Orientation in Objects

) Bootstrap und LESS)

Version 1.1.0 (01.03.2018, Autor: Frank Bongers, Webdimensions.de)

Weinheimer Straße 68
68309 Mannheim
http://www.oio.de

Inhaltsverzeichnis

1. Seminar Bootstrap / LESS – Themenverortung

Die Gestaltung von webbasierten Userinterfaces mit HTML und CSS ist eine komplexe Angelegenheit. Unter den zahlreichen Frameworks, die dem Designer und Programmierer bei der Arbeit zur Seite stehen, nimmt **Bootstrap** von *Twitter* eine Sonderstellung ein: Es ist das meistverwendete Framework dieser Zielrichtung und bietet neben responsiven Layouthelfern wie Grids auch fertig gestaltete Oberflächenelemente „out of the box“ bis hin zu JavaScript-gesteuerten Widgets. Der Entwicklungsprozess beim Gestalten von Oberflächen wird durch Tools wie Bootstrap gewaltig vereinfacht und beschleunigt.

✔ Die zum Umgang mit Bootstrap benötigten Kenntnisse über die Bootstrap-Klassen, Grids, Formular-, Button- und andere Layoutklassen, sowie einige ausgewählte Bootstrap-Widgets werden in diesem Seminar vermittelt.

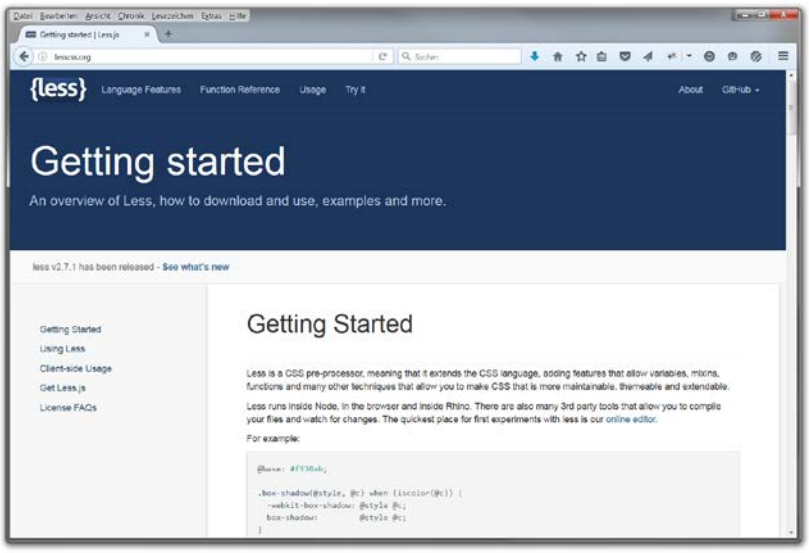
Behandelt wird **Bootstrap 3.3.7**.

Die hinter dem Bootstrap-Framework stehenden *CSS-Deklarationen* sind äußerst umfangreich und unübersichtlich, was zum Problem wird, sobald die von Bootstrap vorgegebene Gestaltung abgewandelt werden muss, um sie beispielsweise an eine Firmen-CI anzupassen. Statt, was tatsächlich aufwendig wäre, die CSS-Sourcen *direkt* anzupassen, können aber auch auf in **LESS** oder SASS definierte *Precompiler-Sourcen* zurückgegriffen werden. Diese können verhältnismäßig einfach angepasst und zu einem angepassten Bootstrap-CSS kompiliert werden.

✔ Die für die Arbeit an den Bootstrap-Sourcen erforderlichen Kenntnisse in LESS werden im Rahmen dieses Seminars ebenfalls vermittelt.

2. Einführung in LESS

LESS wird entwickelt von *Alexis Sellier*, besser bekannt als *cloudhead*. (`www.cloudhead.io`). LESS stellt eine *Erweiterung* (Superset) für CSS dar und ist deswegen nicht nur abwärtskompatibel, sondern nutzt CSS Syntax für zusätzliche Features.



✔ **Siehe:** http://lesscss.org

2.1. Was ist LESS?

LESS ist eine **CSS-Precompilersyntax** und erweitert die Möglichkeiten von CSS mit dynamischem Verhalten in Form von Variablen, Mixins, Berechnungen und Funktionen.

Üblicherweise wird LESS nach reinem CSS *kompiliert*, kann aber (unter Performanceverlust) auch mit Hilfe eines Interpreters sowohl client- als auch serverseitig *direkt* verwendet werden.

✔ Die aktuelle Version ist **LESS v3.0.1**

2.2. LESS kompilieren

Da eine LESS-Datei von der Syntax von einer CSS-Datei abweicht, kann sie nicht direkt genutzt werden, sondern muss kompliert, also in „echtes“ CSS umgewandelt werden. Dies ist die Aufgabe des LESS-Compilers, der als Node-Modul installiert werden muss:

```
npm install -g less
```

Hierfür muss bereits eine globale Installation von Node vorliegen. Erstens, weil sonst der NPM zum Laden der Komponente nicht verfügbar ist, zweitens, weil Node auch die Laufzeit für den Compiler darstellt.

Überprüfen Sie die korrekte Installation, indem Sie einfach die Version des Compilers abfragen:

```
lessc -version
> lessc 3.0.1 (Less Compiler) [JavaScript]
```

Ein LESS-Stylesheet kann bei globaler Installation des Compilers von jedem Verzeichnis aus an der Kommandozeile kompiliert werden:

```
lessc styles.less styles.css
```

Dies kompiliert eine Datei *styles.less* und erzeugt dabei im gleichen Verzeichnis eine Datei *styles.css*.

2.2.1. LESS direkt einsetzen

Für Testzwecke mag es interessant sein, eine **LESS-Datei unmittelbar im Browser** einzusetzen. Hierfür muss zusätzlich eine JavaScriptdatei *less.js* geladen werden, die die Verarbeitung vornimmt.

> Es ist nachvollziehbar, dass dies mit Performanceeinbußen einhergeht, weshalb von einem Produktiveinsatz dieser Lösung abzuraten ist.

Im Head sind hierfür zuerst die LESS-Source und anschließend die JavaScript-Datei einzubinden:

```
<link rel="stylesheet/less"
      type="text/css" href="styles.less">
<script src="less.js" type="text/javascript"></script>
```

✔ Zu beachten ist das `rel`-Attribut, dass den Link zu einem Link auf eine LESS-Datei erklärt.

2.3. Variablen

Mit Variablen können mehrfach im Stylesheet gebrauchte Werte an einer zentralen Stelle definiert und anschließend überall referenziert werden. Änderungen sind dadurch schneller und fehlerfrei möglich.

```
/* ----------------- LESS ----------------- */
@color: #4D926F;
#header {
  color: @color;
}
h2 {
  color: @color;
}


/* ------------ Generiertes CSS ------------ */

#header {
  color: #4D926F;
}
h2 {
  color: #4D926F;
}
```

Im Zuge der Definition von Variablen können auch **Berechnungen** angestellt und andere Variablen einbezogen werden:

```
/* ----------------- LESS ----------------- */

@stahlblau: #5B83AD;
@hellblau: (@hellblau + #111);

#header {
   color: @hellblau;
}

/* ------------ Generiertes CSS ------------ */

#header {
```

```
   color: #6c94be;
}
```

Der Scope von Variablen funktioniert in LESS analog zu dem in Programmiersprachen üblichen Verfahren: Zuerst werden Variablen und Mixins *lokal* (im aktuellen **Block**) gesucht. Werden sie nicht gefunden, geht die Suche im *nächst höheren* Geltungsbereich weiter.

```
@var: red;

#page {
  @var: white;
  #header {
    color: @var; // white
  }
}

#footer {
  color: @var; // red
}
```

2.4. Kommentare

Die üblichen CSS Kommentare gelten in LESS ebenfalls.

```
/* Ich bin ein CSS Kommentar. Und ein LESS Kommentar */
.class {    color: black   }
```

✔ Ein solcher Kommentar wird ins erzeugte CSS ausgegeben, sofern nicht eine minifizierterte Datei erzeugt werden soll.

In LESS können auch **einzeilige Kommentare mit Doppelslash** eingesetzt werden, wie sie aus Javascript bekannt ist.

```
// Kommentar nur im LESS-Code sichtbar
.class {    color: white    }
```

✔ **Praktisch:** Die einzeiligen Kommentare werden **nicht** ins kompilierte CSS übernommen.

2.5. Mixins

Mit **Mixins** können alle Werte einer Klasse in eine andere Klasse eingefügt werden, wofür lediglich der Name der einzufügenden Klasse anzugeben ist. Das Prinzip entspricht dem bei Variablen, ist jedoch auf komplette Regelsätze erweitert.

```
/* ----------------- LESS ----------------- */

.bordered {
   border-top: dotted 1px black;
   border-bottom: solid 2px black;
}

#menu a {
   color: #111;
   .bordered;   // einfach Klassenselektor anführen
}

.post a {
   color: red;
   .bordered;
}
```

Die Werte der `bordered` Klasse tauchen nun wie durch Magie in `#menu a` und `.post a` auf:

```
/* -------------- Generiertes CSS -------------- */

#menu a {
   color: #111;
   border-top: dotted 1px black;
   border-bottom: solid 2px black;
}

.post a {
   color: red;
   border-top: dotted 1px black;
   border-bottom: solid 2px black;
}
```

✔ **Anmerkung:** Jede beliebige CSS Klasse, aber auch id-Regeln können so verwendet werden.

2.6. Mixins mit Parametern

LESS bietet in Form der **parametrierten Mixin-Klassen** eine spezielle Abwandlung von Klassen, die eher Funktionsdeklarationen ähneln. Sie verfügen über Parameterklammern, in denen LESS-Variablen deklariert werden, die bei der Referenzierung übergeben und in den Regelblock weitergegeben werden.

✔ Diese Mixin-Klassen sind allein für den Einsatz in LESS-Sourcen gedacht und werden in den Compile-Vorgang *nicht* einbezogen.

Sie werden ansonsten wie gewöhnliche Klassen als Mixin eingesetzt, wobei aber Argumente übergeben werden.

```
/* ----------------- LESS ----------------- */

// akzeptiert einen Parameter @radius ...
.border-radius (@radius) {
    // ... der im Inneren verwendet wird:
    border-radius: @radius;
   -moz-border-radius: @radius;
   -webkit-border-radius: @radius;
}
```

Ein Beispiel für den Einsatz in der LESS-Source:

```
/* ----------------- LESS ----------------- */

#header {
   .border-radius(4px); // Mixin mit Argument
}

.button {
   .border-radius(6px);  // Mixin mit Argument
}
```

Im Ergebnis erscheint die Mixinklasse selbst *nicht*, aber ihre Werte sind in die CSS-Regeln eingeflossen:

```
/* -------------- Generiertes CSS -------------- */

#header {
   border-radius: 4px;
  -moz-border-radius: 4px;
```

```
  -webkit-border-radius: 4px;
}

.button {
   border-radius: 6px;
  -moz-border-radius: 6px;
  -webkit-border-radius: 6px;
}
```

Es können bei der Definition der Mixinklasse auch **Defaultwerte** vorgegeben werden:

```
/* ----------------- LESS ----------------- */

.rounded-corners (@radius: 5px) {
    border-radius: @radius;
   -moz-border-radius: @radius;
   -webkit-border-radius: @radius;
}
```

Die parametrierte Mixinklasse kann nun *ohne* Argument verwendet werden:

```
#header {
   .rounded-corners;       // Kein Argument: 5px
}
```

Es kann aber dennoch optional ein Argument übergeben werden:

```
#footer {
   .rounded-corners(10px);  // 10px
}
```

Nach dem Kompilieren erhält man folgendes CSS:

```
/* -------------- Generiertes CSS -------------- */

#header {
   border-radius: 5px;
  -webkit-border-radius: 5px;
  -moz-border-radius: 5px;
}

#footer {
   border-radius: 10px;
  -webkit-border-radius: 10px;
```

```
  -moz-border-radius: 10px;
}
```

Um eine Klasse ohne Argumente (die also wie eine normale CSS-Klasse aussieht) als nur für Mixins gedacht zu kennzeichnen, kann man sie als parametrierte Mixinklasse *ohne* Parameter definieren. Sie wird dann im Ausgabestylesheet unterdrückt. Ansonsten wird sie wie gewohnt in LESS eingesetzt.

```
/* ----------------- LESS ----------------- */

// leere Parameterklammern -> LESS-Mixin
.wrap () {
   text-wrap: wrap;
   white-space: pre-wrap;
   white-space: -moz-pre-wrap;
   word-wrap: break-word;
}

pre {
   .wrap  // kein Argument; wie Klassen-Mixin
}
```

Im Ergebnis-CSS ist die `.wrap` Klasse nicht vorhanden.

```
/* -------------- Generiertes CSS -------------- */

pre {
  text-wrap: wrap;
  white-space: pre-wrap;
  white-space: -moz-pre-wrap;
  word-wrap: break-word;
}
```

2.7. Gruppierung mit Namespaces

Manchmal ist es sinnvoll Variablen und Mixins zu **gruppieren**, weil dies für bessere Organisation und Abkapselung sorgt. LESS bezeichnet dies als „Namespaces“ und verwendet für diese ID-Selektoren.

Das Ganze ist recht intuitiv. Um *diverse* Mixins und Variablen unter dem Selektor `#gruppe` zu gruppieren, geht man wie folgt vor:

```
/* ----------------- LESS ----------------- */

#gruppe {
  .buttonlook () {
    display: block;
    border: 1px solid black;
    background-color: grey;

    &:hover {
       background-color: white;
    }
  }
  .tab {
      // weiteres Bundle-Mixin
  }
  .citation {
      // weiteres Bundle-Mixin
  }
}
```

Um nun `.buttonlook` zu `#header a` hinzuzufügen, schreibt man folgendes:

```
/* ----------------- LESS ----------------- */

#header a {
   color: orange;
   #gruppe > .buttonlook;
}
```

Dies ergibt folgende Ausgabe, wobei die Mixins aus dem Bundle nicht erscheinen:

```
/* -------------- Generiertes CSS -------------- */

#header a {
  color: orange;
  display: block;
  border: 1px solid black;
  background-color: grey;
}

#header a:hover {
  background-color: white;
}
```

2.8. Verschachtelung von Regeln

Mit LESS kann man neben dem üblichen „cascading“ auch **„nesting“**, also das Verschachteln von Regeln, einsetzen. Anstatt für das Styling ineinander verschachtelter Strukturen entsprechend lange Selektorketten zu schreiben, werden LESS-Selektoren einfach ineinander verschachtelt.

```
/* ----------------- LESS ----------------- */

#header {
  h1 {
    font-size: 26px;
    font-weight: bold;
  }

  p { font-size: 12px;
    a { text-decoration: none;
      &:hover { border-width: 1px }
    }
  }
}
```

Wichtig ist hier der &-Kombinator. Dieser wird als Platzhalter für das Elternelement verwendet, was besonders für die Definition von Pseudoklassen, wie `:hover` und `:focus`, praktisch ist.

```
/* -------------- Generiertes CSS -------------- */

#header h1 {
   font-size: 26px;
   font-weight: bold;
}

#header p {
   font-size: 12px;
}

#header p a {
   text-decoration: none;
}

#header p a:hover {
```

```
   border-width: 1px;
}
```

2.9. Funktionen und Berechnungen

Bei der Zuweisung von Farbwerten kann LESS Berechnungen anstellen, um ausgebend von einer Basisfarbe einen helleren oder dunkleren Ton zu übergeben („darken“ oder „lighten“), oder um einen Farbwert mit einem anderen zu verrechnen. Hierbei stehen Funktionen zur Verfügung, die die Sättigung veränderen („saturate“), oder (etwas weniger intuitiv) den Farbton auf dem Farbrad drehen („spin“).

```
/* ----------------- LESS ----------------- */

@borderbreite: 1px;
@basisfarbe: #111;
@meinrot:        #842210;

#header {
   color: (@basisfarbe * 3);
   border-left: @borderbreite;
   border-right: (@borderbreite * 2);
}

#footer {
   color: (@basisfarbe + #003300);
   border-color: desaturate(@meinrot, 10%);
}
```

Wir erhalten aus dieser Source folgendes CSS:

```
/* -------------- Generiertes CSS -------------- */

#header {
   color: #333;
   border-left: 1px;
   border-right: 2px;
}

#footer {
  color: #114411;
  border-color: #7d2717;
}
```

Es kann mit jeder Zahl, Farbe oder Variable gerechnet werden. Hier einige Beispiele:

```
/* ----------------- LESS ----------------- */

@base: 5%;
@base-color: #aa6633;
@filler: (@base * 2);
@nochwas: (@base + @filler);

.test {
   color: (#888 / 4);
   background-color: (@base-color + #111);
   height: (100% / 2 + @filler);
}
```

Das Ergebnis entspricht der intuitiven Erwartung: LESS versteht den Unterschied zwischen Farben und Maßeinheiten.

```
/* -------------- Generiertes CSS -------------- */

.test {
   color: #222222;
   background-color: #bb7744;
   height: 60%;
}
```

Ist eine Maßeinheit in einer Berechnung enthalten, so behält LESS sie:

```
@var: (1px + 5);
```

Wir erhalten hier 6px.

