body>-Tag

Wir verwenden weiterhin unsere eigenen Layoutstyles für die Navigation und haben zusätzlich die Styles für `.nav-collapse` und die Erkennung, ob JavaScript im Browser aktiv ist (oder nicht), aus dem *responsive-nav.css* übernommen. Hier wird jetzt die Klasse `.js` in das `<html>`-Element gesetzt. Wenn also JavaScript aktiv ist und `.js` existiert, wird die Navigation versteckt und »getoggelt«. Das heißt, dass unser Menü auch hier ohne JavaScript zugänglich ist: Es wird dann nicht versteckt.

```
.js .nav-collapse {
   clip: rect(0 0 0 0);
   max-height: 0;
   position: absolute;
   display: block;
   overflow: hidden;
   zoom: 1;
}
.nav-collapse.opened {
   max-height: 9999px;
}
```

Listing 9.15 CSS-Anweisungen, die wir aus der Datei responsive-nav.css übernehmen

Der hohe Wert von 9.999 Pixeln für `.nav-collapse.opened` dient als Fallback, falls die JavaScript-Berechnung der realen maximalen Höhe der geöffneten Navigation fehlschlägt. Für den Menü-Toggle-Button haben wir ebenfalls unsere eigenen Styles verwendet, die wir in diesem Beispiel auf `#nav-toggle` gelegt haben, da das Plug-in Responsive Nav diese mitbringt. Der Menülink ist wieder absolut in der oberen Leiste positioniert.

Für größere Viewports wird der Menübutton aus- und das Menü selbst wieder eingeblendet werden. Dazu haben wir aus der Datei *responsive-nav.css* des Plug-ins die passenden Styles übernommen, aber den Viewport an unsere Bedürfnisse angepasst. Der Breakpoint, an dem sich die Darstellung unseres Menüs verändert, liegt bei 640 Pixel (40 em). Daraus folgt:

```
@media screen and (min-width: 40em) {
   .js .nav-collapse {
      position: relative;
      overflow: visible;
   }
```

```
    .js .nav-collapse.closed {
        max-height: none;
    }
    /* Menülink verstecken in großen Viewports */
    a.nav-toggle {
        display: none;
    }
}
```

Listing 9.16 CSS zum Aufheben der vorher definierten Styles für größere Viewports

Den gesamten Quellcode des Praxisbeispiels finden Sie unter *praxisbeispiele/kap09/bsp_09-06/*.

Es gibt auch die Möglichkeit, ein Toggle-Menü ausschließlich mit CSS umzusetzen. Dafür wird die CSS-Pseudoklasse `:target` verwendet. Auch dazu haben wir Ihnen ein Beispiel im Download-Paket zur Verfügung gestellt: *praxisbeispiele/kap09/bsp_09-05/*.

9.8 Off-Canvas-Menü

Eine andere schöne Lösung ist das *Off-Canvas-Menü*. Dabei wird das Menü neben oder über dem Hauptinhalt angeordnet und dieser Bereich im Normalfall nicht angezeigt. Nach Klick auf einen Menülink schiebt sich dann das Menü von oben oder von der Seite in den Viewport und überlagert entweder den Inhalt, oder es schiebt für die Dauer der Menüwahl den Inhalt aus dem Viewport. Eine schematische Darstellung finden Sie in Abbildung 9.12. Die Off-Canvas-Technik selbst haben wir Ihnen als Layoutkonzept auch schon in Kapitel 7, »Responsive Layout-Patterns«, vorgestellt.

Auf Smartphones und Tablets ist das Layoutkonzept der Off-Canvas-Menüführung sehr weit verbreitet und in vielen Varianten vorhanden. Der Menüblock kann sich über dem Inhalt ausbreiten oder diesen beiseite schieben, er kann den Inhalt nur partiell verdecken oder den gesamten Viewport einnehmen. Oft wird für das Menü auf kleinen Screens eine 2/3 Bildschirmausfüllung gewählt; der daneben noch sichtbare Inhaltsbereich kann mit einem halbtransparenten Overlay abgedunkelt werden, wie beispielsweise beim Off-Canvas-Menü auf der Bodum-Website zu sehen ist (siehe Abbildung 9.13). Das Overlay dient einerseits dazu, den Inhalt in den Hintergrund zu rücken und das Menü hervorzuheben, und andererseits als Klickfläche, über die das Menü auch ohne Auswahl eines Menülinks wieder geschlossen werden kann. Das Schließen des Menüs sollte natürlich auch über den Menübutton möglich sein, aber wenn zusätzlich auch die freie Fläche um den sichtbaren Menüblock angeklickt werden kann, bedeutet das eine einfachere Bedienung für die Nutzer.

