3 Responsive Web Design

Unter ***Responsive Web Design*** versteht man die Positionierung und das Layout von HTML-Elementen in Abhängigkeit von den Abmessungen des Bildschirms bzw. des Fensters. Das ist bei Web-Apps besonders wichtig, da die mobilen Geräte sehr unterschiedliche Bildschirmgrößen aufweisen und ein „One-Size-Fits-All“-Ansatz sehr schnell zu unzufriedenen Anwendern führt.

Die Anpassungen müssen zudem dynamisch stattfinden, weil z.B. das Drehen eines Tablets vom Landscape- in den Portrait-Modus und umgekehrt jederzeit passieren kann und sofort zu einer veränderten Oberfläche führen soll.

3.1 Öffnen der Arbeitsmappe

Sie kennen es schon: Ihre Arbeitsmappe muss geöffnet werden. Dann muss der Stand vom letzten Mal, also der ***app***-Ordner unterhalb von ***exercise02,*** nach ***exercise03*** kopiert werden. Vergessen Sie nicht, die Datei ***index.html*** in ***exercise03*** zur neuen Startseite zu machen.

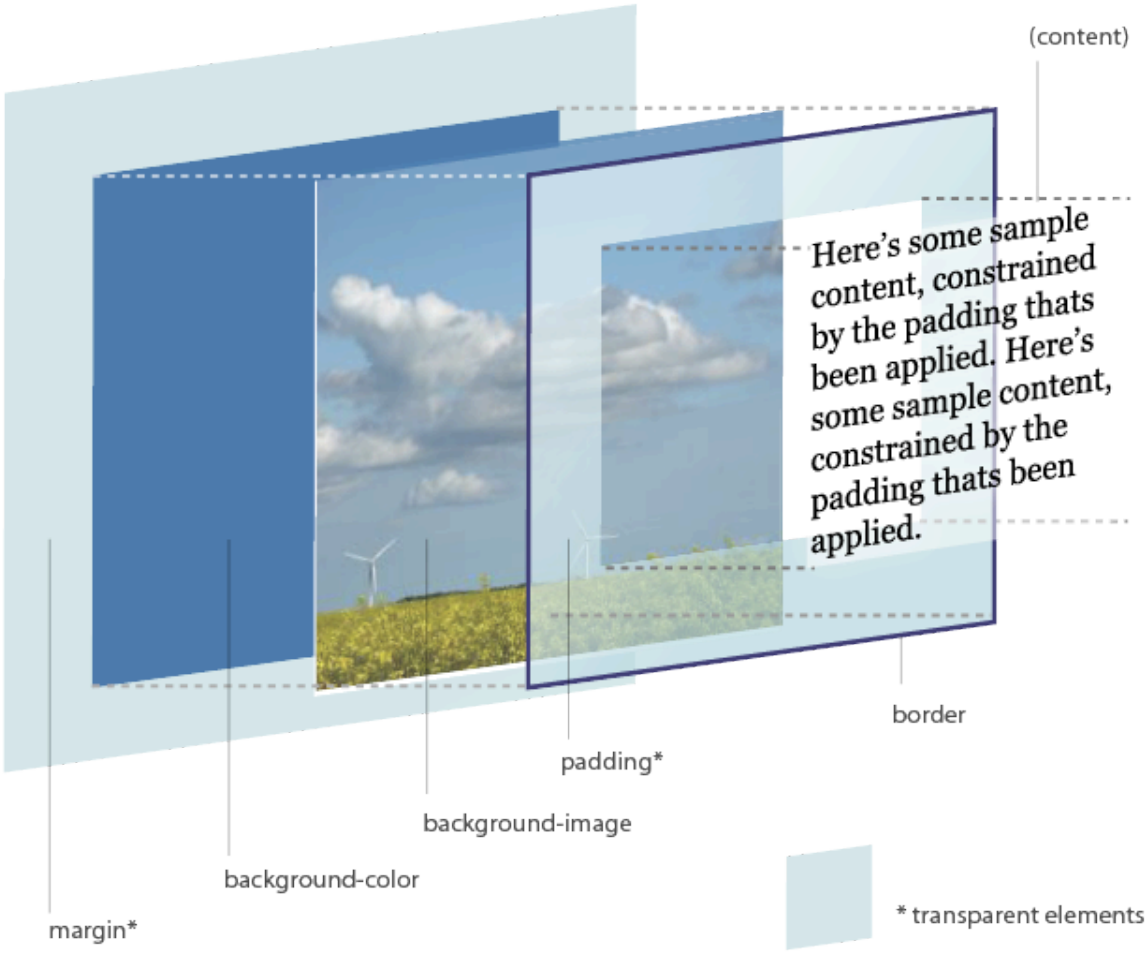
Sollten Sie unsicher sein, wie das genau geht, sehen Sie bitte in der Beschreibung des letzten Praktikums nach.