Auch die Verwendung von Klammern ist möglich:

```
width: (@var + 5) * 2;
```

Auch verschachtelte Klammern sind erlaubt:

```
width: ((@var + 5) * 2);
```

Das folgende Beispiel verwendet die `percentage()`-Funktion, um 0.5 in 50% umzuwandeln, erhöht die Sättigung einer Grundfarbe um 5% (durch `saturate()`) und setzt die Hintergrundfarbe, ausgehend von der Grundfarbe, auf eine Farbe, die um *25% heller* als diese (mittels `lighten()`) und auf dem Farbrad um *8 Grad gedreht* ist (mittels `spin()`):

```
/* ----------------- LESS ----------------- */

@base: #f04615;
@width: 0.5;

.beispiel {
   width: percentage(@width);     // ergibt 50%
   color: saturate(@base, 5%);
   background-color: spin(lighten(@base, 25%), 8);
}
```

Dies ergibt folgendes CSS:

```
/* -------------- Generiertes CSS -------------- */

.beispiel {
  width: 50%;
  color: #f6430f;
  background-color: #f8b38d;
}
```

2.9.1. Mathematische Funktionen

Sollen Zahlenwerte verrechnet werden, so bietet LESS einen Grundstock an einschlägigen Funktionen, die in den LESS-Regeln eingesetzt werden können.

```
ceil(@number);
// Rundet auf zur nächsten Ganzzahl

floor(@number);
// Rundet ab zur nächsten Ganzzahl

percentage(@number);
// Konvertiert nach %, z.B. 0.5 -> 50%

round(number, [places: 0]);
// Rundet zu einer Zahl mit n Nachkommastellen
```

2.9.2. Funktionen für Farbberechnungen

Hier ist der Großteil der LESS-Funktionen angesiedelt. Es gibt eine Gruppe von Funktionen, die Farben verschiedener Farbmodelle generieren:

```
rgb(@r, @g, @b);
// Erstellt aus drei Zahlen eine RGB-Farbe

rgba(@r, @g, @b, @a);
// Erstellt aus vier Zahlen eine RGBA-Farbe

hsl(@hue, @saturation, @lightness);
// Erstellt aus drei Zahlen eine HSL Farbe

hsla(@hue, @saturation, @lightness, @alpha);
// Erstellt aus vier Zahlen eine HSLA-Farbe

hsv(@hue, @saturation, @value);              /
/ Erstellt aus drei Zahlen eine HSV-Farbe

hsva(@hue, @saturation, @value, @alpha);
// Erstellt aus vier Zahlen eine HSVA-Farbe
```

Eine weitere Gruppe von Funktionen extrahiert einen **Kanal** aus einer Farbe eines beliebigen Farbmodells, um dessen Wert beispielsweise einer Variable zuzuweisen:

```
hue(@color);
// Gibt den Hue-Kanal von @color zurück

saturation(@color);
// Gibt  den Saturation-Kanal von @color zurück

lightness(@color);
// Gibt die Lightness-Kanal von @color zurück

red(@color);
// Gibt den Rot-Kanal von @color zurück

green(@color);
// Gibt den Grün-Kanal von @color zurück

blue(@color);
// Gibt den Blau-Kanal von @color zurück
```

```
alpha(@color);
// Gibt den Alpha-Kanal von @color zurück

luma(@color);
// Gibt den Luminiszenz-Kanal von @color zurück
```

Weitere Funktionen erzeugen eine Farbe indem Helligkeit, Sättigung oder weitere Parameter einer übergebenen Farbe verändert werden, oder berechnen aus zwei Farbwerten einen neuen:

```
saturate(@color, 10%);
// Erhöht die Sättigung um 10%

desaturate(@color, 10%);
// Verringert die Sättigung um 10%

lighten(@color, 10%);
// Erhöht die Helligkeit um 10%

darken(@color, 10%);
// Dunkelt um 10% ab

fadein(@color, 10%);
// Erhöht die Opazität um 10%

fadeout(@color, 10%);
// Verringert die Opazität um 10%

fade(@color, 50%);
// Setzt die Transparenz auf 50%

spin(@color, 10);
// Dreht den „Hue“ um 10 Grad auf dem Farbrad

mix(@color1, @color2, [@weight: 50%]);
// Mischt @color1 und @color2

greyscale(@color);
// Wandelt in Graustufenwert gleicher Helligkeit
```

Die letzte Funktionsgruppe **verrechnet zwei Farbwerte** miteinander. Das Procedere gleicht den Ebenenverrechnungen in Photoshop.

```
multiply(@color1, @color2);

screen(@color1, @color2);
```

```
overlay(@color1, @color2);

softlight(@color1, @color2);

hardlight(@color1, @color2);

difference(@color1, @color2);

exclusion(@color1, @color2);

average(@color1, @color2);

negation(@color1, @color2);
```

2.10. Media Queries mit LESS

LESS kann sehr gut mit Media-Queries umgehen und erleichtert das erstellen von CSS, das solche Blöcke einbezieht, enorm.

Media Queries können hierfür genauso wie Selektoren verschachtelt werden. Hier wird die Klasse `.one` für einen Breakpoint definiert und eine Zusatzquery für diesen Breakpoint definiert:

```
/* ----------------- LESS ----------------- */

.one {
    @media (width: 400px) {
        font-size: 1.2em;

        @media print and color {
            color: blue;
        }
    }
}
```

Im generierten Stylesheet wird dies nach `@media`-Blöcken getrennt ausgegeben:

```
/* -------------- Generiertes CSS -------------- */

@media (width: 400px) {
   .one {
     font-size: 1.2em;
```

```
    }
}

@media (width: 400px) and print and color {
    .one {
      color: blue;
    }
}
```

2.11. Import von Dateien in LESS

Analog zu CSS existiert auch in LESS ein `@import`-Statement, mit dem sowohl *.less*-Dateien oder *.css*-Dateien importiert werden können.

✔ Das Verhalten des Prozessors unterscheidet sich jedoch bei beiden Dateitypen voneinander.

Durch Importe werden beispielsweise Variablen und Mixins verfügbar gemacht. Die Nennung der Dateiendung `.less` ist optional:

```
@import "lib.less";
```

Dies entspricht in der Wirkung

```
@import "lib";
```

LESS unterscheidet dabei zwischen LESS- und CSS-Dateien. Eine CSS-Datei muss zwar den LESS-Prozessor nicht durchlaufen, kann aber dennoch importiert werden (natürlich muss die Endung genannt werden!):

```
@import "beispiel.css";
```

Achtung:
Dies führt jedoch nicht, wie vielleicht erwartet wird, dazu, dass LESS den CSS-Code in das Ergebnisdokument *einfügt*, sondern lediglich, dass dieses *Importstatement* ins Ergebnisdokument kopiert wird.

Da das kopierte Importstatement dort, wie es sich gehört, am **Anfang** des Dokuments auftaucht, werden die Regeln an der diesem entprechenden Position (wohl also zu früh!) registriert und können keine der anderen Regeln überschreiben.

Ist die Position des Imports **relevant**, kann man den LESS-Prozessor veranlassen, die CSS-Daten „inline“, d.h. exakt an der Stelle ins Ergebnisdokument **einzufügen**, die der Position des CSS-Imports entspricht.

Hierfür muss beim Import das `inline`-Statement angegeben werden:

```
@import (inline) "beispiel.css";
```

Hierdurch wird der CSS Code aus *beispiel.css* in die Ziel-Datei eingefügt und zwar an der Position, der dem Importstatement entspricht.

Zusammengefasst stellt sich die Lage wie folgt an:

- **Import von LESS-Dateien:**
  Eine importierte LESS-Datei durchläuft den LESS-Prozessor und das Ergebnis wird anstelle des Importstatements in der Ergebnisdatei eingefügt. Dies ermöglicht es, eine beliebige Anzal von LESS-Modulen in kontrollierter Reihenfolge (wichtig!) in ein Ergebnisdokument zu „plätten“.

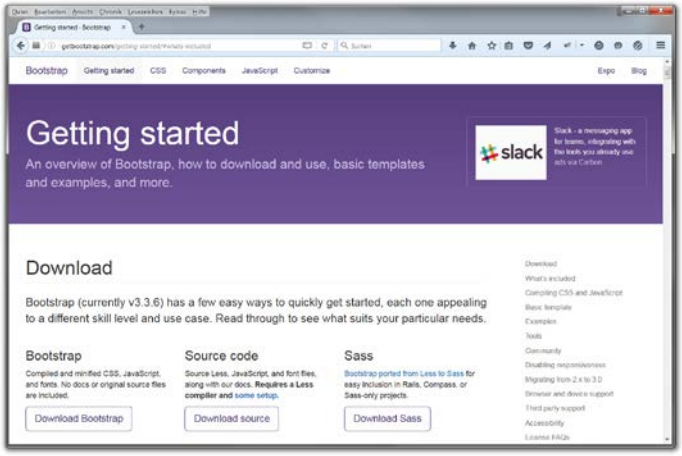
- **Import von CSS-Dateinen mit inline-Statement:**
  Eine mit `inline`-Statement importierte CSS-Datei verhält sich wie eine LESS-Datein. Der Prozessor wird sie zwar nicht verarbeiten, plättet die Datei jedoch in das Ergebnisdokument und platziert ihre Statements exakt dort wo gewünscht.

- **Import von CSS-Datei ohne inline-Statement:**
  Die Datei bleibt *separat* erhalten und wird nicht in das Ergebnisdokument integriert. Vielmehr wird dort ein Importstatement eingefügt, der diese Datei im Kontext des Ergebnisdokuments zur Verfügung stellt. Die Statements des CSS-Imports liegen aber in der CSS-Kaskade vor den Statements des Ergebnisdokuments und können dieses zwar *ergänzen*, aber nicht dessen Regeln *überschreiben*.

3. Einstieg in Bootstrap 3.3.7

3.1. Download und Verwendung von Bootstrap 3.3.7

Die **Dateien** für Bootstrap 3.3.7 können Sie für ein Projekt über verschiedene Methoden erhalten und einbinden, je nachdem ob Sie mit dem Framework einfach nur direkt arbeiten möchten, oder aber mittels der Sourcen eine eigene Version gestalten möchten.



NEU: Bootstrap 4.0

Mittlerweile ist die neue, stark überarbeitete **Version 4.0** des Bootstrap-Frameworks erschienen. Sowohl die *technische Basis* (Flexbox anstelle von Floats) als auch die *Selektorbenennung* (einige CSS-Klassen wurden umbenannt oder entfernt) haben sich geändert. DIe Inhalte dieses Seminars sind daher auf Bootstrap 4.0 nur eingeschränkt übertragbar!

3.1.1. Download von Bootstrap (reines Framework)

Einen Download von Bootstrap 3.3.7 können Sie unter folgender Adresse erhalten.

http://getbootstrap.com/docs/3.3/getting-started/#download

Wenn Sie sich dafür entscheiden, das reine Bootstrap-Framework downzuloaden (linker Button **„Download Boottrap“**), erhalten Sie die CSS-Sourcen, die Icon-Fonts und die erforderlichen JavaScript-Dateien.

Datei: bootstrap-3.3.7-dist.zip

```
bootstrap/
├── css/
│   ├── bootstrap.css
│   ├── bootstrap.css.map
│   ├── bootstrap.min.css
│   ├── bootstrap.min.css.map
│   ├── bootstrap-theme.css
│   ├── bootstrap-theme.css.map
│   ├── bootstrap-theme.min.css
│   └── bootstrap-theme.min.css.map
├── js/
│   ├── bootstrap.js
│   └── bootstrap.min.js
└── fonts/
    ├── glyphicons-halflings-regular.eot
    ├── glyphicons-halflings-regular.svg
    ├── glyphicons-halflings-regular.ttf
    ├── glyphicons-halflings-regular.woff
    └── glyphicons-halflings-regular.woff2
```

✔ In diesem Paket sind *keine* LESS-Sourcen, Dokumentationen oder Beispiele enthalten. Dafür ist es von der Größe (279 kb) relativ kompakt. Auch **jQuery** benötigen Sie ggfs. zusätzlich!

3.1.2. Download Bootstrap Source code

Ein umfangreicheres Paket können Sie ebenfalls über die vorhin genannte Adresse als Zip downloaden: Mittlerer Button, **„Download Source“**.

Das Paket enthält neben den fertigen Frameworkdaten (entsprechend dem Standardpaket) auch die diesem zugrunde liegenden Sourcen in Form von Less-Daten, JavaScript-Sourcen und Fonts. Daneben ist auch eine lokale Version der Dokumentation enthalten.

✔ Zur Verarbeitung der LESS-Sourcen wird ein LESS-Compiler benötigt.

Datei: bootstrap-3.3.7.zip

```
bootstrap/
├── less/          <- die LESS-Sourcen
```

```
├── js/
├── fonts/
├── dist/          <- das fertige Framework
│   ├── css/
│   ├── js/
│   └── fonts/
└── docs/          <- die Dokumentation
    └── examples/
```

✔ Diesen Paket hat einen Umfang von rund 4 MB

3.1.3. Einbinden über Bootstrap CDN

Möchte man Bootstrap nicht selbst hosten, so kann man die erforderlichen Dateien auch aus einem **Repository** einbinden.