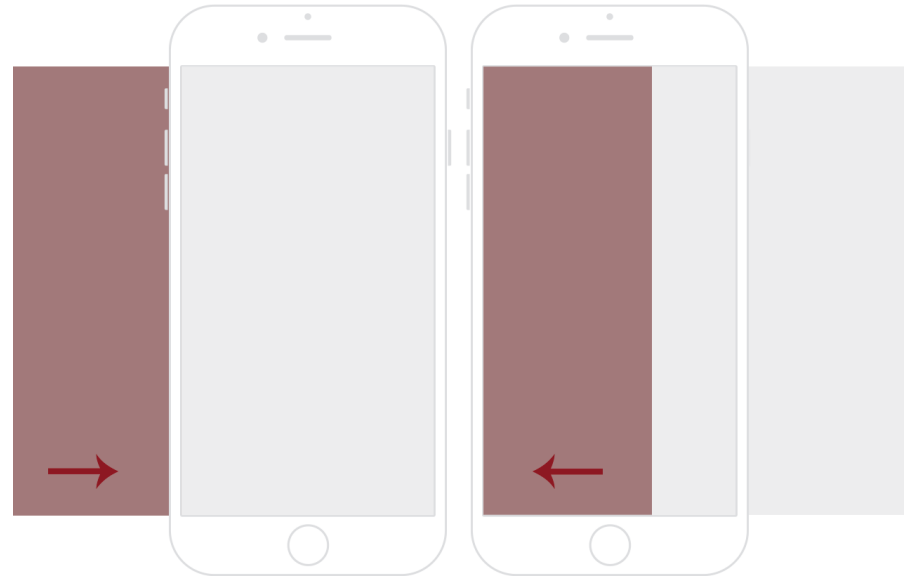

Abbildung 9.12 Schematische Darstellung eines Off-Canvas-Menüs auf kleinen mobilen Geräten: Links ist das Menü außerhalb des sichtbaren Bereichs, rechts schiebt es sich seitlich auf den Bildschirm.

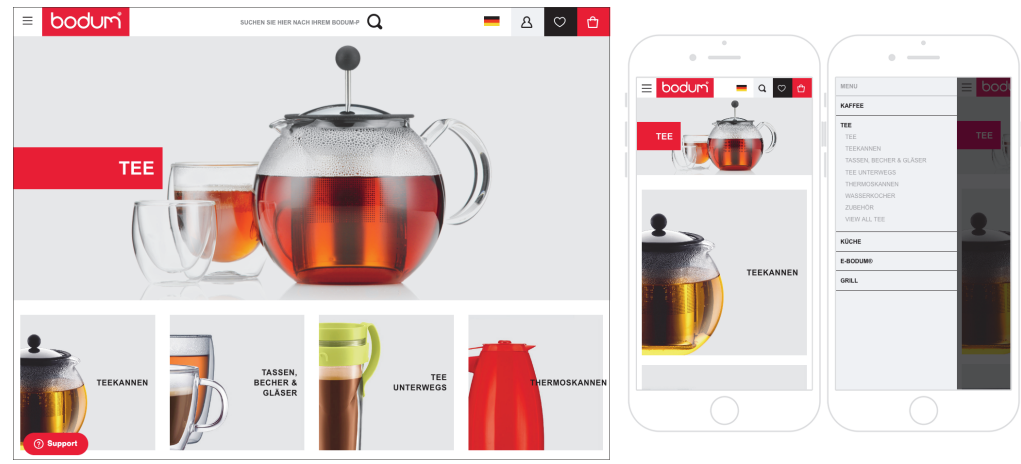

Abbildung 9.13 Auch Bodum arbeitet mit einem Off-Canvas-Menü für kleine Viewports (http://www.bodum.com).

9.8.1 Praxisbeispiel: Off-Canvas-Menü mit Pushy

Ein moderneres Off-Canvas-Menü ist das jQuery-Plug-in *Pushy* von Christopher Yee. Sie finden seine Demo-Site unter *www.christopheryee.ca/pushy* und können das Plug-in von Github herunterladen: *https://github.com/christophery/pushy*.

Die Implementierung in unser Layout ist ganz einfach. Alle Container im HTML-Code, die beiseite geschoben werden, wenn das Menü sichtbar wird, bekommen die extra Klasse `.push` (siehe Listing 9.17). Zusätzlich bekommt die Navigation die Klasse `.pushy` und ein `.pushy-left` oder `.pushy-right`, je nachdem, ob das Menü von rechts oder links auf den Screen geschoben werden soll. Der Menülink braucht die zusätzliche Klasse `.menu-btn` für die JavaScript-Funktionalität (unser Layout für Menü und Menülink kommt nach wie vor aus den Styles zu `.toggle-nav` und `.main-nav`). Die Klasse `.pushy-submenu` leitet ein Untermenü ein, und jedes Listenelement (<li>), das die Klasse `.pushy-link` bekommt, schließt das Menü auch wieder. (Was hier eher im Demo interessant ist; wenn reale Links in den LIs liegen, wird nach einem Klick ohnehin eine neue Seite geladen, und das Menü ist dann wieder geschlossen.) Das angepasste HTML unseres Beispiels sieht nun so aus:

```
<div class="page-wrapper">
   <header class="header push" role="banner">
      <div class="logo">...</div>
      <a class="toggle-nav menu-btn" href="#">Menü</a>
   </header>
   <nav class="pushy pushy-left nav" role="navigation">
      <ul class="main-nav">
         <li class="pushy-link"><a href="/">Startseite</a></li>
         <li class="pushy-submenu"> <a href="#">Submenu 1</a>
            <ul>
               <li class="pushy-link"><a href="#">Item 1</a></li>
               <li class="pushy-link">...</li>
            </ul>
         </li>
         <li class="pushy-link">...</li>
         <li class="pushy-link">...</li>
      </ul>
   </nav>
   <div class="site-overlay"></div>
   <div class="main-wrapper push">...</div>
   <footer class="footer push" role="contentinfo">...</footer>
</div>
<script src="js/jquery.js"></script>
<script src="js/pushy.js"></script>
```