3.2 Boxmodell

Wir sind uns vermutlich schnell einig, dass die Positionierung wenig mit der Struktur der Information zu tun hat, sondern vorrangig eine Darstellungsfrage ist. Damit ist der ursprünglich beliebte Ansatz, Positionierungen mithilfe von HTML-Tabellen zu erzwingen nicht unser Ziel. Vielmehr werden wir CSS benutzen, um die Positionierung von HTML-Elementen zu beeinflussen.

Der Browser greift für die Positionierung jedes Elements auf das ***Boxmodell*** zurück:

- Bei jeder Box ist der eigentliche Inhalt zunächst von einer Art Passepartout (engl. ***padding***) umgeben, das für Abstand zu einem ggf. unsichtbaren Rahmen (engl. ***border***) sorgt. Dieser Rahmen wiederum wird durch einen Rand (engl. ***margin***) von umgebenden anderen Elementen getrennt.
- Darüber hinaus kann für den Bereich innerhalb des Rahmens eine Hintergrundfarbe (***background-color***) und/oder ein Hintergrundbild (***background-image***) festgelegt werden.
- Jedes Element hat zunächst einmal Standard-Einstellungen für alle diese Attribute, die jedoch mit CSS-Regeln überschrieben werden können.



Vielleicht hat Ihnen bei der Mondlandesimulation das letzte Mal auch nicht gefallen, dass die beiden Buttons unmittelbar nebeneinander angezeigt wurden. Viel schöner wäre es doch, wenn ein gewisser Abstand zwischen den Buttons eingehalten würde. Außerdem sollten die Buttons nicht zu klein sein, damit man sie auf Mobilgeräten auch mit dem Finger bedienen kann.

Wir definieren dazu in unserer Datei ***mobile.css*** eine CSS-Regel, die wir allerdings nicht direkt an die beiden Buttons binden, sondern über einen Class-Selektor zugänglich machen:

```
.btn-style {
    min-width: 150px;
    margin-right: 10px;
    margin-top: 20px;
}
```

Damit die Buttons auch von dieser Regel erfasst werden, muss nun noch die entsprechende Klasse im HTML hinterlegt werden. Da wir die Buttons ja in der JavaScript-Datei landung.js generieren, ist dort auch die Änderung vorzunehmen:

```
$("body").append("<button id='energy' class='btn-style'>Triebwerk an</button>");
$("body").append("<button id='no-energy' class='btn-style'>Triebwerk aus</button>");
```

Im Gegensatz zum Attribut ***id*** muss das Attribut ***class*** innerhalb des HTML-Dokuments nicht eindeutig sein, d.h. es können mit einem Class-Selektor mehrere Elemente angesprochen werden. Außerdem können im ***class***-Attribut mehrere Klassennamen aufgeführt werden, so dass für ein solches Element auch mehrere Class-Selektoren relevant sein können.

Testen Sie Ihre App.

3.3 Elementfluss

Normalerweise werden die Elementboxen vom Browser einfach hintereinander bzw. untereinander gesetzt. Je nach Elementtyp wird entschieden, ob dabei eine neue Zeile begonnen werden muss (sog. ***block***-Elemente wie z.B. ***div***, ***p***, ***h1***, ...) oder ob die Box einfach in der Zeile angehängt werden kann (sog. ***inline***-Elemente wie z.B. ***button***, ***small***, ***span***, ...).

Mithilfe der CSS-Eigenschaft ***display*** kann das Standardverhalten eines Elementtyps verändert werden.

	Bedeutung
display: inline	Das Element wird an die laufende Zeile angehängt, auch wenn es sich eigentlich um ein ***block***-Element handelt.
display: block	Für das Element wird eine neue Zeile begonnen, auch wenn es sich eigentlich um ein ***inline***-Element handelt.
display: none	Das Element wird nicht angezeigt.
...	