✔ Die beiden in den `<link>`- und `<script>`-Elementen sichtbaren Attribute `integrity` und `crossorigin` dienen der Integritätsprüfung. Sie wurden von Bootstrap vorgeschlagen. Derzeit werden die jedoch nur von Chrome und aktuellen Firefox-Builds berücksichtigt.

```
<!-- CSS, minifiziert: bootstrap.min.css -->

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/
   bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-
1q8mTJOASx8j1Au+a5WDVnPi2lkFfwwEAa8hDDdjZlpLegxhjVME1fgjWPGmk
zs7" crossorigin="anonymous">

<!-- Optionales Theme: bootstrap-theme.min.css -->

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/
bootstrap/3.3.7/css/bootstrap-theme.min.css" integrity=
"sha384-fLW2N01lMqjakBkx3l/M9EahuwpSfeNvV630u7EYsXMjQV+0En5r"
crossorigin="anonymous">

<!-- JavaScript, minifiziert: bootstrap.min.js -->

<script src="https://maxcdn.bootstrapcdn.com/
bootstrap/3.3.7/js/bootstrap.min.js" integrity="sha384-
0mSbJDEHialfmuBBQP6A4Qrprq5OVfWLqxss1yVqOtnepnHVP9aJ7xS"
crossorigin="anonymous"></script>
```

3.2. Installation mit NPM

Auch die Installation per NPM ist möglich. Hierbei muss unbedingt die Versionsnummer angegeben werden, da sonst die aktuelle 4er Version des Frameworks installiert wird:

```
npm install bootstrap@3.3.7 --save
```

Zusätzlich wird ggfs. jQuery zum Betrieb der Boostrap-Widgets benötigt. Sollten Sie also Navbar, Caroussel oder Colapsibles einsetzen wollen, so benötigen Sie ein jQuery mit dem Versionsstand 2.x.

```
npm install jquery@2.2.4 --save
```

3.3. Verwendung von Bootstrap im Template

Für den Einsatz von Bootstrap im HTML-Template gibt es ein paar Best-Practice-Regeln. Grundsätzlich ist Bootstrap von der Zielrichtung nicht auf Desktop-Websites beschränkt, sondern soll ebensogut auf Mobile Devices arbeiten können.

✔ Dies berücksichtigt man im HTML-Template durch Einfügen bestimmter Meta-Elemente.

Die CSS-Dateien werden im Dokumentkopf verlinkt, die JavaScript-Sourcen dagegen am Ende des Body-Elements eingebunden (von letzterer Regel kann man abrücken; es ist lediglich die aktuell beliebteste Herangehensweise).

Der Grundaufbau eines **Bootstrap-Templates** ist daher in etwa wie folgt:

```
<!DOCTYPE html>
<html lang="en">
  <head>

    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport"
          content="width=device-width, initial-scale=1">

    <title>Basis Bootstrap Template</title>

    <!-- Bootstrap-CSS -->
```

```
    <link href="css/bootstrap.min.css" rel="stylesheet">

    <!-- ggfs. Fix für ältere IE -->
    <!--[if lt IE 9]>
      <script src="js/html5shiv.min.js"></script>
      <script src="js/respond.min.js"></script>
    <![endif]-->
  </head>
  <body>
    <h1>Hallo Bootstrap!</h1>

    <!-- jQuery für Bootstrap JavaScript-Plugins -->
    <script src="js/jquery.min.js"></script>
    <!-- Bootstrap-JS oder Einzelfiles der Plugins -->
    <script src="js/bootstrap.min.js"></script>
  </body>
</html>
```

Die drei Meta-Tags haben folgende Aufgabe:

- **charset:**
  Definiert das Encoding des Dokuments. Üblicherweise *UTF-8*.

- **X-UA-Compatible:**
  Schaltet IE ggfs. aus dem Back-Compat-Mode in den Standard-Renderingmode „edge“ um. Funktioniert *nicht* im Rahmen von Intranets!

- **viewport:**
  Skalierungs- und Darstellungsanweisungen für die Mobildarstellung. Keine Wirkung im Desktop-Browser, daher gefahrlos. Pflicht.

Das CSS. Wird in jedem Falle benötigt:

- **bootstrap.min.css:**
  Enthält alle Bootstrap-CSS-Regeln

Das JavaScript im Head; wird nur für IE < 10 benötigt. Sollten ältere IE nicht unterstützt werden müssen, so kann dieser Teil des Templates entfallen:

- **html5shiv.min.js** (in Conditional Comments):
  Stellt HTML5-Kompatibilität sicher.
  Nicht Teil des Bootstrap-Download-Pakets.

✔ **respond.min.js** (in Conditional Comments):
Polyfill für *MediaQueries* im IE 6-8; wird für Responsive Design benötigt. Nicht Teil des Bootstrap-Download-Pakets. (Sollte auch kein Thema mehr sein.)

Das JavaScript (im Body). Kann entfallen, falls keine Bootstrao-Widgets verwendet werden:

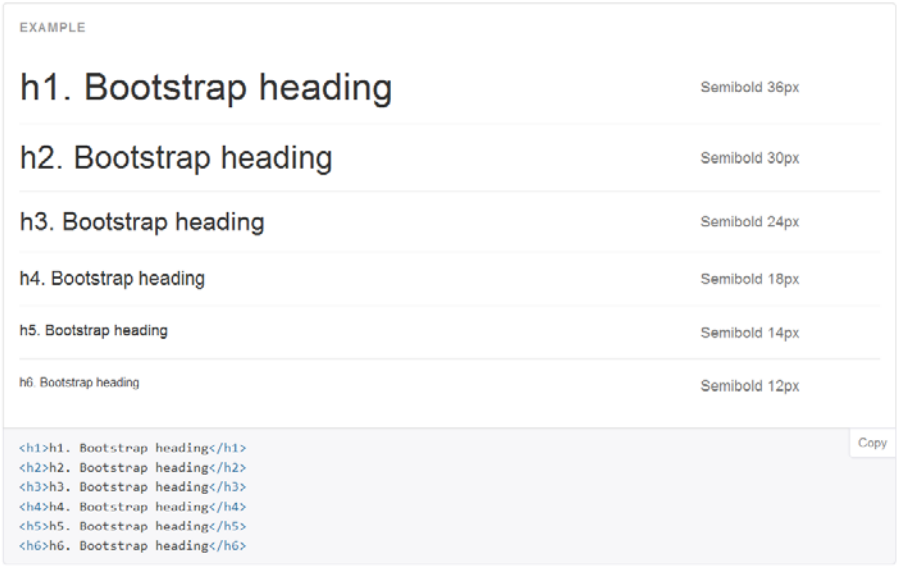
✔ **jquery.min.js:**
Erforderlich als Basis für einige der Bootstrap-Widgets. **Muss** für Bootstrap 3 eine Version *vor* jQuery 3.0 sein (z.B. **jQuery 2.2.x**). Achtung, jQuery ist nicht Teil des Bootstrap-Download-Pakets!

Sie erhalten es unter: https://code.jquery.com/

✔ **bootstrap.min.js:**
Das JavaScript zum Betrieb der Bootstrap-Widgets. Setzt auf jQuery auf.

3.4. Typographie mit Bootstrap

Bootstrap greift bereits auf der Ebene der Typographie ein und stellt einige **Basisstyles** zur Verfügung. Dies betrifft die Hintergrundfarbe des Body (explizit auf „white“ gesetzt), Farben für Text und Links, sowie die Basisschriftgröße, Font-Family und Größenverhältnisse zwischen Überschriften, Textabsätzen und weiteren Elementen.



Bei der Bootstrap-Typographie gibt es ein paar Auffälligkeiten. Bootstrap setzt Basisschrifteigenschaften für den **Body**. Hierbei wird die `font-size` (Basisschriftgröße) auf `14px` gesetzt, die Zeilenhöhe `line-height` auf `1.43em`. Die Schriftfarbe ist *dunkelgrau* statt reinschwarz, es ist eine *serifenlose* anstelle einer Serifenschrift gefordert (Default: Times o.ä.).

```
body {
  font-family: Helvetica, Arial, sans-serif;
  font-size: 14px;
  line-height: 1.42857143;
  color: #333;
  background-color: #fff;
}
```

Die Schriftgröße ist damit etwas geringer als der Browser-Default, die Zeilenhöhe dagegen etwas größer. Die Bootstrap-Typographie wirkt so ein wenig luftiger.

Das Margins der **Textabsätze** werden genullt bis auf das Bottom-Margin, das mit 10px obendrein viel geringer ist als der Browser-Default:

```
p {
  margin: 0 0 10px;
}
```

Überschriften erhalten hingegen ein Top-Margin, das Bottom-Margin ist ebenfalls konstant 10px. Auch diese Werte sind anders bzw. kleiner als die Defaults.

```
h1, h2, h3 {
  margin-top: 20px;
  margin-bottom: 10px;
}

h4, h5, h6 {
  margin-top: 10px;
  margin-bottom: 10px;
}
```

Die Schriftgröße ist in Pixel angegeben (sonst in em üblich) und insgesamt **größer** als die Defaults (h1: 36px gegenüber 2em, was ca. 32px entspricht).

```
h1 {
  font-size: 36px;
}

h2 {
  font-size: 30px;
}

h3 {
  font-size: 24px;
}
```

✔ Um eine **gleichmäßige Layoutbasis** über alle Browserplattformen hinweg zu gewährleisten, ist das **normalize.css-Projekt** (von *Nicolas Gallagher* und *Jonathan Neal*) in das Bootstrap-CSS integriert. Beispielsweise werden die Margins des Body genullt (u.v.a. mehr).

Typographische Klassen:

```
.text-left   .text-right   .text-center   .text-justify
.h1    .h2   .h3   .h4   .h5   .h6
.small  .lead
```

3.5. Tabellen und Listen mit Bootstrap

Bootstrap bietet ein Grundstyling für Formulare, Tabellen und Listen an, das zum Teil ohne weiteres Addieren von Klassen greift.

3.5.1. Listen

Für die UL und OL-Elemente sind die Top-Margins genullt, die Bottom-Margins auf 10px gesetzt. Das Padding-Left wurde nicht angetastet.

- Lorem ipsum dolor sit amet
- Consectetur adipiscing elit
- Integer molestie lorem at massa
- Facilisis in pretium nisl aliquet
- Nulla volutpat aliquam velit
  - Phasellus iaculis neque
  - Purus sodales ultricies
  - Vestibulum laoreet porttitor sem
  - Ac tristique libero volutpat at

Klassen für Listen:	
.list-unstyled	.list-inline

3.5.2. Tabellen

Tabellen-Styles müssen hingegen über die Klasse „table“ an der Tabelle extra aktiviert werden. Durch Zusatzklassen wie „table-striped“ und „table-bordered“ lassen sich weitere Layouteigenschaften hinzufügen.

```
<table class="table table-bordered table-striped">
  <tr>
   <td>Testdaten</td>
   ...
  </tr>
</table>
```

Eine Tabelle mit Bootstrap-Styles

Testdaten	Testdaten	Testdaten	Testdaten
Testdaten	Testdaten	Testdaten	Testdaten
Testdaten	Testdaten	Testdaten	Testdaten

Abb: Eine Zebra-Tabelle mit „table-striped“

Klassen für Tabellen:
```
.table
.table-striped    .table-bordered .table-hover
.table--condensed .table-responsive
```

3.6. Helferklassen

Für verschiedene Aufgaben stellt Bootstrap sogenannte **Helferklassen** zur Verfügung, die auf (beinahe) beliebige Elemente angewendet werden können, um *Hervorhebungen* vorzunehmen (Farben) oder eine bestimmte *Ausgabe* (Screenreader) zu erreichen. Auch können sehr einfach *Floats* ausgesprochen oder Inhalte *zentriert*, *gezeigt* oder *versteckt* werden.

Helfer-Klassen:
```
.pull-left   .clearfix  .center-block
.pull-right  .show      .hidden    .invisible
.sr-only     . sr-only-focusable
```

3.6.1. Helferklassen für Screenreader

Die Klasse `.sr-only` verbirgt ein Element vor allen Geräten außer Screenreadern. Kombiniert mit `.sr-only-focusable` wird ein Element angezeigt, wenn es fokussiert wird (z.B. über eine Tastatur).

Hier ein Sprungmenü („Skiplink“) wie er Screenreadern zum Überspringen einer (längeren) Navigation als Direktzugang zu den Inhalten zur Verfügung gestellt wird:

```
<a class="sr-only sr-only-focusable"
   href="#content">
   Zum Hauptinhalt springen
</a>
```

✔ Die Berücksichtigung von Screenreadern ist erforderlich, um den optimalen Vorgehensweisen für Barrierefreiheit Folge zu leisten.

3.6.2. Floats

Bootstrap bietet zwei eigene Klassen, die dort eingesetzt werden können, wo ein Interface-Element “mal eben schnell” gefloatet werden muss.

```
<div class="pull-left"> Links gefloatet! </div>
<div class="pull-right">Rechts gefloatet!</div>
```

Dies kann beispielsweise ein Image in einem Fließtext sein. Selbstverständlich könnte man ein `style`-Attribut verwenden, um das gewünschte Ergebnis zu erzeilen:

```
<p>Lorem ipsum dolor sit amet,
  <img src="illu.jpg" style="float:left;">
   consectetur adipisicing elit.  Amet corporis dolores
   ducimus quibusdam, rem sequi!</p>
```

Dies ist jedoch unschön, schwer wartbar und unter Umständen auch nicht von außen widerrufbar. Besser ist es, hier die **Bootstrap-Klasse** „pull-left“ einzusetzen:

```
<p>Lorem ipsum dolor sit amet,
  <img src="illu.jpg" class="pull-left">
   consectetur adipisicing elit.  Amet corporis dolores
   ducimus quibusdam, rem sequi!</p>
```

Die Klassen sind wie folgt definiert (beachten Sie das `!important`-Flag):

```
.pull-left {
   float: left !important;
}


.pull-right {
   float: right !important;
}
```

3.6.3. Zentrieren von Blöcken

Um ein Element **innerhalb** eines Blockes horizontal zu zentrieren, stellt Bootstrap die Klasse `.center-block` zur Verfügung. Die Anwendung ist einfach:

```
<p>
   <img class="center-block" src="bild.jpg">
</p>
```

Die Klasse verwendet den Wert `auto` für das rechte und linke Margin. Eine Zentrierung ist davon abhängig, dass der Elterncontainer dem zu zentrierenden Element entsprechend Raum gibt (das zu zentrierende Element darf also nicht auf 100% Breite gesetzt sein). Ein Blockelement mit impliziter Breite kann also nicht auf diesem Weg zentriert werden!

Die Klasse ist wie folgt definiert:

```
.center-block {
   display: block;
   margin-left: auto;
   margin-right: auto;
}
```

Tipp: Wenn Sie Text (oder Inline-Elemente wie Bilder) *innerhalb* eines Block zentrieren wollen, so setzen Sie stattdessen am Block die Klasse `text-center`.

```
<p class="text-center">
   <img src="img/bild.jpg" alt="">
</p>
```

3.6.4. Zeigen und verstecken

Die Klassen `.show` und `.hidden` erzwingen, dass ein Element gezeigt oder ausgeblendet wird (Screenreader inbegriffen). Versteckte Elemente werden dabei aus dem Flow entfernt. Alternativ versteckt die Klasse `.invisible` ein Element und belässt es dabei im Flow (d.h. es nimmt weiterhin Platz im Layout ein).

```
<div class="show">...</div>
<div class="hidden">...</div>
<div class="invisible">...</div>
```

Die Klassen sind nur für das Ein- und Ausblenden von Blockelementen verfügbar.