Listing 9.17 Wir brauchen ein paar HTML-Anpassungen, damit das pushy.js greift.

Wir binden die JavaScript-Datei *pushy.js* hinter der jQuery-Bibliothek am Ende der HTML-Datei (vor dem schließenden <body>-Tag) ein. Das *pushy.css* des Plug-ins übernehmen wir mit einigen kleinen Anpassungen. Wir begrenzen sein Wirken mit

einem Media Query mit `max-width` auf 1.023 Pixel (63.938 em), damit das Menü für die Desktopversion wieder aussieht wie in all unseren Praxisbeispielen. Das ist natürlich nicht zwingend. Viele Websites verwenden das Off-Canvas-Menü in allen Layoutstufen. Bis auf die Positionierung und die Hintergrundfarbe mussten wir keine Anpassungen im CSS vornehmen, damit sich das Layout in unseres integriert. In Abbildung 9.14 sehen Sie das Ergebnis. Das komplette Beispiel finden Sie im Download-Paket und auf der Website zum Buch unter *praxisbeispiele/kap09/bsp_09-07/*.

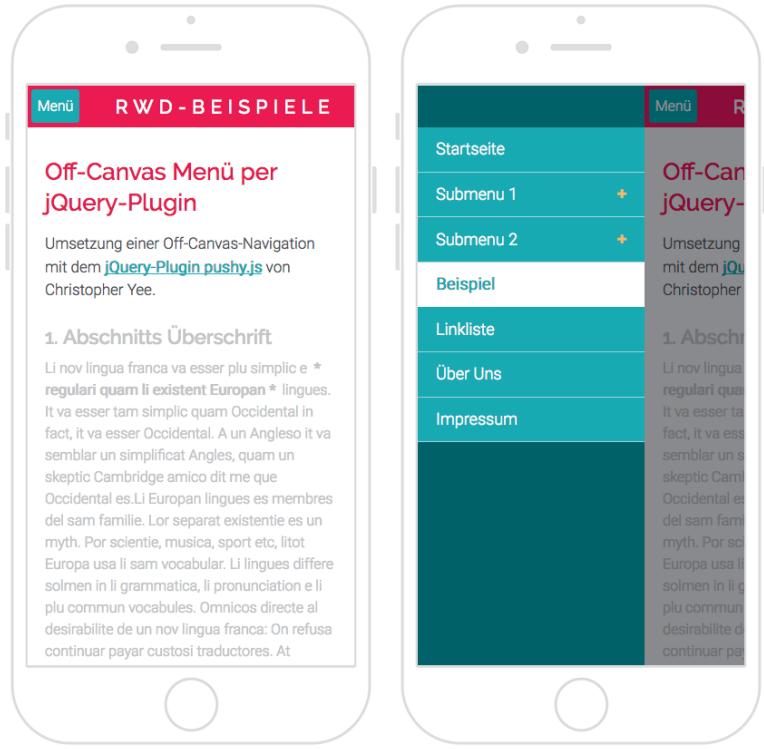

Abbildung 9.14 Beim Off-Canvas-Menü pushy.js ist sowohl die Unterstützung von Untermenüs als auch das Schließen des Menüs durch Klick auf das Overlay über den verschobenen Inhaltsbereich schon dabei.

9.8.2 Praxisbeispiel: Swipe-Menü mit Slideout.js

Eine weitere elegante Variante des Off-Canvas-Menüs ist das Swipe-Menü. Auch hier wird das Menüpanel seitlich auf den Screen geschoben. Das kann entweder wieder über einen Klick auf den Menübutton ausgelöst werden oder durch eine Geste auf dem Touchscreen, das »Swipen« – eine seitliche Wischbewegung mit einem Finger (siehe Abbildung 9.15). Viele native Apps bedienen sich dieses Verhaltens. In der Benutzerfreundlichkeit ist es unschlagbar, da das Swipen praktisch überall auf dem Touchscreen ausgeführt werden kann.

