mozaikoo 3.5 Design Guides

Einleitung

Diese Dokumentation richtet sich an Benutzer, die Arbeiten am Template einer vorhandenen Frontend-Applikation in Ember.js* (inklusive API und Backend) vornehmen möchten.

Dazu gehören:

- die Konfigurationsmöglichkeiten und Struktur der Templates
- der Templateaufbau
- die Einstellungen für das Template im Backend, die aus den Konfigurationsdateien des Templates gelesen werden
- und das Design der Seiten und Elemente.

Die JavaScript-WebApp von mozaikoo verwendet (Ausnahmen) zum Erstellen der Templates die Glimmer-VM mit einer Handlebars-Syntax, die sich htmlbars nennt.

Für Ember-WebApps wurden die Handlebars- und ember.js-Tags um Helper und Komponenten erweitert. Dazu gehören Formatierungs-Hilfmittel, Textbehandlungshilfen, Logik-Operatoren, Variablen-Hilfen und Hilfmittel zum Errechnen von Werten.

Referenzen für den Einstieg

Das mozaikoo 3.5 Webdesign verwendet als Standard-Framework UIKit. Wir empfehlen, die UIKit-Klassen-Dokumentation zu lesen oder parat zu halten. Für JavaScript-Komponenten aus UIKit wurden ECMAScript-kompatible Wrapper geschrieben. Diese können als Komponenten im Templatedesign verwendet oder im Backend direkt im Editor als HTML aufgerufen werden. Style-Anweisungen werden standardmäßig in LESS geschrieben und zu CSS kompiliert. Wir empfehlen die LESS-SCSS-Dokumentation.

Als Einleitung in das Templating empfehlen wir die Templating Basics von Ember.

Als Editor zur Bearbeitung sowohl der Styles, als auch der Handlebars-Dateien (.hbs) empfehlen wir Visual Studio Code von Microsoft (kostenlos) mit Ember JS Snippets und Ember Language Server als aktivierte Erweiterungen sowie ES- und HTML-Linter nach Wahl.

---

* Für die Installations-Anleitung aus dem GIT-Repository, API-Deployment oder alternative Frontend-Apps in React, mobile Apps in React-Native oder als Portals, (s. google web.dev) wenden Sie sich bitte an den Support.

Templating-Einleitung

Dateien des Templates in der App

Die wichtigsten Verzeichnisse für Templating und Design sind

```
├── ...
├── app                         das Hauptverzeichnis der App
│   └── index.html              das Basis-HTML der App
│   └── styles                  LESS-Styles der App
│   │   └── app.less            Basis-LESS-Style der App
│   │   ├── ...                 weitere Style-Dateien
│   └── template                Handlebars (.hbs) Dateien der App
│   │   └── application.hbs     Basis .hbs-Template der App
│   ...
├── theme                       veränderte Template-Dateien der App
```

> **Dateien im Template-Verzeichnis**
>
> **Im Template-Verzeichnis sollte nicht gearbeitet oder dort Änderungen vorgenommen werden. Dazu die zu verändernden Dateien in das Theme-Verzeichnis in der identischen Ordnerstruktur kopieren, wie Sie im Template-Verzeichnis vorliegen.** Bei der Kompilierung der App werden dann die original Template-Dateien durch mit den angelegten Dateien aus dem Theme-Verzeichnis überschrieben.

index.html

Die **index.html** ist das HTML5-Boilerplate für die Webapp.

Es enthält die wichtigsten Favicon- und Manifest-Tags und die Links für das Einbetten des Codes aus der Webapp.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <title></title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="apple-touch-icon" sizes="152x152" href="/apple-touch-icon.png">
    <link rel="icon" type="image/png" sizes="32x32" href="/favicon-32x32.png">
    <link rel="icon" type="image/png" sizes="16x16" href="/favicon-16x16.png">
    <link rel="manifest" href="/site.webmanifest">
    <link rel="mask-icon" href="/safari-pinned-tab.svg" color="#5bbad5">
    <meta name="msapplication-TileColor" content="#2b5797">
    <meta name="theme-color" content="#ffffff">

    {{content-for 'head'}}

    <link rel="stylesheet" href="{{rootURL}}assets/vendor.css">
    <link rel="stylesheet" href="{{rootURL}}assets/webapp.css">

    {{content-for 'head-footer'}}

    {{content-for 'ga-script'}}

  </head>
  <body class="fastboot">
    <!--[if lte IE 9]>
    <div style="color: black;">
       <h3 style="color: black;">Ihr Browser wird nicht unterstützt</h3>
      Bitte installieren Sie einen aktuellen Browser
    </div>
```

```
    <![endif]-->

    {{content-for 'ga-frame'}}

    {{content-for 'body'}}

    <script src="{{rootURL}}assets/vendor.js"></script>
    <script src="{{rootURL}}assets/webapp.js"></script>