Die Klassen sind wie folgt definiert:

```
.show {
   display: block !important;
}

.hidden {
   display: none !important;
}

.invisible {
   visibility: hidden;
}
```

✔ Diese Klassen verwenden das Flag `!important`, um Probleme mit Spezifität zu vermeiden. Sie haben auf diese Weise Vorrang zu anderen denkbaren Aussagen zur Sichtbarkeit des Elements.

3.6.5. Text- und Hintergrundfarben

Um einem Text mittels einer **Textfarbe** eine Bedeutung zuzuweisen, können die Hilfsklassen für Kontextfarben verwendet werden. Entsprechend markierte Texte folgen stets dem entsprechenden Farbscgema von Bootstrap.

Texte in Kontextfarben

```
<p class="text-muted">...</p>
<p class="text-primary">...</p>
<p class="text-success">...</p>
<p class="text-info">...</p>
<p class="text-warning">...</p>
<p class="text-danger">...</p>
```

Änliches ist für die Hintergrundfarben von Containern möglich.

Hintergründe für Container in Kontextfarben

```
<p class="bg-primary">...</p>
<p class="bg-success">...</p>
<p class="bg-info">...</p>
<p class="bg-warning">...</p>
<p class="bg-danger">...</p>
```

Klassen für Text und Hintergrund:

```
.text-muted
.text-danger      .text-info      .text-primary
.text-success          .text-warning
.bg-danger        .bg-info        .bg-primary
.bg-success       .bg-warning
```

4. Das Grid-System von Bootstrap

Ein **Gestaltungsraster** (Grid) erleichtert die Platzierung von Inhalten im Rahmen eines Layouts mittels einer **Zeilen/Spalten**-Metapher. Hierfür wird über den Screen ein gedachtes Raster gelegt, dass die Breite des Gestaltungsraums in virtuelle Spalten teilt.

✔ Spaltencontainer (oder auch Inhalte) werden anschließend anhand des Spaltenrasters mit Breitenbemessungen versehen und platziert.

	Extra-kleine Geräte Smartphones (<768px)	Kleine Geräte Tablets (≥768px)	Mittlere Geräte Desktop-PCs (≥992px)	Große Geräte Desktop-PCs (≥1200px)
Raster-Verhalten	Durchgehend horizontal	Zuerst minimiert, über Umbruchpunkten horizontal		
Container-Breite	Keine (auto)	750px	970px	1170px
Klassen-Präfix	`.col-xs-`	`.col-sm-`	`.col-md-`	`.col-lg-`
Spaltenanzahl	12			
Spaltenbreite	Auto	~62px	~81px	~97px
Abstandsbreite	30px (15px auf jeder Seite einer Spalte)			
Verschachtelbar	Ja			
Abrückung	Ja			
Spalten-Umordnung	Ja			

4.1. Ein zwölfspaltiges Grid-System

Bootstrap verwendet ein, wie dies ausgedrückt wird, „zwölfspaltiges“ Grid-System, unterteilt den Gestaltungsbereich also in der Horizontalen in zwölf Teile, die als „Spalten“ (*columns*) bezeichnet werden.

✔ Natürlich wird damit kein, im Sinne des Wortes, „zwölfspaltiges“ Layout erzeugt. Vielmehr ist es ein **12er Raster aus Einheiten**, was

beispielsweise ein *zweispaltiges* Layout durch Zuweisung von je 6 Einheiten, ein *dreispaltiges* durch die Zuweisung von je 4 Einheiten an die Spaltencontainer erreicht.

✔ Jede Spaltenzahl und -konfiguration ist möglich, solange pro Layoutzeile die Summe von 12 Einheiten nicht überschritten wird.

Die **effektive Breite** dieser Spalten richtet sich nach dem verfügbaren Raum, ist also von der Bildschirm- oder Viewportbreite abhängig. Ob das Layout bildschirmfüllend oder mit seitlichem Rand dargestellt wird, richtet sich nach der Art des **Containerelements**, mit dem das Layout umgeben wird.

✔ In die Regeln für die Gridklassen sind **Media-Queries** einbezogen, sodass das Gestaltungsraster am Ende auch „responsive“ ist. Das Layoutraster kann dabei für vier verschiedene **Geräteklassen** (*klein*, *mittel*, *groß* und *sehr groß*) unterschiedlich angegeben werden.

4.2. Ein einfaches Grid-Layout

Bootstrap empfiehlt, dass die Zeilen und Spalten eines Layouts mit einem „Container“-Wrapper umgeben werden sollten, um korrekte Ausrichtung und Abstände zu gewährleisten. Hierfür stehen zwei CSS-Klassen zur Verfügung, nämlich „container“ und „container-fluid“.

Bei der Klasse „container“ handelt es sich um Container mit statischer Breite, die zwar auch resonsive sind, deren Breite aber *fix* ist und nur bei den Responsive-Breakpoints umspringt. Ein solcher Container zentriert das Layout automatisch, indem er links und rechts Margins definiert.

```
<div class="container">
   <div class="row">
   ...
   </div>
</div>
```

Die Klasse „container-fluid“ hingegen beschreibt einen Container variabler Breite, der sich an die Dimensionen des Viewports anpasst. Ein „fluid“ Container hat also stets die volle Bildschirmbreite:

```
<div class="container-fluid">
   <div class="row">
   ...
```

```
   </div>
</div>
```

Innerhalb des Containers wird das Layout platziert, wobei ein **Zeilencontainer** der Klasse „row“ um die Spaltencontainer gelegt wird, die die Gridklassen zugewiesen bekommen.

In einer Row sollen sich (üblicherweise) die Summe der Gridzuweisungen auf **zwölf Einheiten** belaufen. Wird die Summe von 12 überschritten, so bricht die überschüssige Spalte in eine **Folgezeile** um, wo sie die ihrer Gridklassen entsprechende Breite einnimmt (Zeilen können so auch nur *unvollständig* gefüllt sein!).

Die Verwendung mehrerer expliziter Rows ermöglicht, **verschiedene Spaltenkonfigurationen** innerhalb des Layouts einer Geräteklasse einzusetzen. Bootstrap erlaubt in einem Layout beliebig viele Zeilen, die aber in ihrer Gesamtheit in einen Container platziert werden müssen.

✔ Bootstrap empfiehlt dabei, nicht mehrere, sondern tatsächlich **nur einen einzigen Container** einzusetzen.

```
<!DOCTYPE html>
<html>
<head>
  <title>Die Bootstrap Grid Klassen</title>
  <meta name="viewport"
     content="width=device-width, initialscale=1.0">
  <!-- Bootstrap -->
  <link href="css/bootstrap.min.css"
    rel="stylesheet" media="screen">
</head>
<body>
<div class="container">

  <h1> Hallo Bootstrap </h1>
  <p>Dieses Beispiel soll die Grid-Klassen zeigen.</p>

  <div class="row">

    <!-- Summe: 12 Einheiten für Medium Devices -->
    <div class="col-md-4">
      <h2>Rosen</h2>
      <p>Lorem ipsum dolor sit amet, ... </p>
    </div>
```

```
      <div class="col-md-4">
        <h2>Tulpen</h2>
        <p>Lorem ipsum dolor sit amet, ... </p>
      </div>
      <div class="col-md-4">
        <h2>Nelken</h2>
        <p>Lorem ipsum dolor sit amet, ...</p>
      </div>

    </div> <!-- Ende row -->

</div> <!-- Ende container -->
</body>
</html>
```

✔ Dieses Grid verwendet die *md*-Grid-Klassen für “medium devices”.

Das bedeutet, dass das Grid für Geräte dieser Viewportgröße **dreispaltig** erscheint, für *kleinere* Viewports aber auf **einspaltige** Ansicht umschaltet. Für größere Geräte bleibt es bei der dreispaltigen Ansicht.

Abb.: Die dreispaltige Ansicht des MD-Grids

Sobald der Viewport verkleinert wird (oder wenn das Dokument auf einem Gerät mit entsprechend kleinem Bildschirm dargestellt wird), springt das Layout auf **einspaltige** Darstellung um.

✔ Die Spaltencontainer werden auf 100% Breite gebracht, weshalb sie wieder in Quelltextreihenfolge untereinander zu liegen kommen.

Abb.: Die einspaltige Ansicht des MD-Grids

4.3. Ein responsives Grid-Layout

Platziert man Gridklassen für **mehrere Screengrößen** in einem Layout, so verhält sich das Raster im Viewport unterschiedlich je nach Geräteklasse die zum Betrachten verwendet wird.

Hier ist das Layout für mittlere Geräte (und größer) dreispaltig in einer Ausgabezeile und für kleinere Geräte zweizeilig, mit zwei Spalten in der ersten und einer Spalte in der zweiten Zeile.

```
<div class="container">

  <h1> Hallo Bootstrap </h1>
  <p>Dieses Beispiel soll die Grid-Klassen zeigen.</p>

  <div class="row">

    <!-- Summe md: 4 + 4 + 4 (einzeilig)
         Summe sm: 6 + 6 (1.Zeile) + 12 (2.Zeile) -->

    <div class="col-md-4 col-sm-6">
      <h2>Rosen</h2>
      <p>Lorem ipsum dolor sit amet, ...</p>
    </div>
    <div class="col-md-4 col-sm-6">
      <h2>Tulpen</h2>
      <p>Lorem ipsum dolor sit amet, ...</p>
    </div>
```

```
      <div class="col-md-4 col-sm-12">
        <h2>Nelken</h2>
        <p>Lorem ipsum dolor sit amet, ...</p>
      </div>
    </div> <!-- Ende row -->

</div> <!-- Ende container -->
```

Abb.: Das Responsive Grid im Modus “kleine Geräte”

Grid-Klassen:

```
.col-lg-* /*(1-12)*/    .col-md-* /*(1-12)*/
.col-md-* /*(1-12)*/    .col-xs-* /*(1-12)*/
```

4.4. Push/Pull – Einfluss auf die Spaltendarstellung

Mittels Push und Pull kann die Reihenfolge der Spalten in der Darstellung modifiziert werden.

```
<div class="row">
   <div class="col-xs-9 col-xs-push-3">
      <h1>Pushed</h1>
      <p>Im Quelltext zuerst.</p>
   </div>
   <div class="col-xs-3 col-xs-pull-9">
      <h1>Pulled</h1>
      <p>Im Quelltext als zweite.</p>
```

```
    </div>
</div>
```

Die erste Spalte wird drei Einheiten nach rechts geschoben, die zweite um neun Einheiten nach links gezogen. Damit tauschen sie die Plätze.

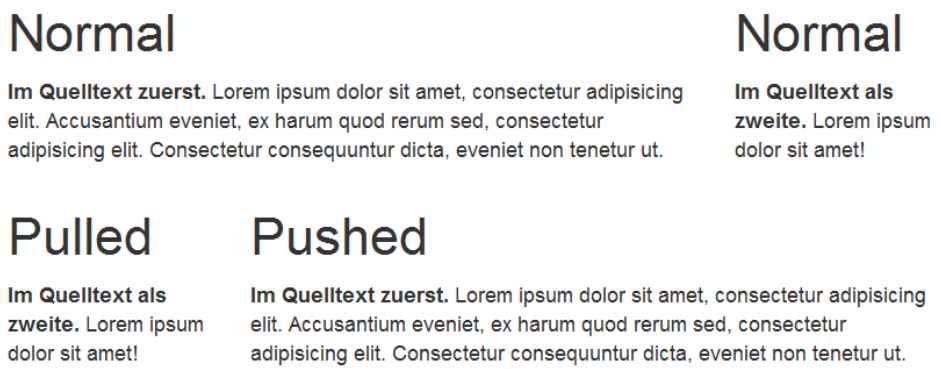


Abb.: „Push-Pull“. Obere Zeile normal, untere repositioniert.

✔ Die Vertauschung gilt, da die *xs*-Klasse für *kleine Geräte* gewählt wurde, für sämtliche Geräteklassen, da eine Vereinbarung stets ebenso für alle „großeren“ Geräte gilt.

Column-Push/Pull-Klassen:

```
.col-lg-pull-* /*(0-12)*/    .col-lg-push-* /*(0-12)*/
.col-md-pull-* /*(0-12)*/    .col-md-push-* /*(0-12)*/
.col-sm-pull-* /*(0-12)*/    .col-sm-push-* /*(0-12)*/
.col-xs-pull-* /*(0-12)*/    .col-xs-push-* /*(0-12)*/
```

4.5. Offsets - Lücken im Grid

Die Offset-Klassen, die ebenfalls nach Geräteklassen sortiert sind, ermöglichen es, zwischen den Spaltencontainern einer Zeile gezielt Lücken zu definieren.

Man kann pro Geräteklasse Offsets zwischen 0 und 12 Einheiten Breite definieren:

```
col-xs-offset-0, col-xs-offset-1 ... col-xs-offset-12

col-sm-offset-0, col-sm-offset-1 ... col-sm-offset-12
```

```
col-md-offset-0, col-md-offset-1 ... col-md-offset-12

col-lg-offset-0, col-lg-offset-1 ... col-lg-offset-12
```

✔ Die Offset-Klassen sind von 0 Einheiten Breite bis 12 Einheiten Breite mit wachsendem `margin-left` (von 0% bis 100%) definiert.

Hierbei trägt eine Offsetklasse gleichberechtigt zu den Spaltencontainern zur Anzahl der horizontalen Grideinheiten bei. Die Summe aus Spaltenbreiten und Offsetbreiten beträgt stets ebenfalls 12.

```
<div class="container">

   <div class="row"><!-- 12 Einheiten: -->
      <div class="col-md-4">
         ...
      </div>
      <div class="col-md-4 col-md-offset-4 " >
         ...
      </div>
   </div>

   <div class="row"><!-- 12 Einheiten: -->
      <div class="col-md-3 col-md-offset-3" >
         ...
      </div>
      <div class="col-md-3 col-md-offset-3" >
         ...
      </div>
   </div>
   <div class="row">
      <div class="col-md-6 col-md-offset-3 sp5" >
         .col-md-6 <b>.col-md-offset-3</b>
      </div>
   </div>
</div>
```

Hier ein Screenshot anhand der im Quelltext vorgestellten Gridkonfiguration für “medium” Ausgabegeräte.

Abb.: Grind mit Offset-Lücken

Column-Offset-Klassen:

```
.col-lg-offset-*   /*(0-12)*/
.col-md-offset-*   /*(0-12)*/
.col-sm-offset-*   /*(0-12)*/
.col-xs-offset-*   /*(0-12)*/
```

4.6. Zeigen und Verstecken im Grid

Responsive Visibiltity-Klassen:

```
.hidden       .hidden-lg       .hidden-md
              .hidden-sm       .hidden-xs

.visible-lg              .visible-lg-block
.visible-lg-inline       .visible-lg-inline-block
.visible-md              .visible-md-block
.visible-md-inline       .visible-md-inline-block
.visible-sm              .visible-sm-block
.visible-sm-inline       .visible-sm-inline-block
.visible-xs              .visible-xs-block
.visible-xs-inline       .visible-xs-inline-block

.hidden-print
.visible-print           .visible-print-block
.visible-print-inline    .visible-print-inline-block
```

5. Bootstrap-Komponenten

Bootstrap kennt neben den Grids auch CSS-Konfigurationen, die für Inhalte und deren Gestaltung zuständig sind, die sogenannten “Komponenten”. Wir betrachten zwei Untergruppen, die „Page-Komponenten“ für allgemeine

Inhalte und die „Navigations-Komponenten“, die speziell für die Gestaltung von Menükonfigurationen eingesetzt werden.