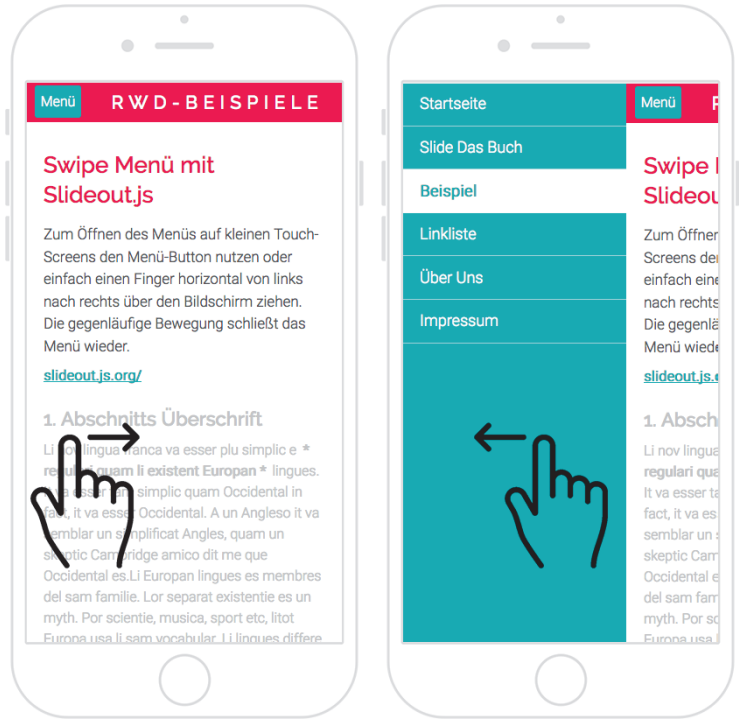

Abbildung 9.15 Praxisbeispiel mit Swipe-Menü. Durch einfache Wischgesten kann das Menü auf den und vom Screen geschoben werden.

Für die Umsetzung haben wir *Slideout.js* von Mango (*https://github.com/Mango/slideout*) in das Praxisbeispiel integriert. Für dieses Beispiel müssen wir das Markup (HTML) des Praxisbeispiels wieder etwas verändern. Die Navigation verschieben wir innerhalb des `div.page-wrapper` vor den `<header>` und geben ihm die `id="menu"`. Außerdem brauchen wir noch einen zusätzlichen Container, der `<header>`, `div.main-wrapper` und `<footer>` umschließt. Dieser neue Wrapper bekommt die `id="panel"`. Letztendlich können Sie die IDs benennen, wie Sie möchten, wichtig ist nur, dass bei der Initialisierung in JavaScript (Listing 9.18) die richtigen Bezeichner gesetzt werden.

```
<script src='js/slideout.min.js'></script>
<script>
   var slideout = new Slideout({
      'panel': document.getElementById('panel'),
      'menu': document.getElementById('menu'),
      'padding': 256
   });
   document.querySelector('.toggle-button') ➲
      .addEventListener('click', function() {
         slideout.toggle();
   });
</script>
```

Listing 9.18 JavaScript-Einbindung und Initialisierung für das Slideout.js wie vor dem schließenden <body>-Tag

Wir belassen die Default-Positionierung des Plug-ins für das Menü auf der linken Seite. Mit den Parametern `'side': 'right'` könnten Sie das bei der Initialisierung auch auf die rechte Seite setzen. Das `'padding': 256` ist der Wert, um den der Inhalt des `div#panel` verschoben wird. Das `.slideout-menu` hat als Weite im CSS-Beispiel (siehe Listing 9.19) den gleichen Wert bekommen. Wir nehmen einige Anpassungen für unser Layout an den Standardstyles des Plug-ins vor.

```
.slideout-menu {
   position: fixed;
   top: 0;
   width: 256px;
   -webkit-overflow-scrolling: touch;
   z-index: 0;
   display: none;
}
.slideout-menu-left {
   left: 0;
}
.slideout-menu-right {
   right: 0;
}
.slideout-panel {
   position: relative;
   z-index: 1;
   will-change: transform;
   background-color: #fff;
}
.slideout-open,
.slideout-open body,
.slideout-open .slideout-panel {
   overflow: hidden;
}
.slideout-open .slideout-menu {
   display: block;
}
```

Listing 9.19 Für unser Layout angepasste Styles aus der Default-CSS-Datei für slideout.js

Für die Positionierung der Navigation bekommt der `.page-wrapper` ein `position: relative` und das `UL.main-nav` eine Mindesthöhe, sodass sie die gesamte Höhe des `.page-wrapper` einnimmt.

```
.page-wrapper {
   position: relative;
}
.main-nav {
   overflow: auto;
   min-height: 100%;
}
```

In kleinsten Viewports müssen wir noch etwas Platz schaffen für den Toggle-Button, der jetzt links neben dem Logo im Header sitzt:

```
@media screen and (max-width: 23em) {
   .header {
      text-align: right;
   }
   .logo a {
      padding-left: 5rem;
   }
}
```