Möchte man ein Element völlig aus dem Elementfluss herausnehmen, kommt eine weitere CSS-Eigenschaft ins Spiel: ***position***. Normalerweise hat diese Eigenschaft bei allen Elementen den Wert ***static***, was dafür sorgt, dass die Elemente im Elementfluss gesetzt werden:

	Bedeutung
position: static	Normaler Elementfluss
position: absolute	Das Element wird aus dem Elementfluss entfernt und an absoluten Koordinaten dargestellt, die separat angegeben werden.
position: relative	Für das Element wird eine Lücke im Elementfluss freigehalten und es wird relativ verschoben zu dieser Lücke dargestellt.
...	

In unserer App möchten wir dauerhaft eine Menüzeile am oberen Fensterrand darstellen. Dazu werden wir ein ***div***-Element passend positionieren. Zunächst sorgen wir aber in ***mobile.css*** dafür, dass der bisherige Seiteninhalt etwas tiefer beginnt, damit er nicht von unserer Menüzeile verdeckt wird:

```
body {
    padding-top: 70px;
}
```

Anschließend fügen wir das ***div***-Element in unsere Datei ***index.html*** ein. Da das Element später absolut positioniert werden soll, ist es im Prinzip egal, an welcher Stelle innerhalb des ***body***-Elements die Einfügung passiert. Um Verwirrung zu vermeiden, bietet sich aber ein Einfügen gleich zu Beginn an.

```
<body>
    <div class="menu">Hauptmenü</div>
```

Durch die Festlegung des ***class***-Attributs kann nun recht einfach die absolute Formatierung erfolgen:

```
.menu {
    position: absolute;
    top: 0px;
    left: 0px;
    width: 100%;
    height: 30px;
    padding: 10px;
    color: white;
    background-color: black;
}
```

Testen Sie Ihre Web-App!

3.4 Media Queries

Nachdem wir gesehen haben, auf welche Weise die Formatierung eines HTML-Dokuments durch CSS-Regeln beeinflusst werden kann, wenden wir uns der ursprünglichen Fragestellung zu: Wie kann diese Formatierung auf Basis der Bildschirmgröße festgelegt werden?

Die Antwort ist recht einfach: Durch sog. ***Media Queries*** kann die Gültigkeit von CSS-Regeln auf bestimmte Ausgabeformate beschränkt werden. Media Queries können sich auf einzelne Regeln eines CSS beziehen, indem diese Regeln im CSS mit geschweiften Klammern eingefasst und die Query mit ***@media*** vorangestellt wird.

Wir möchten bei kleinen Bildschirmen verhindern, dass unsere Überschrift zweizeilig angezeigt wird und in diesem Fall lieber auf den Nachsatz ***Ein Demoprojekt...*** verzichten. Dies kann erreicht werden durch das Einfügen folgender CSS-Regel:

```
@media screen and (max-width: 500px) {
    h2 small {
        display: none;
    }
}
```

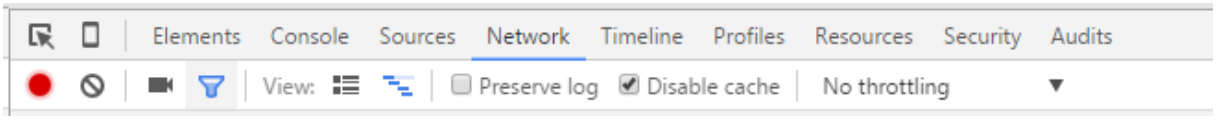
Durch diese Media Query wird bei Bildschirmen mit einer maximalen Breite von 500 Pixeln unser Nachsatz der Überschrift ganz einfach unsichtbar gemacht.

Testen Sie Ihre App und verändern Sie die Breite Ihres Browserfensters!

3.5 Ausschalten des Browser-Cache

Sollten Sie den Eindruck haben, dass Ihre Änderungen an den CSS-Dateien etc. nicht oder erst verzögert beim Browser ankommen, dann kann das daran liegen, dass der Browser die alten Stände gespeichert hat und nicht bei jedem Start aktualisiert. Dieses im Produktivbetrieb gewollte Feature ist während der Entwicklung sehr störend. Sie können es aber in den Entwicklertools des Chrome-Browsers dauerhaft unterbunden.

Wechseln Sie dazu in den bisher nicht genutzten Bereich ***Network*** und setzen Sie ein Häkchen bei ***Disable cache***.