    {{content-for 'piwik'}}

    {{content-for 'body-footer'}}

  </body>
</html>
```

Änderungen in der index.html vermeiden

Änderungen in der index.html sollten vermieden werden. Alle wichtigen Elemente werden durch die WebApp dynamisch befüllt und erweitert. Dazu gehört auch z.B. der Titelaufbau (der title-Tag ist in der index.html deshalb leer). Nur falls spezielle Tracking-Elemente integriert werden sollen, spezielle Klassen oder ein anderer Doctype zwingend ausgeliefert werden soll, empfiehlt sich eine Änderung in der index.html.

Styles-Verzeichnis

Eigene Styles sollten im **Styles**verzeichnis angelegt werden. Dies kann direkt im Verzeichnis oder in einem Unterordner (wenn mehrere partielle Styledateien verwendet werden) geschehen:

```
├── app
│   └── styles
│   │   └── app.less            Basis-LESS-Style der App
│   │   ├── mein-style.less     eigene Style-Datei
```

Laden der Styleanweisung aus der app.less

Da die app.less automatisch in die Webapp eingebunden wird, empfiehlt es sich, die eigenen Styles aus der app.less über die @import Regel nachzuladen:

```
@import 'uikit';
@import 'uikit.theme.less';
@import 'fonts.less';
...
@import 'mein-style.less';
```

Sollten die Basisanweisungen aus UIKit oder der app.less nicht benötigt werden, können diese auskommentiert werden.

Dateien (fonts, png, jpg, svg), die über die LESS-Definitionen eingebunden werden, müssen sich im *public*-Verzeichnis befinden:

```
app                     (das Haupt-Applikationsverzeichnis)
public                  (das Verzeichnis für öffentlich zugängliche Dateien)
├── assets              (Verzeichnis für Dateien, die direkt verwendet werden)
│   ├── img             z.B. Verzeichnis für Bilder
│   └── fonts           z.B. Verzeichnis für Schriften
│   ...
```

> Keine CSS-Dateien in den Public-Ordner legen oder diese dort modifizieren.

In der LESS-Datei können zu verlinkende Dateien relativ, also ausgehend von der späteren Position im *public*-Verzeichnis angegeben werden, z.B.:

```
    .mein-stil {background-image:url(img/example.png);}
```

Sowohl beim späteren Deployment, als auch während der Entwicklung werden die Änderungen in der LESS-Datei und die Dateizuweisungen zum *public*-Verzeichnis korrekt umgesetzt.

Styles werden im Kapitel Design näher erläutert.

Theme-Verzeichnis

Das **Theme**-Verzeichnis enthält alle Dateien, deren Seitenaufbau verändert wurde. Bevor Veränderungen an den Templatedateien vorgenommen werden, sollten diese in das Themeverzeichnis kopiert werden.

Es ist wichtig, dass die Ordnerstruktur und der Name der kopierten Datei im Themeverzeichnis identisch mit der Struktur im Templateverzeichnis ist:

```
├── template
│   └── components
│   │   └── detail
│   │   │   └── document-files.hbs

...

├── theme
│   └── components
│   │   └── detail
│   │   │   └── document-files.hbs
```

Übersicht der Templatebestandteile

Auf der ersten Ebene im im **Template**verzeichnis befinden sich die Dateien, die für den spezialisierten Seitenaufbau notwendig sind:

```
├── template
│   └── application.hbs             Basisseite
│   └── index.hbs                   Renderer für <main>
│   └── head.hbs                    <head> Ergänzungen
│   └── page.hbs                    Logik für Seitentypen
│   └── account.hbs                 Basis für den Benutzeraccount

...

│   └── login.hbs                   Login-Seite
│   └── logout.hbs                  Logout-Seite
│   └── register.hbs                Registrierungs-Seite
│   └── verify-email.hbs            Überprüfung Registrierungs-E-Mail
│   └── request-password-reset.hbs  Passwort zurücksetzen
│   └── password-reset.hbs          Passwort zurücksetzen
```

sowie die Detailverzeichnisse:

```
├── template
│   └── account                     Alle Account-Funktionsseiten
│   └── components                  Alle Funktionsseiten
│   └── content                     Alle Partials der application.hbs
```

Was sind *Partials*

Partials sind .hbs-Dateien, die innerhalb einer .hbs-Datei integriert werden können. Das vereinfacht die .hbs-Datei und sorgt für mehr Übersichtlichkeit. Besonders interessant ist die Verwendung von Partials, wenn identische Inhalte in einer Template-Datei mehrfach integriert werden sollen, aber keine besondere Funktionalität haben. Partials sind an Ihrem Dateinamen zu erkennen, er beginnt immer mit einem "-".

Innerhalb des Content-Ordners befinden sich alle Partials, die von der application.hbs aufgerufen werden. Es sollten für jede Position aus der application.hbs auch eine Position in den Templateinstellungen existieren. Wie Positionen angelegt werden erklärt die Templatekonfiguration.

Konfigurationsmöglichkeiten des Templates

Positionen des Templates

mozaikoo verwendet Seiten und Widgets, um die Seiten im CMS aufzubauen. Seiteninhalte werden abhängig vom Typ innerhalb des <main>-Tags ausgegeben. Die Widgets können Positionen und Seiten zugeordnet werden, um wiederkehrende Inhalte oder Inhalte mit spezieller Funktionalität auszugeben.

Standardpositionen des Standardtemplates von mozaikoo.

> **Standardpositionen können frei verändert werden**
>
> Die gezeigten Positionen sind in der application.hbs und der Template-Konfigurationsdatei angelegt. Sie können die Positionen entfernen, neue definieren und in der application.hbs mittels HTML an anderer Stelle anordnen.

Die Positionen des Templates, die dann im Backend für Widgets verfügbar sind, werden in der Template-Konfigurationsdatei frei definiert:

```
├── config               (das Konfigurationsverzeichnis)
│   └── template.json    Konfigurationsdatei des Templates
```

Die Positionen werden unter **positions** eingetragen:

```
    "positions": {
        "interner-name": "Dargestellter Name",
    ...
```

Der interne Name wird im Handlebar-Template verwendet, der dargestellte Name wird im Backend bei der Ausgabekonfiguration von Inhalten an den Positionen angezeigt.

> **MAIN bzw. Seiteninhalt**
>
> Für den Inhalt, der in <main> gerendert wird (in der Grafik **Seiteninhalt**), wird keine Position angegeben.

Positionen in der application.hbs

application.hbs ist das Haupttemplate. Dieses Template regelt die Ausgabe der Standardseite und den in der Template-Konfigurationsdatei definierten Positionen:

```
{{#if media.isMobile}}
  <nav id="offcanvas">
    {{partial "content/offcanvas"}}
  </nav>
{{/if}}

{{#if userProxy.isLoggedIn}}
  <dialog>
    {{partial "editor"}}
  </dialog>
{{/if}}

<header>
    {{partial "content/logo"}}
```

```
  <nav>
    {{partial "content/menu"}}
  </nav>
</header>


{{partial "content/top"}}

{{partial "content/breadcrumbs"}}

<main>

  {{partial "content/main-top"}}

  <!-- The Main-Content -->
  {{outlet}}

  {{#if widgets.sidebar}}
    <aside>
      {{partial "content/sidebar"}}
    </aside>
  {{/if}}

  {{partial "content/main-bottom"}}

</main>

{{partial "content/bottom"}}

<footer>
  {{partial "content/footer"}}
</footer>
```

Reduzierter Beispielaufbau der Positionen in der application.hbs

application.hbs und <body>

In der application.hbs wird kein **<body>** definiert. Der Inhalt der application.hbs steht direkt innerhalb des **body**-Tags aus der index.html.

Responsive Breakpoints ohne Media-Query

mozaikoo verwendet breakpoint.js, um die Anzeige von responsiven Inhalten zu regeln. Der Vorteil von breakpoint.js ist, dass Inhalte, die im definierten Media-Query nicht dargestellt werden sollen, auch nicht im DOM gerendert werden. Ember steuert automatisch das Erzeugen der Elemente, wenn andere Media-Query-Einstellungen ein Element anzeigen.

Media-Queries im CSS

Es können im CSS/LESS auch weiterhin Media-Queries definiert werden, jedoch ist aus Performancegründen die Steuerung über breakpoint.js vorzuziehen. Elemente, die über ein CSS-Media-Query gesteuert werden befinden sich immer im DOM, auch wenn diese nicht dargestellt werden.

Die Datei kann editiert werden und befindet sich im App-Verzeichnis auf der obersten Ebene.

Die in breakpoint.js definierten Media-Queries stehen im Backend für die Widgets zur Konfiguration zur Verfügung.

```
export default {
  mobile:  '(max-width: 767px)',
  tablet:  '(min-width: 768px) and (max-width: 959px)',
  desktop: '(min-width: 960px) and (max-width: 1199px)',
  large:   '(min-width: 1200px)'
};
```

Die Anweisungen in breakpoint.js entsprechen dabei den bekannten Media-Query-Anweisungen aus CSS/LESS/SASS und können entsprechend erweitert werden, z.B.:

```
    tabletLandscape:  '(min-width: 768px) and (max-width: 1024px) and (orientation:
landscape)',
```

Breakpoint.js stellt zudem beliebig erweiterbare Variablen zur Verfügung, die in den Template-Dateien verwendet werden können. So ist es zum Beispiel denkbar, eine Variable zu definieren, die mit einem `{{#if}}` Statement nur Elemente einblendet, wenn der Viewport der Seite eine geringe Höhe hat:

```
    smallHeight: '(max-height: 800px)'
```

Im Template werden die Media-Queries (z.B. mittels IF-Abfrage) wie folgt eingebunden:

```
    {{#if media.isMobile}}
    ... Anweisung für die mobile Ausgabe
    {{/if}}
```

Die definierte Variable wird im media-Objekt mit vorangestelltem `is` in **camelCase** abgefragt.

Komponenten

Jede eingebundene Komponente hat einen eigenen Template-Ordner. Der Aufbau dieser Templates wird unter Templateaufbau erläutert.

Templatekomponenten

Die wichtigsten Typen. Kleine Komponenten, die innerhalb eines Templates verwendet werden haben auch ein hbs.

Account

Eine Besonderheit der Account-Komponente ist, dass die Komponente auch eine Route besitzt, die nur für den jeweiligen Benutzer zugänglich ist. Der Accout besitzt eine öffentliche Seite (Profil) und Seiten, die nur für den eingeloggten Benutzer erreichbar sind (/account).

In dem Verzeichnis **account** befinden sich alle Routen, die im Backend für die jeweilige Benutzergruppe freigeschaltet wurden.

Contentbuilder

Info wird zur Zeit überarbeitet.

Detail

In der Route Detail befinden sich alle Container, die bei einer Detailanzeige einer dynamischen Seite verwendet werden.

Kategorie-Container

Zeigt die Detailansicht bei Aufruf einer Kategorie.

Dieser Container wird rekursiv verwendet. D.h. wenn eine Unterkategorie ausgewählt wird, wird derselbe Container verwendet, um den Inhalt einer Unterkategorie darzustellen.

Dokumenten-Container

Zeigt die Detailansicht bei Aufruf eines Dokumentenverzeichnisses.

Dieser Container wird rekursiv verwendet, wenn die Option "Navigation" aktiviert ist. D.h. wenn eine Unterverzeichnis ausgewählt wird, wird derselbe Container verwendet, um den Inhalt des Unterverzeichnisses darzustellen.

Entry-Container

Zeigt die Detailansicht eines Entries.

Event-Container

Zeigt die Detailansicht einer Veranstaltung.

Produkt-Container

Zeigt die Detailansicht eines Produktes.

POI-Container

Zeigt die Detailansicht eines Point-of-Interest.

Profile-Container

Zeigt die öffentliche Ansicht eines Profils.

Newsletter-Container

Zeigt die öffentliche Ansicht eines Newsletters.

Diese Route wird verwendet, um Voransichtslinks von Newslettern und Links in Newsletter-Mails zu generieren, die den Newsletterinhalt innerhalb der Website anzeigen.

Map

Enthält die Templates, die für die Anzeige der Map angepasst werden können:

- **base-map** Die Standardkarte
- **detail-map** Die Karte auf einer Detailseite eines Karteneintrags

- **map-popup** Das Popup-Panel bei Auswahl eines Markers auf der Kartenübersichtsseite
- **map-popup-detail** Das Popup-Panel auf der Detailseite eines Karteneintrags

sowie die Komponentenanpassung für die Suche auf der Karte mit **search-map**.

Page

Wird zur Zeit überarbeitet.

Press

Templates für das Redaktionssystem.

Search

Die Hauptkomponente der Suche ist der **search-container**.

Das verzeichnis Search enthält zudem die Templates der untergeordneten Komponenten, die für den partiellen Aufbau der Suchseite und deren Filter notwendig sind:

- **category-browser** und **category-browser-flat** für die Kategorieanzeige auf der Suchseite
- **selected-categories** für die Anzeige der als Filter ausgewählten Kategorien auf der Suchseite
- **search-tags** für die Anzeige der Tags auf der Suchseite
- **search-paging** für die Anzeige des Pagings auf der Suchseite

sowie das **search-result-item** das den Aufbau des einzelnen Suchergebnisses regelt.

Widgets

Der Widgets-Ordner enthält die Templates der verfügbaren Widgets, die im Backend zugewiesen werden können.

Die Widgets-Templates enthalten nur die Ausgabesteuerung der Widgets in der Webapp.

Die Templates für den Aufbau der Eingabe im Backend befinden sich in der **Admin-Webapp**.

Handlebar-Helper für mozaikoo Templates

Helper für Datums- und Zeitformatierung

mozaikoo Ember-Applikationen werden mit installierten **ember-moment** (Dokumentation) und den dazugehörigen Handlebar-Helpern ausgeliefert.

Helper für Dateien

image-path

background-image

media-path

file-path

file-type

file-icon

file-extension

Helper für logische Operationen

equal

and

not

or

falsy-to-false

is-empty-string

Helper für Formatierungen

ember-moment

number-to-human

format-percent

format-price

format-distance

round-number

ps-to-kw-round

is-last

is-long-text

shorten-text

concat-if

Helper für Schleifen

array-slice

filter-array-props

filter-by-parent

flat-child-cats

is-last

Helper für mathematische Operationen

Addition

Template-Einstellungen im Backend

Template-Einstellungen im Backend

Übersetzungen

Übersetzungen für Inhalte aus den Templates

In den Templates sind unter Umständen Textpassagen enthalten, die nicht über das Backend eingegeben wurden.

Die Textpassagen werden mit dem Übersetzungshelper `{{t 'geparste.übersetzungs.stelle'}}` definiert.

Die dazugehörigen Übersetzungsdateien befinden sich im Übersetzungsverzeichnis der App:

```
app                             (das Haupt-Applikationsverzeichnis)
├── locales                     (Sprachverzeichnis)
│   ├── de                      Sprachkürzel ISO-2-Digit
│   │   └── config.js           Konfiguration der Sprache
│   │   └── translations.js     Übersetzungsangaben
│   ...
```

Die Übersetzungsdatei muss valides JSON sein. Der Aufbau der Übersetzungsvariablen ist frei gestaltbar:

```
export default {
 "gruppe": {
     "uebersetzungsName": "Das ist meine Übersetzung",
 ...
```

Die Benennung der Übersetzungsvariablen sollte in `camelCase` erfolgen (mit Kleinschreibung beginnen, zweites Wort mit führendem Großbuchstaben ohne Trennzeichen direkt an das erste Wort).

Die Übersetzungsvariable kann im Template dann in Handelbar-Brackets verwendet werden:

```
{{t "gruppe.uebersetzungsName"}}
```

Aufruf des Übersetzungshelpers mit `t`.
Name der Variable in Anführungszeichen.

Übersetzung in HTML-Attributen

Steht die Übersetzung selbst im HTML in einer Attributbeschreibung in Anführungszeichen mit weiteren Attributen muss der Variablenname in 'Einzel-Anführungszeichen' angegeben werden.

Einleitung

Wenn mozaikoo mit der Standard-Ember-Webapp ausgeliefert wird, wird das Routing auf der Domain von der App gesteuert. Man spricht von einer Single-Page-Application (SPA). Das Routing von Ember sorgt dafür, dass die URLs so ausgegeben werden, wie Sie im Backend festgelegt wurden.

Aufgrund des schnellen Seitenaufbaus und der parallelen Auslieferung mit Fastboot als statische Seiten, haben wir auf sogenannte Page-Transitions beim Wechsel einer URL verzichtet. Diese könnten auch mit Ember nachimplementiert werden, wenn gewünscht, zum Beispiel mit liquidfire. Ist eine vollständige native Anmutung für mobile Apps gewünscht, empfehlen wir, die Frontend-App mit React oder React-Native auszuliefern.

Designprinzipien

Wir verzichten in dieser Anleitung auf spezifische UI/UX-Prinzipien. Die Basis-Templates von mozaikoo sind gemäß den aktuellen W3C-Richtlinien aufgebaut und ergänzen entsprechende Tags mit role-Attributen und alt-Tags für Accessibility.

Besonders für „Mobile-First"-Applikationen sollte aber ein angepassten Deployment gegebenenfalls ohne UIKit in Erwägung gezogen werden. Es empfiehlt sich die Auslieferung als React/Reactnative App oder nach wie vor als Ember-App mit spezialisiertem Design für Android (Material-Design) oder IOS mit Ionic-CSS.

Wichtige Dateien für das Design

Die wichtigsten Verzeichnisse für das Design sind