Ab der Tablet-Layoutstufe gibt es wieder ein paar Layoutanpassungen. Der Menübutton bekommt mehr Raum im jetzt größeren Header und wird neu positioniert, und der erste Navigationspunkt bekommt mehr Abstand im seitlichen Menü nach oben:

```
@media screen and (min-width: 40em) {
   .slideout-menu {
      padding-top: 7.9rem;
   }
   a.toggle-nav {
      top: 2.1rem;
      left: 2rem;
      right: auto;
   }
   .logo {
      margin-left: 6rem;
   }
   .logo a {
      padding-left: 0;
   }
}
```


Wenn wir ab der Desktop-Layoutstufe wieder unser übliches Layout des Praxisbeispiels verwenden und die Navigation an den linken Rand rücken wollen, setzen wir die dafür notwendigen Änderungen in einem weiteren Media Query für große Bildschirme. Wir verstecken den Toggle-Button und setzen die Positionierungen zurück:

```
@media screen and (min-width: 64em) {
   .slideout-menu {
      position: absolute;
      margin-top: 3rem;
      width: 20%;
      min-height: 0;
      z-index: 2;
      display: block;
   }
   .slideout-panel {
      min-height: 0;
      background-color: #189ca4;
   }
   a.toggle-nav {
      display: none;
   }
   .main-wrapper {
      width: 80%;
      margin-left: 20%;
   }
   .logo {
      margin-left: 0;
   }
   .main-nav > li {
      border: none;
   }
   .main-nav > li > a,
   .main-nav > li > b {
      padding: 0.5em 3rem;
   }
}
```

Listing 9.20 Zusätzliche Styles für slideout.js, um das Plug-in zu integrieren

In der Praxis findet man das Swipe-Menü auch schon als alleinige Lösung für alle Layoutstufen. Damit verschwindet das Menü aber auch von der Desktopversion. Benutzerfreundlicher ist es, das Menü auf großen Screens wieder standardmäßig einzublenden, wie wir es in unserem Praxisbeispiel machen. Sie finden es im Download-Paket auf der Website zum Buch unter *praxisbeispiele/kap09/bsp_09-08/*.

9.9 Multilevel-Menüs

Menüs mit mehreren Ebenen stellen eine besondere Herausforderung im responsiven Layout dar. Auf Desktoprechnern wird von verschachtelten Menüs oft nur die erste Ebene dargestellt, entweder in einer vertikalen oder in einer horizontalen Anordnung. Beim Überfahren eines Menüpunktes mit der Maus oder beim Klick auf einen Menüpunkt öffnet sich dann das Untermenü. Auf dem Desktop ist technisch sowohl das eine als auch das andere möglich. Bei einem Menü in der Seitenleiste, bei dem alle Menüpunkte untereinander dargestellt werden, ist es allerdings nicht sinnvoll, bei jedem Darüberfahren mit der Maus das Öffnen und Schließen auszulösen, weil sich ja jedes Mal der gesamte Rest des Menüs nach oben oder unten verschiebt. Ein Aus- und Einklappen weiterer Ebenen beim Hovern wirkt hier sehr unruhig, es sei denn, Sie lassen die nächste Ebene seitlich erscheinen (siehe Abbildung 9.16).

Abbildung 9.16 Multilevel-Menü mit aufgeklapptem Untermenü auf Desktop, Tablet(-Größe) und Smartphone

Bei einer Verwendung des Menüs in einer horizontalen Leiste hingegen funktioniert sowohl das Ausklappen beim Hovern als auch das Ausklappen nach Anklicken. Aber was machen Sie mit dem Menü in der gleichen Ansicht auf einem Tablet mit Touchbedienung – dort gibt es ja kein Hover? Auch wenn der Platz ausreichend groß sein kann, um die gleiche Darstellung zu verwenden wie auf Desktoprechnern: Wie kommen Sie ohne den Hovereffekt an die nächsttiefere Menüebene heran, um von dort aus weiter zu navigieren?

9.9.1 Die native Einbindung von Multilevel-Menüs auf iOS und Android

Das native Verhalten von Geräten mit Touchdisplay bei der Interpretation von Hovermenüs mit mehreren Ebenen ist noch immer recht unterschiedlich bei Android

und iOS. iOS-Systeme wandeln einen Hover selbstständig in eine »Touchkaskade« um. Das heißt, wenn Sie ein Menü mit Hovereffekt verwenden, löst der erste Touch den eigentlichen Hovereffekt aus und erst der zweite den Link. Unter Android gibt es dieses Verhalten nicht. Hier führt der erste Touch sofort auf die verlinkte Seite; das Hovermenü wird ignoriert. Wenn Sie das selbst auf Ihrem Android-Gerät nachvollziehen wollen, können Sie das über das Praxisbeispiel *praxisbeispiele/kap09/bsp_09-09/* tun. (In Abschnitt 12.3.2 »Das Home-Device-Lab«, beschreiben wir Werkzeuge, um das Testen auf Mobilgeräten zu erleichtern, unter anderem die Software Browsersync.)