Da wir gerade im Bereich Network sind – Sie sehen hier u.a. die Dauer der einzelnen Ladevorgänge aller in das gerade angezeigte Dokument eingebundener Dateien und können ggf. Optimierungen vornehmen.

Name	Status	Type	Initiator	Size	Time	Timeline – Start Time
index.html	200	document	Other	913 B	5 ms	
mobile.css	200	stylesheet	index.html:6	780 B	9 ms	
es6-shim.min.js	200	script	index.html:15	20.1 KB	78 ms	
jquery-1.12.0.min.js	200	script	index.html:16	42.8 KB	94 ms	
landung.js	200	script	index.html:18	1.2 KB	70 ms	

Untersuchungen bei Shopsystemen haben gezeigt, dass die Ladedauer einer Seite entscheidenden Einfluss auf die Kaufbereitschaft des Kunden hat. Daher ist die Optimierung der Ladezeiten im E-Commerce ein wichtiges Thema.

3.6 Bootstrap

Mit den kennengelernten Hilfsmitteln ist es also möglich, Layout und Aussehen eines HTML-Dokuments in Abhängigkeit der Bildschirmabmessungen zu definieren. Sie können sich aber sicher vorstellen, dass dies recht aufwändig und unübersichtlich werden kann.

Daher werden wir im Folgenden eine weitere Bibliothek einbinden, die die Entwicklung von responsiven Webseiten vereinfacht: ***Bootstrap***. Anders als die bisher verwendeten Bibliotheken besteht Bootstrap jedoch nicht nur aus einer JavaScript-Datei mit dem Namen ***bootstrap.min.js***, die Sie wie gewohnt aus dem eLearning-Kurs herunterladen und dem ***lib***-Verzeichnis hinzufügen können, sondern auch es umfasst auch eine CSS-Datei ***bootstrap.min.css***, die Sie an gleicher Stelle finden und in das Verzeichnis ***css*** importieren.

Doch damit nicht genug: Bootstrap beinhaltet auch noch typische Symbole wie z.B. ein Mülleimer-Icon, die wir später auch in unserer Web-App benötigen. Dazu steht ein eigener Zeichensatz namens ***Glyphicons*** bereit, der in verschiedenen Formaten ebenfalls in unser Projekt eingebracht werden muss. Laden Sie dazu bitte alle entsprechenden Dateien aus dem eLearning-Kurs herunter und fügen Sie diese einem unterhalb von ***app*** neu angelegten Verzeichnis ***fonts*** hinzu.

Natürlichen müssen sowohl die CSS-Datei als auch die JavaScript-Datei in der Startseite ***index.html*** eingebunden werden. Die CSS-Datei in der bekannten Form im ***head***-Element, die JavaScript-Datei wie gehabt im ***body***-Element. Während die genaue Platzierung der CSS-Einbindung egal ist, sollte die JavaScript-Datei nach jQuery und vor den eigenen JavaScript-Dateien geladen werden.

```
<script src="lib/es6-shim.min.js"></script>
<script src="lib/jquery-3.3.1.min.js"></script>
<script src="lib/bootstrap.min.js"></script>
<!-- <script src="scripts/app.js"></script> -->
<script src="scripts/landung.js"></script>
```

Eine Einbindung der Glyphicons ist nicht notwendig.

3.7 Navbar

Als erstes werden wir unsere begonnene Menüleiste durch eine Bootstrap ***Navbar*** (Navigation Bar, Navigationsleiste) ersetzen. Dazu führt Bootstrap ein neues HTML-Element ***nav*** ein, das vom Browser zwar geladen werden kann, mit dem er aber zunächst nichts anfangen kann. Nach dem Ladevorgang sorgt dann das Bootstrap-JavaScript zusammen mit dem Bootstrap-CSS für das gewünschte Verhalten. Entfernen Sie also unser bisheriges ***div***-Element und verwenden Sie:

```
<nav class="navbar navbar-inverse navbar-fixed-top">
</nav>
```

Allerdings ist die Navbar bisher nicht beschriftet. Soll innerhalb der Navigationsleiste etwas angezeigt werden, so ist dies in das Element einzufügen. In Bootstrap sollten Inhalte allerdings immer in ein ***div*** mit der Klasse ***container*** verpackt sein, damit später die Regeln für Responsive Web Design greifen können. Außerdem markieren wir unseren Text noch mit der Klasse ***navbar-brand***, damit er mit einem schönen Hover-Effekt dargestellt wird:

```
<nav class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
        <span class="navbar-brand">
            Patientenverwaltung
        </span>
    </div>
</nav>
```

Auf diese Weise haben wir wieder das bisherige Layout nachgebaut; die alte CSS-Regel zu ***.menu*** könnte jetzt entfernt werden. Aber was haben wir durch Bootstrap gewonnen?