5.1. Page-Komponenten – Container und mehr

Zu den Page-Komponenten gehören einige „selbstverständliche“ Elemente, die von Bootstrap aber mit einem speziellen Styling versehen werden.

5.1.1. Container

Bei den Grids wurden sie bereits vorgestellt: Die **Container-Klassen** `container` und `container-fluid` können allerdings auch ohne den Einsatz eines Grids eingesetzt werden, bzw. dienen allgemein als **Layoutrahmen**.

✔ Sie zählen deshalb zu den Page-Komponenten.

```
<div class="container">
   <div class="row">

   <!-- dieses Layout ist zentriert
        und hat eine feste Breite pro Geräteklasse -->

   </div>
</div>
```

Die Klasse `container-fluid` hingegen beschreibt einen Container variabler Breite, der sich an die Dimensionen des Viewports anpasst. Ein „fluid“ Container hat also stets die volle Bildschirmbreite:

```
<div class="container-fluid">
   <div class="row">

   <!-- dieses Layout ist flexibel
        und hat volle Viewportbreite -->

   </div>
</div>
```

Container-Klassen:

```
.container        .container-fluid
```

5.1.2. Seiten-Header

Die Page-Header-Komponente ist im Prinzip nichts Besonderes. Der Seitenheader-Bereich wird jedoch durch ein erhöhtes Padding nach oben etwas deutlicher hervorgehoben.

```
<div class="page-header">
  <h1>Ein Page-Header
        <small>mit kleinerem Zusatztext</small>
  </h1>
</div>
```

Abb.: Der Page-Header-Container gibt der Überschrift mehr „Luft“

Header-Klassen:
`.page-header`

5.1.3. Well - Hervorhebung für Inhalte

Das Well („Senke“) ist eine **Hervorhebung** für Inhalte, die in diesem Bereich erscheinen. Das Well wird grau hinterlegt und besitzt ein Padding. Häufig wird es für die Darstellung von Quelltext verwendet.

```
<div class="well">...</div>
```

Abb.: Die drei Wells von Bootstrap: normal, large und small

✔ Mit den Zusatzklassen „well-lg“ und „well-sm“ wird das Padding geringfügig erhöht bzw. verringert:

```
<div class="well well-lg">...</div>
<div class="well well-sm">...</div>
```

Well-Klassen:
`.well` `.well-lg` `.well-sm`

5.1.4. Jumbotron

Das **Jumbotron** ist eine bootstrapeigene Präsentation, um Teaser möglichst prominent darzustellen. Die Überschrift wird hervorgehoben, der Inhalt, ähnlich wie beim *Well* grau hinterlegt.

Abb: Normales Jumbotron in einem Container

```
<div class="container">
   <div class="jumbotron">
      <h1>Ich bin ein Jumbotron</h1>
      <p>Lorem... </p>
      <p><a class="btn btn-primary btn-lg"
            href="#" role="button">
            Mehr erfahren?</a>
      </p>
   </div>
</div>
```

Ein Jumbotron kann auf volle Viewportbreite aufgezogen werden, wenn man den Container ins Innere nimmt. In diesem Fall hat es auch keine runden Ecken.

```
<div class="jumbotron">
  <div class="container">
    ...
  </div>
</div>
```

Abb.: "Full-width" Jumbotron mit Container im Inneren

Jumbotron-Klassen:
`.jumbotron`

5.2. Navigations-Komponenten

Beim Styling von **Navigationselementen** ist Bootstrap breit aufgestellt. Es existieren verschiedene vordefinierte Dasrtellungen für passive Navigationen, wie Tabs und Pills, aber auch komplexe, interaktive Komponenten wie Navbars, die Unterstützung durch JavaScript benötigen. Wir betrachten zunächst die etwas einfacher gestalteten Navigationen.

Klassen für Navigationen:

```
.nav          .nav-divider     .nav-justified
.nav-tabs     .nav-tabs-justified
.nav-pills    .nav-stacked
```

5.2.1. Nav-Tabs

Wie gewöhnlich ist das HTML-Gerüst hinter einem Navigationsmenü eine UL-Liste. Dies ist bei Bootstrap nicht anders. Ein als Navigation zu verwendender UL-Container erhält zunächst die Klasse **„nav“,** um ihn *allgemein* als „Navigation“ zu kennzeichnen.

```
<ul class="nav">
   ...
</ul>
```

Anschließend muss sich auch weiterhin nicht der Mühe eines Grundstylings unterziehen, um aus der nackten Liste eine Navigation eine Karteireiter-Metapher (Tabs) mit nebeneinanderliegenden Navigationselementen zu generieren. Das Zuweisen der **Zusatzklasse** „nav-tabs“ genügt.

```
<ul class="nav nav-tabs">
   <li role="presentation" class="active">
      <a href="#">Rosen</a>
   </li>
   <li role="presentation">
      <a href="#">Tulpen</a>
   </li>
   <li role="presentation">
      <a href="#">Nelken</a>
   </li>
</ul>
```

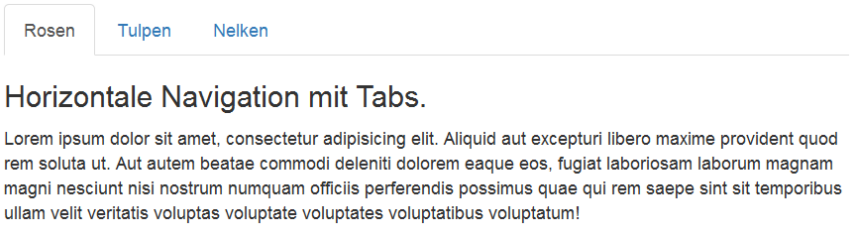


Abb.: Ein Karteireiter-Menü durch die Klasse “nav-tabs”

Eine Abwandlung der Tabs sind die “justified Tabs”, die eine Zusatzklasse „nav-justified“ erhalten. Die Tabs werden hierdurch über die gesamte Breite

der Layoutspalte gestreckt, für kleine Viewports liegen sie (Vorsicht!) mit voller Breite *übereinander*.

```
<ul class="nav nav-tabs nav-justified">
  ...
</ul>
```

Abb: "Justified" Tabs, hier mit 5 Menüpunkten

✔ Bei einer passenden Zahl von Elementen und einer geeigneten Zielgeräteklasse mag dieses Layout durchaus interessant sein.

5.2.2. Nav-Pills

Eien beliebte Präsentaionsform von Navigationsbuttons sind die von Bootstrap so genannten **"Pills"**, für die einzelne Navigationselemente in Buttonform mit abgerundeten Ecken und einem Abstand dazwischen nebeneinander dargestellt werden.

Wir gehen auch hier wieder von einer UL-Liste mit Klasse „nav" aus, die diesmal die Zusatzklasse „nav-pills" erhält. Die HTML-Struktur ist also dieselbe wie bei der Karteireiter-Navigation.

```
<ul class="nav nav-pills">
   <li role="presentation" class="active">
      <a href="#">Rosen</a>
   </li>
   <li role="presentation">
      <a href="#">Tulpen</a>
   </li>
   <li role="presentation">
      <a href="#">Nelken</a>
   </li>
</ul>
```

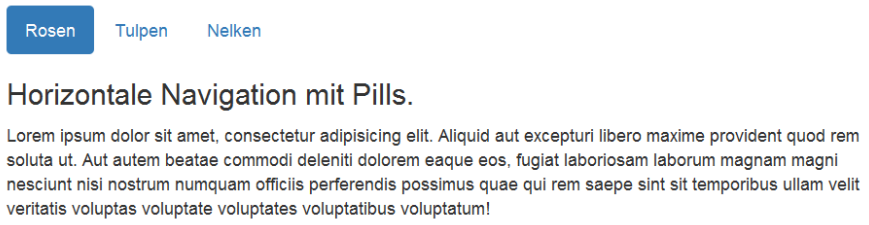


Abb.: Nebeneinanderliegende Pills formen eine Navigation

✔ Die „Pill“ mit Klasse „active“ erhält die Primärfarbe zugewiesen, die anderen sind transparent, hovern aber hellgrau.

Die Pills können alternativ auch **vertikal** („stacked“) angeordnet werden. Hierbei wachsen sie jedoch auf die Breite des umliegenden Container an:

```
<ul class="nav nav-pills nav-stacked">
  ...
</ul>
```

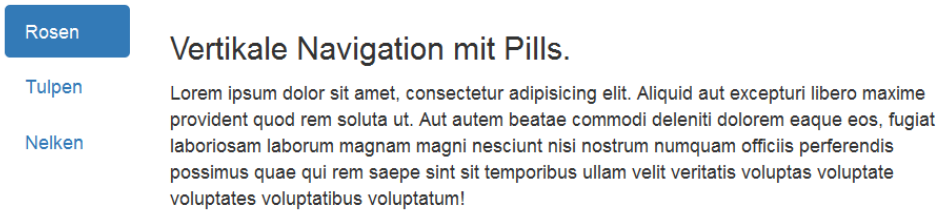


Abb. „Stacked“ Pills in einer zwei Einheiten breiten Layoutspalte

5.2.3. Nav-Bars

Für den Betrieb der **Nav-Bar** wird JavaScript (jQuery) benötigt. Neben den Bootstrap-Styles muss daher (in dieser Reihenfolge) jQuery und Bootstraps Plugin-Datei bootstrap.js verlinkt werden:

```
<link href="css/bootstrap.min.css" rel="stylesheet"
      media="screen">
<script src="js/jquery.js"></script>
<script src="js/bootstrap.js"></script>
```

Die eigentliche HTML-Struktur der Navbar ist etwas aufwendig. Wir beginnen mit einem Außencontainer. Ein `<div>`-Element tut es zwar, aber aus Gründern der semantischen Eindeutigkeit soll es diemal ein `<nav>`-Container sein, der das Element als „Hauptnavigation“ ausweist.

Witr brauchen zwei Klassen, nämlich „navbar“ und „navbar-default“. Letztere steuert das Layout. Es ist somit eine „normale“ Navbar:

```
<nav class="navbar navbar-default">
   ...
</nav>
```

Da eine Navbar üblicherweise die volle Bildschirmbreite haben soll, wird unmittelbar ein “fluid” Container eingefügt:

```
<nav class="navbar navbar-default">
  <div class="container-fluid">
      ...
   </div>
</nav>
```

Die eigentliche Navbar besteht aus zwei Strukturen. Die erste ist der Navbar-Header, der auch für das „Toggling“ zuständig ist, also das Expandieren einer in der Mobilansicht eingeklappten Navbar. Hierfür enthält der Header neben einem Logo („Brand“) auch einen Toggle-Button.

Die zweite Struktur ist der eigentliche Navbarcontainer, der bei Bedarf eingeklappt („collapsed“) wird. Er enthält die eigentliche Navigation in Form einer UL-Liste.

✔ Wichtig ist der **Identifier** an diesem Container, der als **Target** des Toggle-Buttons verwendet wird.

```
<nav class="navbar navbar-default">
  <div class="container-fluid">
    <!-- Logo und Toggle-Button gruppieren! -->
    <div class="navbar-header">
      ...
   </div>
    <!-- eigentliche Navbar, die getoggelt wird: -->
    <div class="collapse navbar-collapse"
         id="navbar-collapse">
      ...
    </div>
   </div><!-- /.container-fluid -->
</nav>
```

Betrachten wir nun zunächst die erste Struktur, nämlich den Navbar-Header:

```
<div class="navbar-header">
   <button type="button"
           class="navbar-toggle collapsed"
           data-toggle="collapse"
           data-target="#navbar-collapse">
      <span class="sr-only">Toggle Navigation</span>
      <span class="icon-bar"></span>
      <span class="icon-bar"></span>
      <span class="icon-bar"></span>
   </button>
   <a class="navbar-brand" href="#">Logo/Markenname</a>
</div>
```

Der Bereich enthält einen Button, der vier Spans enthält. Einer, mit Klasse „sr-only“, ist nur für **Screenreaderausgabe** gedacht, ist also normalerweise unsichtbar. Die anderen drei, mit der Klasse „icon-bar“, stellen die üblichen drei Querlinien eines Toggle-Buttons bereit.

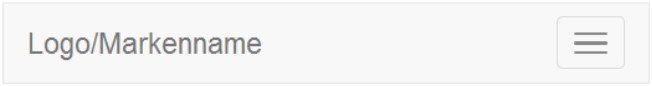


Abb.: Eingeklappte Navbar mit sichtbarem Toggle-Button

Dier Link mit der Klasse „navbar-brand“ ist dazu gedacht, ein Logo oder einen Text zu enthalten, der gleichzeitig als Home-Link dient. Die „Brand“ bleibt auch bei eingeklappter Navbar sichtbar. Hier kann anstelle eines Textes auch ein Image mit einer Grafik eingefügt werden:

```
<a class="navbar-brand" href="#">
   <img alt="Logo" src="das_logo.png">
</a>
```

Ein Klick auf den Botton klappt die Navbar nach unten aus, wobei die Links auf 100% Breite gebracht werden.

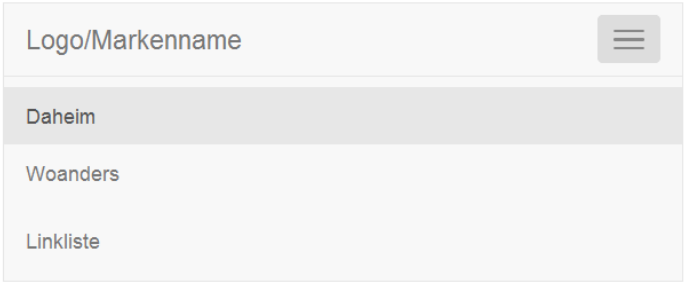


Abb.: Ausgeklappte Navbar. Ein erneuter Klick klappt sie wieder ein.

Von **funktionaler Bedeutung** sind die beiden `data`-Attribute, die der Button erhält.

- ✔ **data-toggle="collapse"**
  Steuert das Verhalten des Buttons. Dieser wird für große Screens versteckt, wenn die Navbar nicht reduziert werden muss. Für kleine Screens wird die Navbar versteckt und der Button gezeigt, der sie dann per Klick einblendet.

- ✔ **data-target="#navbar-collapse"**
  Das Target ist der innere Navbar-Container, der die eigentliche Navigation enthält, und der ein- und ausgeklappt werden soll. Das `data-target`-Attribut nennt den **ID des Navbar-Containers**, wobei, wie bei einem Fragment-Identifier das Hash # vorangestellt wird.