> **Hinweis zu den Beispielen**
>
> Zum Testen der Navigation sind in allen folgenden Beispielen zu Multilevel-Menüs unterhalb der Navigationspunkte BEISPIELE und LINKLISTE mehrere »Navitest«-Seiten verlinkt. Über den Navigationspunkt STARTSEITE kommen Sie immer wieder zurück zur Startseite des aktuellen Beispiels.

9.9.2 Praxisbeispiel: Multilevel-Menü mit DoubleTabToGo.js

Das Problem der unter Umständen fehlenden Hoverunterstützung lässt sich in den Griff bekommen, wenn die Navigationspunkte der ersten Ebene keine eigenen Seiten besitzen und damit keine Verlinkung haben. Wenn die Website-Struktur dies jedoch nicht zulässt, hilft ein kleines JavaScript (*doubleTabToGo.js*), um auch in Android-Geräten die Touchkaskade nachzurüsten.

Im folgenden Beispiel haben wir das Skript *doubleTabToGo.js* des litauischen Webentwicklers Osvaldas Valutis eingebunden (*http://osvaldas.info/drop-down-navigation-responsive-and-touch-friendly*).

Das Menü ist als ungeordnete Liste umgesetzt, die zweite Menüebene sitzt in einer verschachtelten Liste:

```
<nav class="nav" role="navigation">
  <ul id="navMain" class="main-nav first-level">
    <li><a href="multilevel1.html">Beispiele</a>
        <ul class="sec-level">
          <li><a href="navitest1.html">Navitest-01</a></li>
          <li><a href="navitest2.html">Navitest-02</a></li>
          ...
        </ul>
    </li>
    <li><a href=" multilevel2">Linkliste</a>
       <ul class="sec-level">
         <li><a href=" navitest6.html"> Navitest-06</a></li>
```

```
        ...
      </ul>
    </li>
    ...
  </ul>
</nav>
```

Listing 9.21 HTML-Auszug: Multilevel-Navigation (alle Beispiele)

Beim Überfahren mit der Maus wird das Untermenü angezeigt. Das erreichen Sie mit der CSS-Anweisung `li:hover ul {display:block;}`, die das zuvor auf `display: none` gesetzte Menü der zweiten Ebene einblendet. In den unterschiedlichen Viewports wird das Menü genauso angeordnet wie in unseren Beispielen vorher, nur dass die zweite Ebene noch ein paar zusätzliche Styles bekommt.

Am Ende vor dem schließenden `<body>`-Tag werden nun noch jQuery, das kleine jQuery-Plug-in und der Funktionsaufruf eingefügt:

```
<script src="js/jquery.js"></script>
<script src="js/doubletaptogo.js"></script>
<script>
   $(document).ready(function() {
      $('#navMain li:has(ul)').doubleTapToGo();
   });
</script>
```

Listing 9.22 DoubleTabToGo verhindert das sofortige Auslösen des zweiten Taps.

Beim ersten Tap auf einen übergeordneten Menülink verhindert das Skript, dass der Browser die dahinterliegende URL öffnet – bei einem zweiten Tap wird dann die Weiterleitung erlaubt und ausgeführt. Auf diese Weise wird das eingangs genannte Problem gelöst, dass bei Android die Weiterleitung immer sofort erfolgt. Natürlich ergibt dieses Verhalten nur bei den Menüpunkten mit Untermenüs Sinn. Die Zuordnung der Funktion erfolgt deshalb nur an `<li>`s, die wiederum eine `<ul>` beinhalten (`$('#navMain li:has(ul)').doubleTapToGo();`).

Den vollständigen Code entnehmen Sie bitte dem Anwendungsbeispiel (*praxisbeispiele/kap09/bsp_09-10*) im Download-Paket. Das Skript von Osvaldas Valutis finden Sie unter *https://osvaldas.info/examples/drop-down-navigation-touch-friendly-and-responsive/doubletaptogo.min.js*.

9.9.3 Praxisbeispiel: Multilevel-Menü mit Flexnav

Ein anderer Ansatz, sowohl den Menülink als auch das Toggeln des Untermenüs zur Verfügung zu stellen, ist es, die Funktion (zum Ausklappen der nächsten Menüebene)

in einen separaten Button auszulagern. In diesem Fall kann ein Tap auf den Menütitel die damit verlinkte Seite aufrufen und ein Tap auf einen Button daneben das Ausfahren der nächsten Ebene ansteuern. Auf dem Desktop kann man den üblichen Hovereffekt über das ganze Menüelement nutzen oder aus Gründen der Konsistenz ebenfalls einen separaten Button verwenden.

Jason Weaver hat ein jQuery-Plug-in namens FlexNav geschrieben, mit dem Sie ein flexibles Multilevel-Menü erstellen können (*http://jasonweaver.name/lab/flexiblenavigation* – siehe Abbildung 9.17). FlexNav macht aus einer einfachen Liste ein Menü, das auf dem Desktop per Hover und auf Tablets per Touchklick funktioniert und bei sehr kleinen Bildschirmen linearisiert wird.