```
├── ...
├── app                         das Hauptverzeichnis der App
│   └── index.html              das Basis-HTML der App
│   └── styles                  LESS-Styles der App
│   │   └── app.less            Basis-LESS-Style der App
│   │   ├── ...                 weitere Style-Dateien
├── public                      Dateien, die unter '/' erreichbar sind
│   │   favicon.png             favicon-Datei
│   │   ...                     weitere Icon-Dateien
│   └── images                  Bilddateien für LESS-Styles und Templates
│   └── fonts                   Schriftdateien für LESS-Styles
```

Seitengestaltung und Layout

Welche Positionen im Template vorhanden sein sollten, damit das Design auf der Webseite implementiert werden kann, muss einerseits in der Templatekonfiguration festgelegt werden und andererseits in der application.hbs und deren Partials umgesetzt werden.

Container oder Full-Screen

> **Inhalt in fester Breite?**
>
> Wie soll der Seiteninhalt auf der Seite dargestellt werden? Soll sich der Inhalt immer innerhalb einer maximalen breite bewegen oder soll das Layout die gesamte Bildschirmbreite einnehmen? In mozaikoo können sowohl **alle** oder **keine** Seiten automatisch mit Containerklassen ausgeliefert werden. Ist **keine** als Option gewählt muss bei Seiten, bei denen die Laufweite durch Container begrenzt sein soll, diese manuell angegeben werden.

Wenn Inhalte nicht über eine definierte Standardbreite hinausgehen, werden diese Inhalte innerhalb eines Containers dargestellt. UIKit besitzt dafür spezielle Klassen. In mozaikoo kann festgelegt werden, ob jede Seite innerhalb eines Containers automatisch dargestellt werden soll, oder ob die Seiten ohne Container rendern.

Alle Seiteninhalte innerhalb von Containern

Über die Templatekonfiguration kann festgelegt werden, dass alle Seiteninhalte innerhalb eines Container dargestellt werden.

Verantwortliche Containerklassen sind aus UIKit `uk-container`, `uk-container-center`, sowie die Größenklassen `uk-container-expand`, `uk-container-large`, `uk-container-small`, `uk-container-xsmall`.

Farbflächen der jeweiligen Bereiche (sofern nicht anders definiert) ragen über diesen Containerbereich hinaus und füllen den gesamten Bildschirm.

Container-Beispielseite mit UIKit.

Alle Seiteninhalte über volle Bildschirmbreite

Über die Templatekonfiguration alle Containerklassen entfernen (verantwortliche Containerklassen sind aus UIKit `uk-container`, `uk-container-center`, `uk-container-expand`).

Gemischte Darstellung

Über die Templatekonfiguration aus den gewünschten Seitentypen und Positionen die Containerklassen entfernen.

Verantwortliche Containerklassen sind aus UIKit `uk-container`, `uk-container-center`, sowie die Größenklassen `uk-container-expand`, `uk-container-large`, `uk-container-small`, `uk-container-xsmall`.

Danach auf den Seiten – die innerhalb eines Containers dargestellt werden sollen – im Backend Seitenklassen mit Containern angeben, z.B. `uk-container uk-container-center`:

Styleanweisungen

Verwenden von UIKit-Klassen in den Templatedateien

Prinzipiell sollten bereits in den .hbs zur allgemeinen Formatierung die UIKit-Klassen verwendet werden. Durch die Kompressionsalgorithmen von Ember werden die .hbs-Dateien automatisch mit Short-Codes der Klassen versehen. Die Verwendung identischer Klassen verursacht also wenig Speicherbedarf.

Besonderheiten bei der Verwendung von UIKit-Klassen

Da UIKit zum Beispiel die `uk-grid` Klasse erst durch Javascript über das Attribut berechnet und automatisch hinzufügt, ist es notwendig für das Aussehen statischer Seiten, dass die Klasse trotzdem aktiv gesetzt wird:

```
    <div uk-grid>
```

Normalerweise ausreichend in UIKit

```
    <div class="uk-grid" uk-grid>
```

Empfohlen für das Aussehen statischer Seiten

Für eine Reihe weiterer UIKit-Klassen wurden fehlende Definitionen (z.B. Rendern von `a-tags` als Blockelement) in der **app.less** ergänzt.

Verwenden von LESS-Variablen

Für viele Elemente wurden Variablen definiert, die sich in der **variables.less** befinden und zusätzlich in der **app.less**.

Diese Variablen sollten für eine schnelle Farb- oder Dimensionsanpassung der Elemente in der eigenen LESS-Datei überschrieben werden.

```
// Global variables
// ========================================================================

//
// Typography
//

@global-font-family:                    Source, -apple-system...
@global-font-size:                      15px;

@global-xxlarge-font-size:              38px;
@global-xlarge-font-size:               30px;
@global-large-font-size:                24px;
@global-medium-font-size:               20px;
@global-small-font-size:                14px;

@global-emphasis-color:                 #222;

@global-line-height:                    18px;

//
// Base
//

@base-code-font-family:                 'Roboto Mono', monospace;
@base-code-font-size:                   12px;

@base-heading-font-weight:              300;
```

```
@base-pre-font-size:                    12px;
@base-pre-padding:                      25px;
@base-pre-background:                   @global-muted-background;
@base-pre-border-width:                 0;
@base-pre-border-radius:                0;

.hook-base-body() {
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
    text-rendering: optimizeLegibility;
}

//
// Container
//

@container-max-width:                   1380px;
@container-small-max-width:             650px;

//
// Navbar
//

@inverse-navbar-nav-item-color:         @inverse-global-color;
@inverse-navbar-nav-item-hover-color:   @inverse-global-emphasis-color;

@header-height:                         80px;

//
// Nav
//

@nav-header-font-size:                  12px;

//
// Subnav
//

@inverse-subnav-item-color:             @inverse-global-color;
@inverse-subnav-item-hover-color:       @inverse-global-emphasis-color;

//
// Tab
//

@tab-item-padding-horizontal:           20px;
@tab-item-padding-vertical:             9px;
@tab-item-border-width:                 2px;
@tab-item-font-size:                    12px;

.hook-tab-item() { line-height: 20px; }

//
// Table
//

@table-header-cell-font-size:           12px;

//
// Label
//

@label-font-size:                       12px;