Die zweite Struktur ist die eigentliche Navbar. Sie enthält allermindestens eine UL-Liste, die die Hauptnavigation darstellt. Die Liste bekommt die Klassen „nav“ und „navbar-nav“.

```
<div class="collapse navbar-collapse"
     id="navbar-collapse">
   <ul class="nav navbar-nav">
      <li class="active">
         <a href="#">Daheim</a>
      </li>
      <li><a href="#">Woanders</a></li>
   </ul>
</div>
```

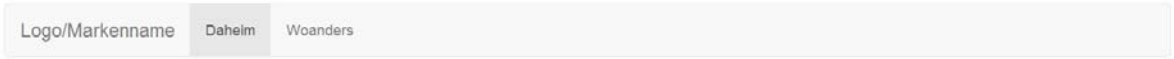


Abb.: Die einfache Navbar besteht nur aus einer Menüstruktur links

Interessant ist, dass eine zweite, rechtsbündige Menüstruktur hinzugefügt werden kann. Hierfür wird einfach eine zweite UL-Liste mit der Klasse „navbar-right“ dazugegeben:

```
<div class="collapse navbar-collapse"
     id="navbar-collapse">
   <ul class="nav navbar-nav">
      ...
   </ul>
   <ul class="nav navbar-nav navbar-right">
```

```
      <li><a href="#">Linkliste</a></li>
   </ul>
</div>
```

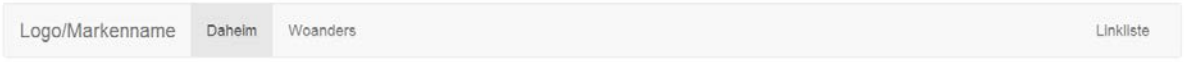


Abb.: Diese Navbar besitzt eine zweite, rechtsbündige Menüstruktur

Aus Platzgründen können Navigationselemente als Dropdownmenüs ralisiert werden. Diese öffnen sich durch Klick auf den Hauptmenüpunkt und bleiben dann geöffnet, bis der Fokus dort wieder entfernt wird.

Hier wird ein Dropdownmenü in der linken Navigation eingefügt:

```
<ul class="nav navbar-nav">
   <li class="active"><a href="#">Daheim</a></li>
   <li><a href="#">Woanders</a></li>
   <li class="dropdown">
      <a href="#"
         class="dropdown-toggle"
         data-toggle="dropdown"
         role="button" aria-haspopup="true"
         aria-expanded="false">
            Blumenliste <span class="caret"></span>
     </a>
      <ul class="dropdown-menu">
         <li><a href="#">Rosen</a></li>
         <li><a href="#">Tulpen</a></li>
         <li><a href="#">Nelken</a></li>
         <li role="separator" class="divider"></li>
         <li><a href="#">Veilchen</a></li>
         <li><a href="#">Orchideen</a></li>
      </ul>
   </li>
</ul>
```

Der Dropdownlink wird als normales Listitem in das Hauptmenü integriert, es erhält dabei aber die Klasse “dropdown”. Der Link, der das Verhalten triggert, bekommt ein `data-toggle`-Attribut „dropdown“. Neben dem Linktext sorgt ein Span mit der Klasse „caret“ für den Pfeil, der das Dropdown kennzeichnet.

Das eigentliche Dropdownmenü wird durch ein UL der Klasse „dropdown-menu“ gebildet. Als optische Unterteilungen können Listitems der Klasse „divider“ eingezogen, die eine Querlinie erzeugen.

Ein zweites Dropdownmenü kommt in die rechte Navigation. Die Struktur ist identisch.

```
<ul class="nav navbar-nav navbar-right">
   <li><a href="#">Linkliste</a></li>
   <li class="dropdown">
      <a href="#" class="dropdown-toggle"
         data-toggle="dropdown"
         role="button"
         aria-haspopup="true"
         aria-expanded="false">
             Themen <span class="caret"></span>
      </a>
      <ul class="dropdown-menu">
<li><a href="#">Ein Thema</a></li>
<li><a href="#">Noch ein Thema</a></li>
<li><a href="#">Drittes Thema</a></li>
<li role="separator" class="divider"></li>
<li><a href="#">Was anderes...</a></li>
      </ul>
   </li>
</ul>
```

Das Ergebnis ist in der folgenden Abbildung zu sehen.

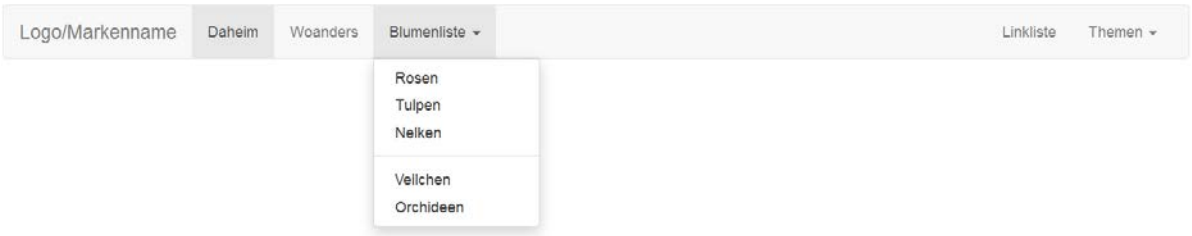


Abb.: Die Navbar hat links und rechts je ein Dropdown-Menü erhalten

Desweiteren kann die Navbar durch ein **Formular** erweitert werden, wobei es sich in der Regel um ein Suchformular handeln wird. Dieses wird einfach zwischen das linke und das rechte Menü (bzw. hinter das linke Menü) gesetzt.

✔ Zu beachten ist, dass das **Formular gefloatet** werden muss, um das Layout der Navbar nicht zu zerbrechen. Es benötigt daher die Klasse „navbar-left“ (nach links floaten).

```
<ul class="nav navbar-nav">
   ...
```

```
</ul>
<form class="navbar-form navbar-left">
   <div class="form-group">
      <input type="text" class="form-control"
             placeholder="Suche">
   </div>
   <button type="submit" class="btn btn-default">
      Suchen
   </button>
</form>
<ul class="nav navbar-nav navbar-right">
   ...
</ul>
```

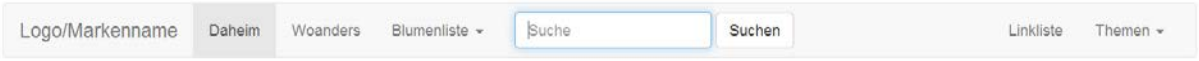


Abb.: Die Navbar wurde um ein Suchformular in der linken Spalte erweitert

Das Suchformular kann ebenso der *rechten* Navigation zugeschlagen werden. Da *alle* rechten Elemente gefloatet werden, ist die Reihenfolge von Bedeutung. Um das Formular in der Mitte zu belassen, muss es nach dem rechten Menü im Quelltext stehen.

✔ Das Formular erhält nun die Klasse **„navbar-right“**.

```
<ul class="nav navbar-nav">
   ...
</ul>
<ul class="nav navbar-nav navbar-right">
   ...
</ul>
<form class="navbar-form navbar-right">
   <div class="form-group">
      <input type="text" class="form-control"
             placeholder="Suche">
   </div>
   <button type="submit" class="btn btn-default">
      Suchen
   </button>
</form>
```

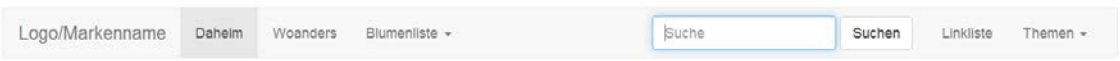


Abb.: Das Suchfeld wurde der rechten Seite der Navbar zugeschlagen

Eine sehr beliebte, dabei einfach zu erreichende, Ansicht erhält man, wenn man den Navbar-Container (außen) auf **Klasse „navbar-inverse“** anstelle von „navbar-default“ setzt. Die Farben der Navbar werden invertiert, der Balken wird schwarz. Die inneren Strukturen bleiben ansonsten identisch.

```
<nav class="navbar navbar-inverse">
   ...
</nav>
```

Abb.: Die Navbar in der Ansicht “navbar-inverse” erscheint schwarz

Es existieren noch **weitere Möglichkeiten** für Navbars. Diese können am oberen (mit `.navbar-fixed-top`) oder unteren (mit `.navbar-fixed-bottom`) Viewportrand fixiert werden, oder auch (mit `.navbar-static-top`) statisch am oberen Rand des Dokuments festgesetzt werden.

✔ **Mehr hierzu:** http://getbootstrap.com/components/#navbar

Navbar-Klassen:

```
.navbar            .navbar-brand    .navbar-btn
.navbar-collapse   .navbar-default  .navbar-fixed-bottom
.navbar-fixed-top  .navbar-form     .navbar-header
.navbar-inverse    .navbar-left     .navbar-link
.navbar-nav        .navbar-right    .navbar-static-top
.navbar-text       .navbar-toggle
```

5.2.4. Breadcrumbs

Neben der Hauptnavigation besteht ein weiterer funktionaler Bereich der Navigation in der **Breadcrumb-Navigation**. Sie dient dazu, dem User in hierarchischen Websites Auskunft über seine *Position* in der Struktur zu geben („You are here“-Indicator) und gleichzeitig zu erlauben, in die darüberliegenden Hierarchiestufen (Bereche) zurückzuspringen.

In der Regel setzt die Breadcrumb mit dem *Homelink* an. Der zweite Link kennzeichnet den *aktuellen Bereich*, dann geht es tiefer in dessen *Unterbereiche* bis zur *aktuellen Seite*. Deren Verweis kann ebenfalls (muss es aber nicht zwingend) als Link ausgestattet sein. Im vorliegenden Beispiel ist dies nicht der Fall:

```
<ol class="breadcrumb">
   <li><a href="#">Home</a></li>
   <li><a href="#">Blumendatenbank</a></li>
   <li class="active">Rosen</li>
</ol>
```

Strukturell wird der Breadcrumb eine OL-Liste hinterlegt, in dem Sinne, dass eine hierarchische *Reihenfolge* dargestellt wird.

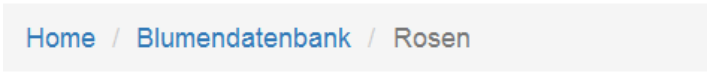


Abb.: Eine Breadcrumb-Navigation

Breadcrumb-Klassen:
`.breadcrumb`

5.3. Komponenten für Inhalte

Auf der Inhaltsebene stellt Bootstrap sogenannte “Panels” zur Darstellung von Inhalten zur Verfügung, sowie eine große Zahl nützlicher Klassen, um alle Arten von Buttons zu gestalten oder zu gruppieren.

5.3.1. Panels

Ein Panel ist im Prinzip ein Layoutelement, das über einen Header, einen Body und einen Footer verfügt. Alles kann beliebig gestaltet werden. Der äußere Container erhält die Klasse „panel“ und zusätzlich eine weitere Klasse, welche die Optik des Panels festlegt. Hier ist es die Defaultdarstellung „panel-default“.

```
<div class="panel panel-default">
   <div class="panel-heading">
      <span class="text-uppercase">
         Geschäftsbedingungen
      </span>
   </div>
```

```
    <div class="panel-body">
       Lorem ipsum dolor sit amet, ...
    </div>
    <div class="panel-footer">
       <a href="#" class="btn btn-danger btn-sm">
          Ich stimme zu
       </a>
       <a href="#" class="btn btn-default btn-sm">
          Ich lehne ab
       </a>
</div>
</div>
```

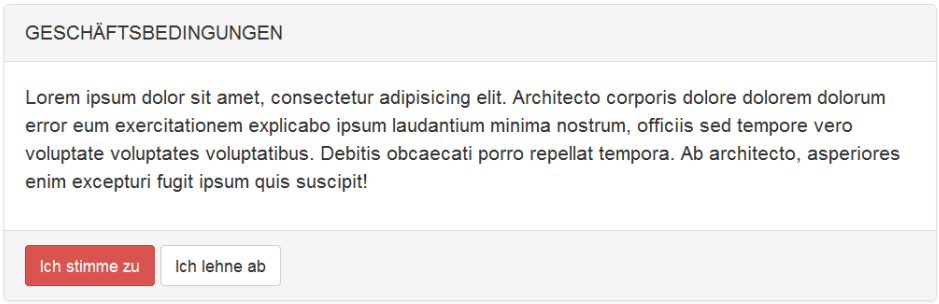


Abb.: Ein Panel-Element

Alternative Darstellungen sind durch Beigabe anderer Darstellungsklassen anstelle von “panel-default” möglich.

Zur Auswahl stehen:

- panel-primary: Dunkelblau
- panel-success: Grün
- panel-info: Hellblau
- panel-warning: Gelb
- panel-danger: Rot

Hier ist ein Panel in der Info-Konfiguration:

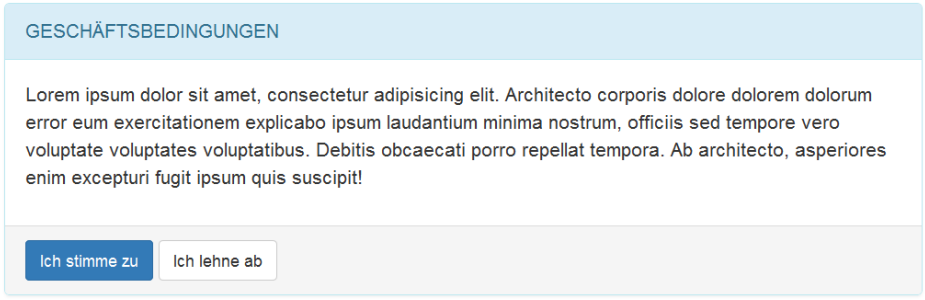


Abb.: Panel als „panel-info“

Panel-Klassen:

`.panel`	`.panel-body`	`.panel-danger`
`.panel-default`	`.panel-footer`	`.panel-group`
`.panel-heading`	`.panel-info`	`.panel-primary`
`.panel-success`	`.panel-title`	`.panel-warning`

5.3.2. Buttons

Bootstrap gibt sich viel Mühe, um gestylte Buttons zur verfügung zu stellen. Grundlage für einen Button können einfache Links, aber auch „echte“ Buttons in Form von Formular-Inputs sein.

✔ Ein Button benötigt die Klasse „btn“ und mindestens eine Zusatzklasse wie „btn-default“ zur Steuerung des Aussehens.

```
<a href="#" class="btn btn-primary">Hier klicken</a>
```

Es gibt Zusatzklassen, um verschiedene **Buttonfarben** zu erzeugen:

- btn-default
  mit weißem Hintergrund
- btn-primary
  mit blauem Hintergrund
- btn-success
  mit grünem Hintergrund
- btn-info
  mit hellblauem Hintergrund

- btn-warning
  mit orangenem Hintergrund

- btn-danger
  mit rotem Hintergrund

Jedem Button kann zusätzlich eine Klasse gegeben werden, die seine **Größe** (Schriftgröße, Padding) steuert:

- btn-lg
  große Buttons

- btn-sm
  kleine Buttons

- btn-xs
  besonders kleine Buttons

Hier ein paar Beispiel im Quelltext:

```
<button type="button" class="btn btn-default btn-lg">
   Großer Default-Button
</button>

<button type="button" class="btn btn-warning">
   Default Button "Warnung"
</button>

<button type="button" class="btn btn-info btn-sm">
   Kleiner Button "Info"
</button>

<button type="button" class="btn btn-danger btn-xs">
   Ganz kleiner Button "Danger"
</button>
```

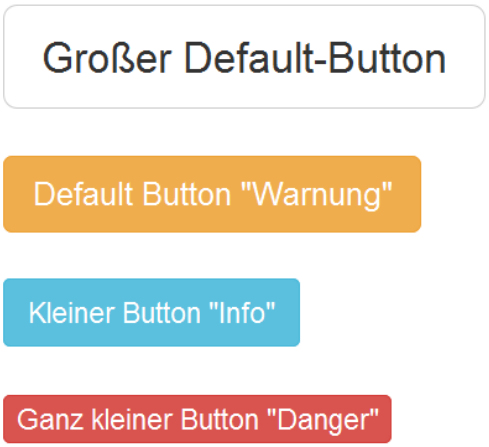


Abb.: Buttons mit unterschiedlichen Größen und Farben

Button-Klassen:

.btn	.btn-block	.btn-danger
.btn-default	.btn-group	.btn-group-justified
.btn-info.	.btn-link	.btn-group-vertical
.btn-primary	.btn-sm	.btn-success
.btn-toolbar	.btn-warning	.btn-xs

5.4. Formulare

Das Styling von Formularen mit Bootstrap geht schnell, da eine Optik weitestgehend vordefiniert ist. Es sind lediglich ein paar Helferklassen und Container erforderlich, allerdinsg nicht über das sonst übliche Maß hinaus.