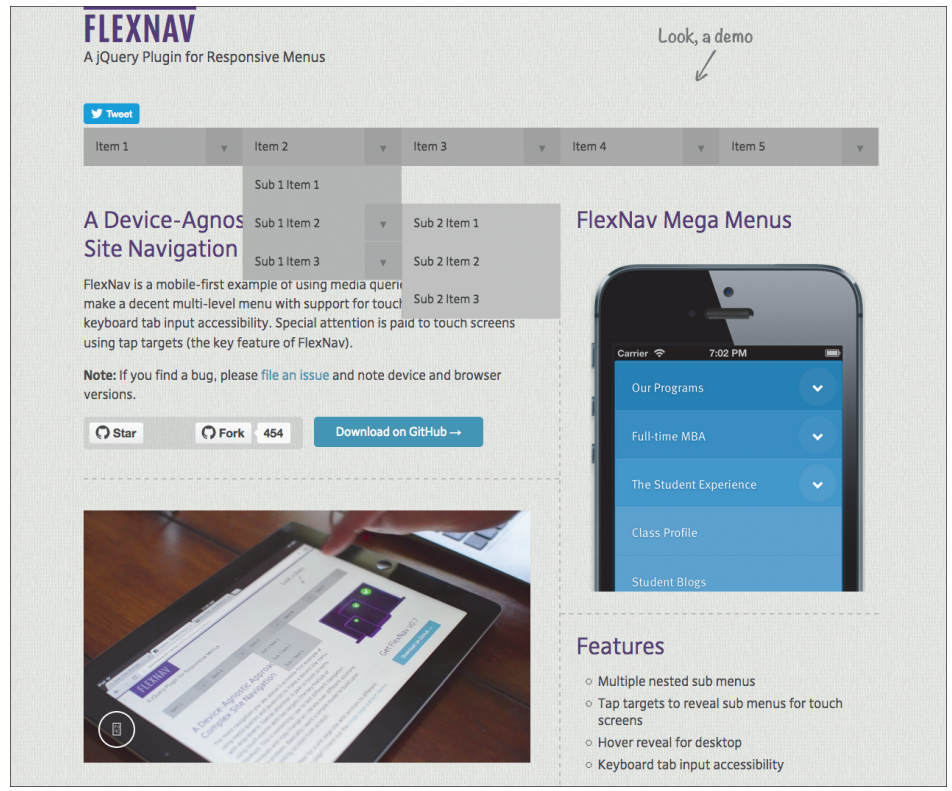

Abbildung 9.17 Die Projektseite von FlexNav dient gleichzeitig auch als Demo für das Plug-in.

Um das Plug-in zu nutzen, müssen Sie lediglich das FlexNav-JavaScript (und natürlich jQuery) am Seitenende einbinden, initiieren und dem Menü die Klasse `.flexnav` und einen Breakpoint mitgeben.

```
<nav class="nav" role="navigation">
  <ul data-breakpoint="640" class="flexnav main-nav first-level">
```

```
      <li><a href="multilevel1.html">Beispiele</a>
          <ul class="sec-level">
            <li><a href="navitest1.html">Navitest-01</a></li>
            <li><a href="navitest2.html">Navitest-02</a></li>
            ...
          </ul>
      </li>
      ...
    </ul>
</nav>
...
<script src="js/jquery.js"></script>
<script src="js/jquery.flexnav.min.js"></script>
<script>
   jQuery(document).ready(function($) {
      $(".flexnav").flexNav();
   });
</script>
```

Listing 9.23 Die Klasse ».flexnav« und ein »data-breakpoint« für die Navigationsliste und Skripteinbindung für das FlexNav-Plug-in