//
// Text
//

.hook-text-lead() { font-weight: 300; }
.hook-text-large() { font-weight: 300; }

//
// Utility
//

@inverse-logo-color:                    @inverse-global-emphasis-color;
```

```
@inverse-logo-hover-color:                  @inverse-global-emphasis-color;

//
// Off-canvas
//

@offcanvas-bar-background:                   #fff;
@offcanvas-bar-color-mode:                   dark;

//
// Inverse
//

@inverse-global-color:                       fade(@global-inverse-color, 80%);
@inverse-global-muted-color:                 fade(@global-inverse-color, 60%);

//
// Colors
//

@global-color:                               #666;
@global-emphasis-color:                      #333;
@global-muted-color:                         #999;

@global-link-color:                          #1e87f0;
@global-link-hover-color:                    #0f6ecd;

@global-inverse-color:                       #fff;

@global-primary-color:                       #1e87f0;
@global-secondary-color:                     #999;


@primary-color:                              rgb(220,0,0);
@primary-color-light:                        rgba(200,0,0,0.3);
@primary-link-color:                         rgb(150,0,0);
@primary-hover-color:                        rgb(200,0,0);
@primary-background-color:                   rgb(160,0,0);
@primary-color-active:                       rgb(255, 80, 80);

//
// Backgrounds
//

@global-background:                          #fff;

@global-muted-background:                    #f8f8f8;
@global-primary-background:                  #1e87f0;
@global-primary-dark-background:             #0f6ecd;
@global-secondary-background:                #222;

@global-success-background:                  #32d296;
@global-warning-background:                  #faa05a;
@global-danger-background:                   #f0506e;

//
// Borders
//

@global-border-width:                        1px;
@global-border:                              #e5e5e5;

//
// Box-Shadows
//

@global-small-box-shadow:                    0 2px 8px rgba(0,0,0,0.08);
@global-medium-box-shadow:                   0 5px 15px rgba(0,0,0,0.08);
@global-large-box-shadow:                    0 14px 25px rgba(0,0,0,0.16);
@global-xlarge-box-shadow:                   0 28px 50px rgba(0,0,0,0.16);

//
// Spacings
//

// Used in margin, section, list
@global-margin:                              20px;
```

```
@global-small-margin:                          10px;
@global-medium-margin:                         40px;
@global-large-margin:                          70px;
@global-xlarge-margin:                         140px;

// Used in grid, column, container, align, card, padding
@global-gutter:                                30px;
@global-small-gutter:                          15px;
@global-medium-gutter:                         40px;
@global-large-gutter:                          70px;

//
// Controls
//

@global-control-height:                        40px;
@global-control-small-height:                  30px;
@global-control-large-height:                  55px;

//
// Z-index
//

@global-z-index:                               1000;


//
// Sidebar
//

@sidebar-left-width:     240px;
@sidebar-left-width-xl:  300px;

@sidebar-right-width:    200px;
@sidebar-right-left:     0px;
@sidebar-right-left-xl:  60px;