✔ Die native Validierung von HTML5 beißt sich nicht mit Bootstrap.

Ein Formular benötigt die Basisklasse „form“.

```
<form class="form">
   ...
</form>
```

Um jeweils ein Inputfeld und sein Label (Beschriftung) zu gruppieren, werden diese zusammen in einen Container „form-group“ gesetzt. Jedes Inputelement erhält darüberhinaus die Klassen „form-control“.

```
<form class="form">
   <div class="form-group">
```

```
      <label for="nn">Name</label>
      <input type="text"
             class="form-control"
             name="nn"
             id="nn" placeholder="Ihr Name">
   </div>
</form>
```

5.4.1. Defaultansicht für Formulare

Etwas komplexer soll es schon werden. Hier gesellen sich weitere Inputs hinzu.

```
<form class="form">
   <div class="form-group">
      <label for="nn">Name</label>
      <input type="text"
             class="form-control"
             name="nn" id="nn"
             placeholder="Ihr Name">
   </div>
   <div class="form-group">
      <label for="email">Email</label>
      <input type="email"
             class="form-control"
             name="email" id="email"
             placeholder="Ihre E-Mailadresse">
   </div>
   <div class="form-group">
      <label for="fon">Telefon</label>
      <input type="text" class="form-control"
             name="fon" id="fon"
             placeholder="Ihre Telefonnummer">
   </div>
   <div class="form-group">
      <label for="kmt">Kommentar</label>
      <textarea class="form-control"
                name="kmt" id="kmt"></textarea>
   </div>
   <button type="submit" class="btn btn-primary">
      Abschicken
   </button>
   <button type="reset" class="btn btn-default">
```

```
      Zurücksetzen
   </button>
</form>
```

Abb.: Defaultformatierung für Formulare

5.4.2. Horizontales Formular

Das Formular eignet sich, wie eben gezeigt, in der Defaultdarstellung *eher nicht* für Desktopanwendungen, es sei denn, man platziert es in einer entsprechend schmalen Gridspalte.

Spontan besser sieht das sogenannte „horizontale“ Formular aus, bei dem Label und Input in ihrem Container-Div *nebeneinander* stehen.

```
<form class="form-horizontal">
   ...
</form>
```

✔ **Ärgerlich:** Das restliche HTML muss leider **angepasst** werden, da die *Breite der Label* festgelegt werden muss.

Dies geschieht durch Einbeziehen von **Gridklassen**, die an die Labels gelegt werden, sowie an einen **Hilfscontainer**, der nun zusätzlich den Input innerhalb des *„form-group“*-Containers umgeben muss.

Hier exemplarisch für einen der Inputs.

```
...
<div class="form-group">
```

```
   <label class="col-sm-2 control-label"
          for="nn">
       Name
   </label>
   <div class="col-sm-10">
      <input type="text"
             class="form-control"
             name="nn" id="nn"
             placeholder="Ihr Name">
   </div>
</div>
...
```

Da die Buttons kein Label besitzen, aus Gründen der Optik allerdings bündig unter den Inputs stehen sollen, erhalten sie ein *Wrapper-Div*, das zusätzlich eine *Push*-Klasse bekommt:

```
<div class="col-sm-10 col-sm-push-2">
   <button type="submit" class="btn btn-primary">
      Abschicken
   </button>
   <button type="reset" class="btn btn-default">
      Zurücksetzen
   </button>
</div>
```

Die Optik hat sich sichtlich geändert.

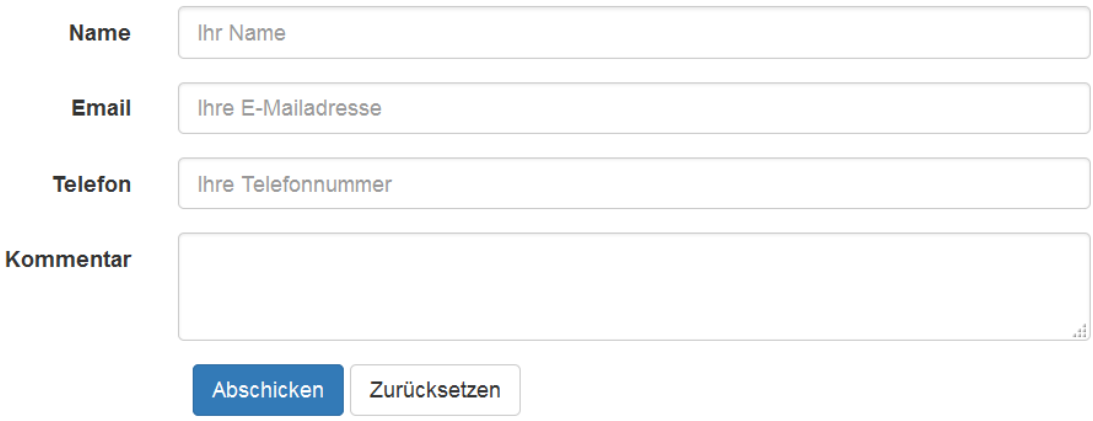


Abb.: Das horizontale Formulare mit zusätzlichen Gridklassen

5.4.3. Inline Formular

Noch einen Schritt weiter geht das “inline” Formular, bei dem sowohl Labels als auch Inputs auf die schmalst mögliche Breite gebracht werden, sodass sie theoretisch in einer Zeile stehen können.

✔ Für Formulare mit einer überschaubaren Zahl von Inputs mag dies angebracht sein.

```
<form class="form-inline">
    <div class="form-group">
      <label for="nn">Name</label>
      <input type="text"
             class="form-control"
             name="nn" id="nn"
             placeholder="Ihr Name">
    </div>
    <div class="form-group">
      <label for="email">Email</label>
      <input type="email"
             class="form-control"
             name="email" id="email"
             placeholder="Ihre E-Mailadresse">
    </div>
    <button type="submit" class="btn btn-primary">
      Abschicken
    </button>
    <button type="reset" class="btn btn-default">
      Zurücksetzen
    </button>
</form>
```

Abb.: Das Inline-Formular

Formular-Klassen:

```
.form-inline        .form-control
.form-horizontal    .form-control-feedback
.form-group         .form-control-static
.input-group        .input-group-addon
.input-group-btn    .input-lg             .input-sm
```

6. Bootstrap-Plugins

Die Programmierung der Plugins, die in Folge vorgestellt werden befindet sich insgesamt in der Datei *bootstrap.js* (bzw. *bootstrap.min.js*), die Teil des Bootstrap-Downloadpakets ist.

✔ Alle **Bootstrap-Plugins** benötigen zusätzlich das **jQuery-Framework**.

✔ **Achtung:** die aktuelle jQuery-Version 3 ist *nicht* auf Bootstrap 3 abgestimmt. Verwenden Sie stattdessen **jQuery 2.x**.

6.1. Dropdowns

Ein Dropdown besteht aus einem Link und einer UL-Liste. Die Liste ist per Default verborgen und wird durch Klick auf den Link (der auch ein Button sein kann) eingeblendet.

```
<div class="dropdown">
  <button id="dLabel" type="button"
          data-toggle="dropdown"
          data-target="#">
    Dropdown auf/zu
    <span class="caret"></span>
  </button>
  <ul class="dropdown-menu">
    <li><a href="#">Link 1</a></li>
    <li><a href="#">Link 2</a></li>
    <li><a href="#">Link 3</a></li>
    <li><a href="#">Link 4</a></li> ...
  </ul>
</div>
```

Die Optik sieht aus wie in der folgenden Abbildung.

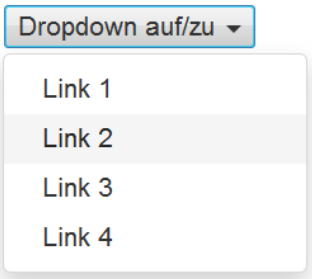


Abb.: Einfaches Dropdown mit Button

Möchte man nicht die Klasse „caret“ für den Dropdown-Pfeil verwenden, so kann alternativ ein Glyphicon wie das Chevron-Symbol eingesetzt werden.

```
<span class="glyphicon glyphicon-chevron-down"></span>
```

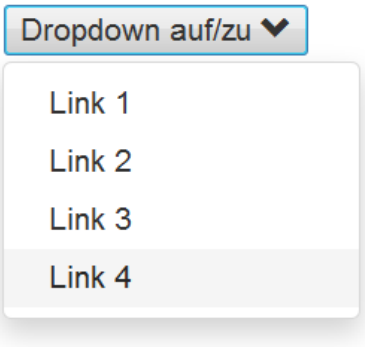


Abb.: Dropdown mit Chevron-Pfeil (Glyphicon)

Dropdown-Klassen:

```
.dropdown
.dropdown-toggle          .dropdown-backdrop
.dropdown-header          .dropdown-menu
.dropdown-menu-left       .dropdown-menu-right
```

6.2. Collapse-Widget

Ein Collpse-Widget dient dazu, einen Inhalt bei Bedarf auf- oder zuzuklappen. Es kann als unabhängiges Layoutelement eingesetzt werden.

✔ Der steuernde Button kann wahlweise als Link oder als Button-Element realisiert werden.

Ein Link zielt mit seinem `href`-Attribut auf den ID des Inhaltscontainers, ein Button benötigt hierfür ein `data-target`-Attribut. Unabhängig davon muss am Trigger das Attribut `data-toggle="collapse"` gesetzt sein.

Der Inhaltbereich erhält die Klasse „collapse“ und ein id-Attribut.

```
<a class="btn btn-primary"
   role="button"
   data-toggle="collapse"
   href="#collapser">
      Auf/Zu mit Link
</a>
<button class="btn btn-primary"
   type="button"
   data-toggle="collapse"
   data-target="#collapser">
      Auf/Zu mit Button
</button>
<div class="collapse" id="collapser">
  <div class="well">
    ...
  </div>
</div>
```

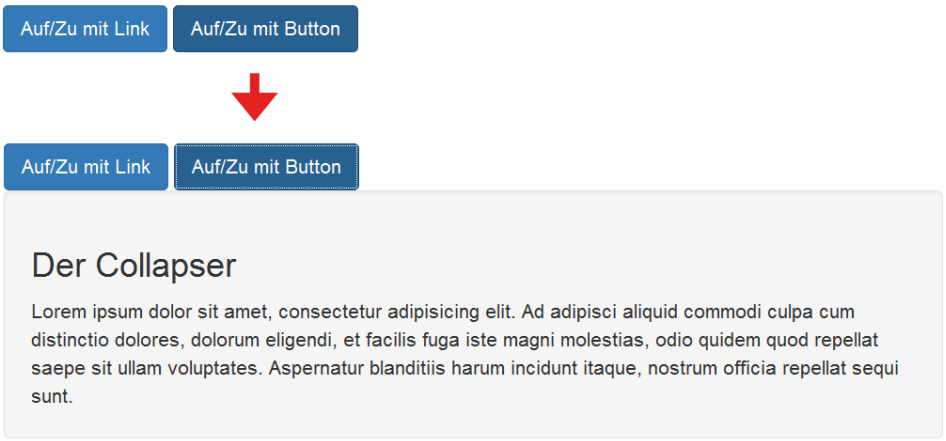


Abb: Ein Collapse-Widget

Collapse-Klassen:
`.collapse` `.collapsing`

6.3. Accordion mit Collapse-Plugin

Ein beliebtes Gestaltungelement, sobald auf einem begrenzten Raum viele Inhalte alternativ zu zeigen sind, ist das Accordion. Hier kann Bootstrap natürlich nicht auf eine adäquate Lösung verzichten. Hier das HTML-Gerüst eines typischen Accordions.

```
<div class="panel-group" id="accordion" role="tablist">
  <div class="panel panel-default">
    <div class="panel-heading" role="tab" id="phead1">
      <h4 class="panel-title">
        <a role="button" href="#pbody1"
           data-toggle="collapse"
           data-parent="#accordion">
          Panel 1
        </a>
      </h4>
    </div>
    <div id="pbody1" role="tabpanel"
        class="panel-collapse collapse in">
      <div class="panel-body">
        Erstes Panel. Lorem ...
      </div>
    </div>
  </div>
  <div class="panel panel-default"> ... </div>
  <div class="panel panel-default"> ... </div>
</div>
```

Das HTML ist etwas komplex.

✔ Ein Accordion besteht aus mehreren Panels der Klasse „panel“, die vom Accordion-Container zu einer „panel-group“ zusammengefasst werden.

✔ Im Headcontainer des Panels befindet sich ein Button (hier als Link in der Überschrift) mit einem Attribut `data-toggle="collapse"` und einem Attribut `data-parent="#accordion"` der auf den ID des Accordion-Containers zeigt.

✔ Das `href`-Attribut des Links zeigt auf den ID des eigentlichen Accordion-Panels, der jeweils zu öffnen ist.

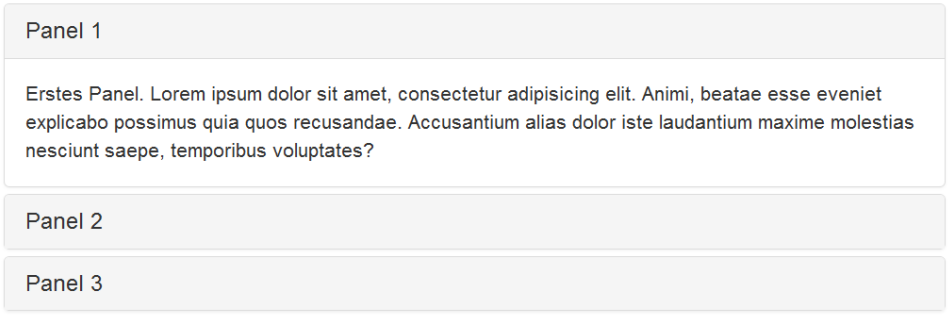


Abb.: Akkordion mit drei Panels

6.4. Tooltip-Widget

An einem Element kann ein Tooltip gezeigt werden, der wahlweise in vier verschiedenen Richtungen (*top*, *right*, *bottom*, *left*) eingeblendet wird. In diesem Beispiel werden Buttons verwendet.

✔ Unabhängig vom Element-Typ werden Tooltips beim **Hover** über dem Element eingeblendet.

Es muss sowohl ein Attribut `data-toggle="tooltip"` gesetzt werden, wie ein Attribut `data-placement`, das die Richtung des Tooltips festlegt. Der Text des Tootips wird dem `title`-Attribut des Elements entnommen.

```
<button type="button"
   class="btn btn-default"
   data-toggle="tooltip"
   data-placement="left"
   title="Tooltip links">
      Tooltip links
</button>

<button type="button"
   class="btn btn-default"
   data-toggle="tooltip"
   data-placement="top"
   title="Tooltip oben">
      Tooltip oben
</button>

<button type="button"
   class="btn btn-default"
   data-toggle="tooltip"
```

```
   data-placement="bottom"
   title="Tooltip unten">
      Tooltip unten
</button>

<button type="button"
   class="btn btn-default"
   data-toggle="tooltip"
   data-placement="right"
   title="Tooltip rechts">
      Tooltip rechts
</button>
```

Aus Günden der Performance sind Tooltips und Popovers nicht grundsätzlich aktiv, sondern müssen ausdrücklich initialisiert werden.