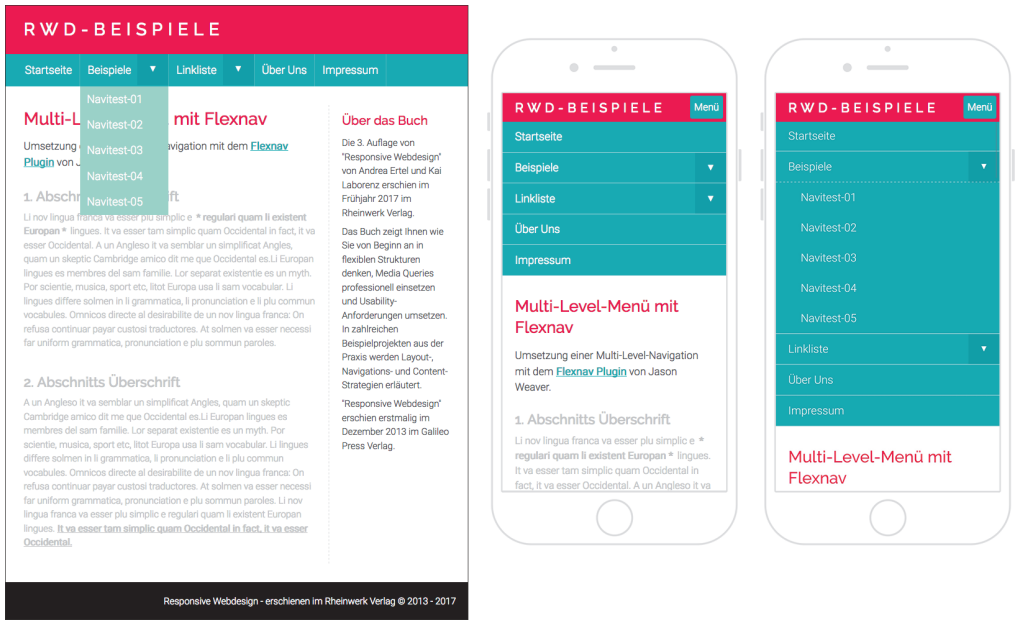

Abbildung 9.18 Das Multilevel-Dropdown-Menü mit FlexNav funktioniert auch auf Smartphones einwandfrei.

Das FlexNav-Plug-in bringt eigene Styles für das Menü mit. Wir haben diese für unsere Beispiel-Website angepasst (siehe Abbildung 9.18). Den vollständigen HTML- und CSS-Code entnehmen Sie bitte dem Anwendungsbeispiel im Download-Paket auf der Website zum Buch unter *praxisbeispiele/kap09/bsp_09-11/*, das Plug-in von Jason Weaver finden Sie unter *http://jasonweaver.name/lab/flexiblenavigation* bzw. *https://github.com/indyplanets/flexnav*.

9.9.4 Noch mehr Multilevel-Menüs ...

Ein weiteres jQuery-Plug-in für Multilevel-Navigationen ist *SlickNav* (*http://slicknav.com*). Es unterstützt die Tastaturbedienung, beinhaltet ARIA-Auszeichnungen und hat ein eingebautes Fallback, falls die JavaScript-Unterstützung im Browser nicht gegeben ist.

Weitere interessante Beispiele für Multilevel-Menüs finden Sie z. B. bei *Codrops*: (*https://tympanus.net/codrops/2015/11/17/multi-level-menu*) oder (*https://tympanus.net/codrops/2013/08/13/multi-level-push-menu*).

Auch Brad Frost hat sich mit dem Thema sehr umfassend beschäftigt und bietet eine Reihe von Ressourcen an: Als Startpunkt eignet sich sein Blogpost »Complex Navigation Patterns for Responsive Design« (*http://bradfrost.com/blog/web/complex-navigation-patterns-for-responsive-design*). Außerdem hat er eine Übersichtslinkliste zu verschiedenen Themen rund um responsives Design mit einer Sektion zu Navigations-Patterns zusammengestellt (*http://bradfrost.github.io/this-is-responsive/patterns.html#navigation*).

9.10 Zusammenfassung

In diesem Kapitel haben wir Ihnen verschiedene Lösungen für die Navigation Ihrer Website vorgestellt. Wir haben über die Grundlagen einer guten Navigation gesprochen und verschiedene Navigationsmuster (Patterns) diskutiert – von der Mininavigation und der Priority-Navigation am oberen Seitenrand über Toggle-Menüs, die sich ein- und ausblenden lassen, bis hin zu verschiedenen Off-Canvas-Lösungen. Aber auch für Menüs mit mehreren Menüebenen haben wir Ihnen Lösungen vorgestellt. Je mehr Raffinesse (oder Verwendung neuester Techniken) so ein »Supermenü« mit sich bringt, desto wichtiger ist es, das Endergebnis auf unterschiedlichen Geräten und Betriebssystemen umfassend zu testen.

Sehen wir uns in den nächsten Kapiteln an, wie wir die Website mit anpassungsfähigen Inhalten füllen können.

Auf einen Blick

Inhalt

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

Andrea Ertel ist Webentwicklerin und Senior Frontend Developerin beim E-Mail-Anbieter Posteo. Für verschiedene Agenturen hat sie zudem barrierearme, anpassungsfähige Websites erstellt und diese in das Content-Management-System TYPO3 integriert.

Kai Laborenz ist im Web seit 1994 zu Hause und hat als Freelancer und in verschiedenen Agenturen zahlreiche Online-Projekte erfolgreich umgesetzt. Er saß als Juror in der Fachjury des Webdesign-Wettbewerbs »BIENE« (Barrierefreies Internet eröffnet neue Einsichten), ausgerichtet von der Aktion Mensch. Für eines seiner barrierefreien Webprojekte wurde er auch selber mit dem Preis ausgezeichnet.

Andrea Ertel, Kai Laborenz

Responsive Webdesign –
Konzepte, Techniken, Praxisbeispiele

524 Seiten, gebunden, 3. Auflage, Mai 2017
39,90 Euro, ISBN 978-3-8362-4578-4

www.rheinwerk-verlag.de/4395