In Bootstrap können wir die Navigationsleiste aus mehreren Bereichen zusammensetzen, die – je nach Breite des Bildschirms – mal unmittelbar angezeigt werden oder erst durch einen Klick aufgeklappt werden müssen.

Wir wollen das ausprobieren durch das Hinzufügen von zwei Menüpunkten, wobei der erste zu Google und der zweite zu Homepage der TH führen sollen. Die verwendeten ***class***-Bezeichnungen und Attributnamen hier und später ergeben sich aus der Dokumentation der Bootstrap Navbar.

```
<nav class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
        <span class="navbar-brand">
            Patientenverwaltung
        </span>
        <ul class="nav navbar-nav">
            <li><a href="http://www.google.de">Google Suchmaschine</a></li>
            <li><a href="http://www.th-nuernberg.de">TH Nürnberg</a></li>
        </ul>
    </div>
</nav>
```

Das sieht jetzt zwar schon ganz gut aus, aber beim Stauchen der Breite wird die Navigationsleiste unschön umgebrochen. Dies kann vermieden werden, indem der Bereich mit den Menüpunkten als kollabierbar ausgezeichnet wird. Dies geschieht durch ein weiteres ***div***-Element mit einer entsprechenden ***class***-Angabe.

```
<nav class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
        <span class="navbar-brand">
            Patientenverwaltung
        </span>
        <div id="menu" class="collapse navbar-collapse">
            <ul class="nav navbar-nav">
                <li><a href="http://www.google.de">Google Suchmaschine</a></li>
                <li><a href="http://www.th-nuernberg.de">TH Nürnberg</a></li>
            </ul>
        </div>
    </div>
</nav>
```

Der Haken an der Sache: im kollabierten Zustand, der ja auf einem kleinen Smartphone permanent gegeben sein kann, sind die Menüpunkte nun nicht mehr auswählbar. Als Lösung werden wir eine Schaltfläche vorsehen, die nur bei ausgeblendetem Menü erscheint und die bei Betätigung die Menüpunkte aufklappt.

```
<nav class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
        <span class="navbar-brand">
            Patientenverwaltung
        </span>
        <button type="button" class="navbar-toggle collapsed"
                data-toggle="collapse" data-target="#menu">
            Klick!
        </button>
        <div id="menu" class="collapse navbar-collapse">
            <ul class="nav navbar-nav">
                <li><a href="http://www.google.de">Google Suchmaschine</a></li>
                <li><a href="http://www.th-nuernberg.de">TH Nürnberg</a></li>
            </ul>
        </div>
    </div>
</nav>
```

Diese Lösung bietet schon die gewünschte Funktionalität, allerdings ist das Layout noch etwas unschön. Durch eine weitere div-Klammer um die stets sichtbaren Elemente wird erreicht, dass für die aufgeklappten Menüpunkte die gesamte Bildschirmbreite zur Verfügung steht.

```
<nav class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
        <div class="navbar-header">
            <span class="navbar-brand">...</span>
            <button class="navbar-toggle collapsed">...</button>
        </div>
        <div id="menu" class="collapse navbar-collapse">
            <ul class="nav navbar-nav">...</ul>
        </div>
    </div>
</nav>
```

Und schließlich ersetzen wir das simple ***„Klick“*** durch den für diesen Fall üblichen Button mit drei waagerechten Streifen.

```
<nav class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
        <div class="navbar-header">
            <span class="navbar-brand">...</span>
            <button type="button" class="navbar-toggle collapsed"
                    data-toggle="collapse" data-target="#menu">
                <span class="sr-only">Toggle navigation</span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
        </div>
        <div id="menu" class="collapse navbar-collapse">...</div>
    </div>
</nav>
```

Dies ist die von Bootstrap empfohlene Grundformatierung für eine Navigationsleiste. Testen Sie Ihre App!