✔ Dies geschieht, indem das Bootstrap-Plugin **„Tooltip“** auf einer geeigneten Selektion von Elementen aufgerufen wird. Die Selektion findet mit Hilfe von jQuery statt (die Methode `tooltip()` von Bootstrap ist also eigentlich ein jQuey-Plugin).

Hier wird das Attribut `data-toggle="tooltip"`, das ein Tooltip kennzeichnet, als Selektionskriterium verwendet:

```
$(function() {
   $('[data-toggle="tooltip"]').tooltip();
});
```

Die funktionierenden Tooltips sehen wie folgt aus:

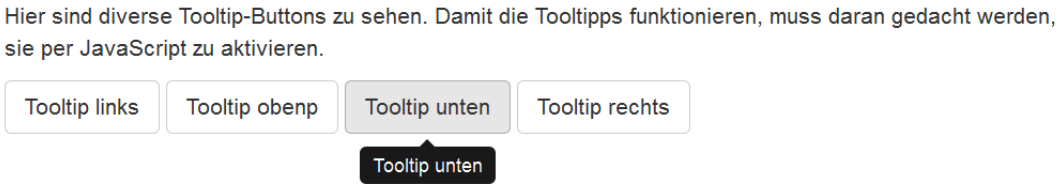


Abb.: Tooltips erscheinen bei Hover über dem Element

7. Bootstrap anpassen mit LESS

Die CSS-Sourcen hinter den Dateien von Bootstrap 3 sind in LESS geschrieben. Dies ermöglicht, den CSS-Code beliebig anzupassen. Der erste Einstiegspunkt besteht in den von Bootstrap zu diesem Zweck angebotenen Konfigurationsvariablen.

Alle LESS-Sourcen befinden sich im Verzeichnis `less/` in der Bootstrap-Distribution. Pro Komponente existiert in der Regel eine dedizierte LESS-Datei. Eine Grundkonfiguration erfolgt über die Datei ***variables.less***.

Der Ordner `mixins/` enthält funktionale Module, die in verschiedenen Bereichen des Frameworks eingebunden sind.

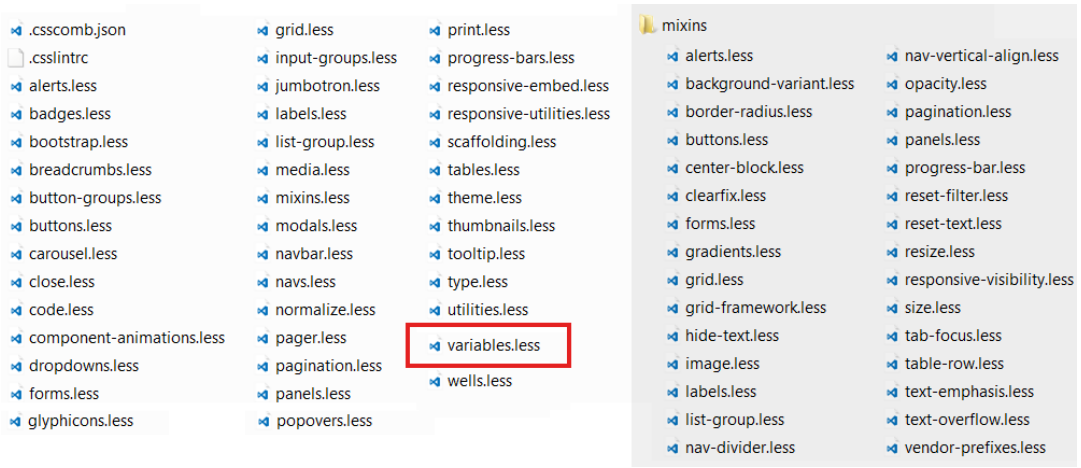


Abb.: Ein Blick in das Verzeichnis less/ der Bootstrap-Distribution

7.1. Variablen und Mixins in Bootstrap

Die Konfigurationsvariablen für diverse Grundeinstellungen von Bootstrap befinden sich in der Datei `variables.less`. Sie ist unterteilt in verschiedene Bereiche:

Colors

Hier werden die Basisfarben von Bootstrap gesetzt. Es handelt sich um sechs Grautöne und die fünf Signalfarben „primary“, „success“, „info“, „warning“ und „danger“. Diese werden überall in Bootstrap genutzt.

Scaffolding

Im Prinzip sind dies ebenfalls Farbdefinitionen, die jedoch funktionaler Natur sind, wie die Body-Hintergrundfarbe, die Text- und Linkfarben sowie entsprechende Hoverfarben.

Typography

Hier regelt man Font-Eigenschaften, wie Zeilenhöhe, Fontfamilien für diverse Einsatzzwecke, sowie die Schriftgrößen für Überschriften und deren dazu gehörigen Zeilenhöhen.

Iconography

Interessant, sofern man andere Icons als die gewöhnlichen Glyphicons einbinden möchte.

Components

Dieser Abschnitt regelt die Geometrie von Komponenten, wie Paddings und Bordereigenschaften (bezogen auf die Schriftgröße), sowie einige Farbwerte, wie die „active color“.

Tables

Hier sind Einstellungen, die Tabellendarstellung betreffen, wie ein Defaultpadding, Hintergrund-, Aktiv-, Border- und Hoverfarben

Buttons

Nach Einsatzgebiet getrennt (Success, Info etc.) funden sich hier Farbdefinitionen und weitere Einstellungen für die Bootstrap-Buttons.

Formulare

Hier regelt man Hintergrundfarben, Eckradius, Margins, Höhe und Breite, sowie weitere Eigenschaften von Inputs und anderen Formularkomponenten wie den Form-Groups und Form-Controls.

Dropdown

Alle Eigenschaften von Dropdownmenüs, wie Farben, Hoverfarben, Linkfarben, Border und Trenner.

Z-Index Master List

Alle von Bootstrap allgemein für Komponenten vergebenen z-index-Werte werden von hier gesteuert (Defaultwerte: 1000).

Media Queries Breakpoints

Die Breakpoints für `XS` (Phone, 480px), `SM` (Tablet, 768px), `MD` (Medium Screen, 992px) und `LG` (Large Screen, 1200px), wie Bootstrap sie setzt, können hier geändert werden.

Grid System

Hier kann die Anzahl der Gridspalten, deren Zwischenräume (*Gutter*) und Breakpoints festgelegt werden.

Container Sizes

Diese Rublik betrifft dei Abmessungen von Containern für verschiedene Ausgabegrößen (`container-tablet, container-desktop` etc.).

Formulare

In der Rubrik „Form states and alerts“ lassen sich die Success-, Warning-, Info-Zusandsfarben festlegen.

Navbar

Geometrie (Höhe, Margins und Padding) der Navbar, sowie Hintergrundfarben werden von hier gesteuert. Zusätzlich befinden sich hier alle Farbeinstellungen wie für Links, Hovers und Toggles sowie für die Grundfarbe (Brand-Color).

Inverted Navbar

Wie Navbar, aber für deren „inverted“ Variante.

Navs

Styles, die allgemein in Navigationen (Links, Tabs, Pills) eingesetzt werden.

Tabs

Styles für das Tab-Widget. Geregelt werden u.a. die Hintergrundfarbe der Tabs oder deren Link- und Hoverfarben

Pills

Styles für Nav-Pills. Im Wesentlichen deren Borderradius und Farben.

Weitere Widget-Styles

Für weitere Widgets, wie Jumbotron, modale Boxen, Labels, Popovers, Tooltips, Pager und Pagination, Hervorhebungen (Alerts), Progress bars, Listen, Panels, Thumbnails, Wells, Badges, Breadcrums und Carousels existieren ebenfalls Variablen, die die Optik steuern.

7.1.1. Farben (Grauskala):

Hier eine Übersicht über die Variablen der Grauskala:

```
@gray-darker:  lighten(#000, 13.5%); // #222

@gray-dark:    lighten(#000, 20%);   // #333

@gray:         lighten(#000, 33.5%); // #555

@gray-light:   lighten(#000, 46.7%); // #777

@gray-lighter: lighten(#000, 93.5%); // #eee
```

7.1.2. Farben (Signalfarben):

Den Signalfarben sind in Bootstrap per Default folgende Farbwerte zugeordnet:

```
@brand-primary: darken(#428bca, 6.5%); // #337ab7

@brand-success: #5cb85c;

@brand-info:    #5bc0de;

@brand-warning: #f0ad4e;

@brand-danger:  #d9534f;
```

7.1.3. Schrift und Typographie

Ein Gruppe weiterer Variablen steuert die Schriftart und -größe.

```
@font-family-sans-serif:
 "Helvetica Neue", Helvetica, Arial, sans-serif;

@font-family-serif:
 Georgia, "Times New Roman", Times, serif;

@font-family-monospace:
 Menlo, Monaco, Consolas, "Courier New", monospace;

@font-family-base:        @font-family-sans-serif;

@font-size-base:          14px;

@font-size-large:
 ceil((@font-size-base * 1.25)); // ~18px

@font-size-small:
 ceil((@font-size-base * 0.85)); // ~12px
```

Für Überschriften existiert eine weitere Gruppe:

```
@font-size-h1:
 floor((@font-size-base * 2.6)); // ~36px

@font-size-h2:
 floor((@font-size-base * 2.15)); // ~30px

@font-size-h3:
 ceil((@font-size-base * 1.7)); // ~24px

@font-size-h4:
ceil((@font-size-base * 1.25)); // ~18px

@font-size-h5:            @font-size-base;

@font-size-h6:
 ceil((@font-size-base * 0.85)); // ~12px
```

Dito für die Zeilenhöhe:

```
@line-height-base:        1.428571429; // 20/14
```

```
@line-height-computed:
 floor((@font-size-base * @line-height-base)); // ~20px
```

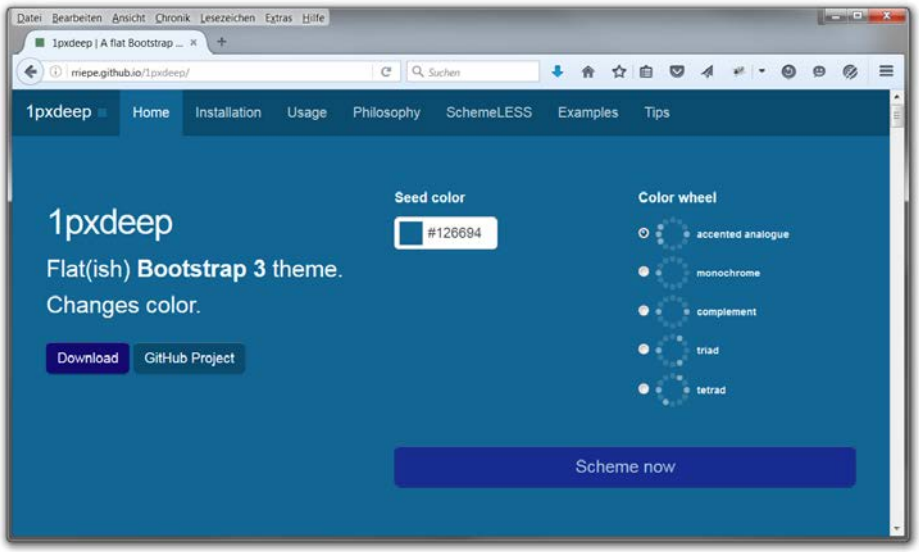
Allgemeine Einstellungen für Überschriften:

```
@headings-font-family:    inherit;

@headings-font-weight:    500;

@headings-line-height:    1.1;

@headings-color:          inherit;
```

7.1.4. Theming von Bootstrap mit LESS

Das **1pxdeep-Schema** ermöglicht sehr einfach eine Modifikation der Bootstrap-Sourcen über eine LESS-Datei, die mittels des Online-Interfaces erzeugt wird. Hierfür wird zunächst eine *Basisfarbe* festgelegt, die sogenannte **Seed-Color**.

Von dieser abgeleitet wird ein passendes **Farbschema**, das mittels der **Color-Wheel** Buttons entweder als kontrastierendes Farbmodell, als monochromes Modell, komplementäres (zwei Farben), triadisches (drei Farben) oder tetradisches System (vier Farben) angelegt werden kann.



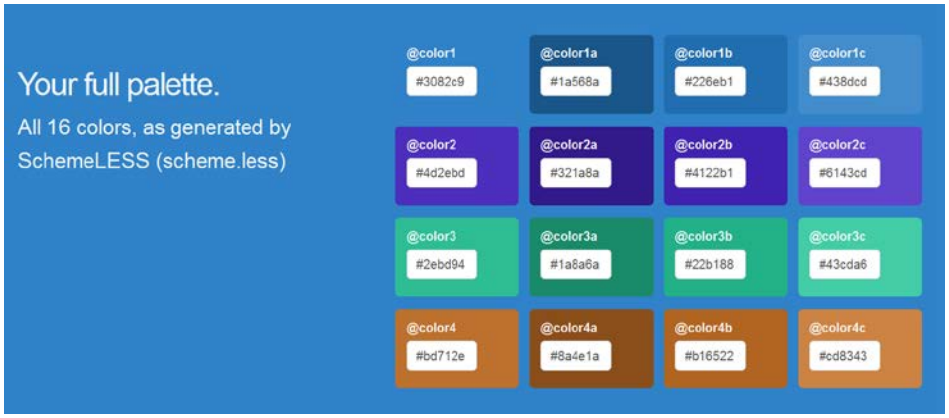
Anschließend braucht das so generierte Schema lediglich downgeloaded zu werden (Download-Button; man erhält eine Zip-Datei).

Die anschließende **Verwendung** erfolgt mit dem LESS-Compiler, indem eine *index.less* erstellt wird, die etwa folgenden Aufbau haben kann:

```
@import "scheme.less"; // generiertes Farbschema
@import "bootstrap.less"; // Bootstrap-Sources
@import "1pxdeep.less"; // Das 1pxdeep Theme
@import "style.less"; // ggfs. eigene Styles
```

Die beiden oben hervorgehobenen Dateien sind im Download enthalten. Hierbei enthält *scheme.less* die gewählte Seed-Farbe und die Funktionen, die von dieser die anderen Farben ableiten.

Das Schema enthält sechzehn Farbvariationen, die über Zusatzklassen auch direkt einer Komponente zugewiesen werden können.



Die zweite Datei, *1pxdeep.less*, muss nach *bootstrap.less* in den Compile-Vorgang eingebunden werden, da hier die Überschreibung der Bootstrap-Variablen stattfindet.

✔ **Siehe:** http://rriepe.github.io/1pxdeep/

8. Literatur

Bootstrap 3 - Stile und Komponenten
Jörg Krause
CreateSpace, 2015

Step By Step Bootstrap 3
Riwanto Megosinarso
CreateSpace, 2014

Bootstrap Essentials
Snig Bhaumik
Packt, 2015

Bootstrap
Jake Spurlock
O'Reilly, 2013

Learning Less.js
Alex Libby
Packt, 2014

Less Web Development Essentials (2nd Ed.)
Bass Jobsen
Packt, 2015