//
// Form
//

@internal-form-checkbox-image:                  "/images/form-checkbox.svg";
```

Breakpoint-Klassen

UIKit verwendet `@xl`, `@l`, `@m` und `@s` als Klassenerweiterung für Breakpoints. Diese wurden unverändert beibehalten.

Für die Klassen `uk-hidden@...` und `uk-visible@...` wurde eine alte UIKit-Klasse aus UIKit2 zusätzlich definiert, um Elemente zum Beispiel nur für die Tabletansicht anzuzeigen.

Eigene Klassen- und Style-Definitionen

Die Klassen in der Standard-Webapp sind nicht nach BEM-Methodologie (Block Element Identifier) mit dem Doppel-"-" aufgebaut, folgen aber der Nomenklatur im Aufbau als Blockelemente → Elemente → "Modifier" (s. BEM Einführung).

Das Blockelemente ohnehin aus HTML5 geerbt werden (und im Templateaufbau verwendet werden), Elemente in Ember mit Komponentenfunktionalität erkennbar sind und
Ember je nach Komponente eigene „Modifier"-Klassen ergänzt, kann auf BEM verzichtet werden, ohne an Übersichtlichkeit zu verlieren.

Es bleibt dem Designer frei, für seine eigene Notation der BEM-Methodik zu folgen.

Klassen im Standardtemplate

Die Gliederung im Template (Positionen) erfolgt mit abgekürzten Templateklassen (tm-), zum Beispiel `tm-top` oder `tm-bottom`.

Die Gliederung der Seite folgt den W3C-Empfehlungen zum Seitenaufbau.

Semantischer Seitenaufbau

mozaikoo folgt den Empfehlungen des W3C zum semantischen Aufbau einer Webseite mit den entsprechenden HTML5-Bereichen bzw. -Tags:

HTML5 Bereichs-Tags und Layoutblöcke

- **header** ist der Kopfbereich des Dokuments oder eines Bereichs
- **footer** ist die Fußzeile des Dokuments oder eines Bereichs
- **nav** hat alle Navigations-Links des Dokuments und ist gegliedert in
- **menu** mit menuitem als einzelne Links innerhalb des Menüs
- **main** umfasst den Hauptinhalt der Seite mit
- **section** als thematische Gruppierung eines Inhalts mit Kopfzeile
- **article** als Artikel mit eigenständigem Hauptinhalt
- **aside** mit weiterführender Information zum Artikel
- **address** mit Kontaktinformation zum Autor, Eigentümer
- **figure** für eigenständige bildliche Information (Illustrationen, Diagramme, Fotos oder auch Code-Blöcke)

Per Definition können sections innerhalb von article, als auch article innerhalb mehrerer sections genutzt werden. Das header-Element kann mehrfach verwendet werden und wird entsprechend seiner Verschachtelungstiefe interpretiert. Zum Beispiel header eines Dokuments und header innerhalb eines article.

HTML5 Element-Tags

Für die Anzeige

- **figcaption** für Bildunterschriften
- **progress** für Fortschrittsanzeigen
- **dialog** für Dialoganzeigen
- **details** für ergänzende, einblendbare Informationen
- **summary** Titel für details
- **mark** für hervorgehobene Textpassagen

Für Einheiten

- **meter** für skalare Maßeinheiten
- **time** für Datums- und Uhrzeitangaben

Für Layoutzwecke

- **wbr** möglicher Zeilenumbruch
- **bdi** für einen Bereich in anderer Leserichtung
- **rt** für Betonungs-Hervorhebungen (ostasiatische Typographie)

Für Grafiken

- **canvas** für script-basierte Grafiken
- **svg** für skalierbare Vektorgrafiken

Für Medien

- **audio** für Audiodateien
- **embed** für externe Nicht-HTML-Elemente
- **video** für Videodateien
- **source** definiert mehrere Medienressourcen für video und audio
- **track** für Textspuren von video und audio

HTML5 Formularelemente

Form-Elemente und -Attribute

- **autocomplete** für automatische Vervollständigung des Formulars
- **novalidate** für das Aufheben der Validierung des Formulars
- **formaction, formenctype, formmethod, formtarget** für die Beschreibung der Übermittlungsart des Formulars.

Input-Elemente

- **datalist** für eine Liste vordefinierter Optionen
- **list** als Attribut für die Zuweisung einer datalist
- **output** für Kalkulationsergebnisse aus Inputfeldern

```
<input list="browsers">
<datalist id="browsers">
    <option value="Internet Explorer">
    <option value="Firefox">
    <option value="Chrome">
    <option value="Opera">
    <option value="Safari">
</datalist>

<form oninput="x.value=parseInt(a.value)+parseInt(b.value)">0
  <input type="range" id="a" value="50">100
  +<input type="number" id="b" value="50">
  =<output name="x" for="a b"></output>
</form>
```

Input-Standard-Attribute

- **button** für Inputfelder, die als Buttons angezeigt werden
- **checkbox** für Ankreuzfelder mit keiner, einer, mehreren Optionen
- **radio** für eine einzelne Auswahlmöglichkeit, gruppiert über name
- **password** für Passwortfelder (Textanzeige als Punkte)
- **text** für einzeilige Texteingaben

submit für das Absenden des Formularinhalts

- **reset** für das Zurücksetzen aller Formularfeld-Inhalte

Input-HTML5-Attribute

(für Browser ohne HTML5, wird der Inhalt wie ein Inputfeld text ausgegeben)

- **date** für eine Datumsangabe, die in Tag, Monat, Jahr gegliedert ist
- **datetime-local** für grupperte Datums- und Zeiteingabe
- **month** für die Auswahl eines Monats und Jahres
- **week** für die EIngabe einer Kalenderwoche und Jahr
- **time** für Zeiteingaben in Stunden, Minuten und Sekunden
- **image** für die Angabe eines Bildes (src) zum Rendern des Feldes
- **number** für numerische Eingaben
- **range** für die Eingabe eines Wertes innerhalb eines Bereichs
- **search** für Suchfelder (verhält sich wie ein Textfeld)
- **tel** für Telefonnummern (Angabe der Gliederung mit pattern)
- **url** für Webadressen
- **color** für eine Farbauswahl mit Color-Picker
- **email** für E-Mail-Adressen
- **file** für Dateien, mit Dateiauswahldialog

Input Restriktionen und Erweiterungen

- **autofocus** fokussiert das Feld bei Seitenaufruf
- **disabled** für ein Feld, dass nicht aktiv ist (keine Eingabe möglich)
- **hidden** für versteckte, nicht angezeigte Felder
- **formnovalidate** für Submit. Überschreibt die Formularvalidierung
- **min, max** für vorgegebene numerische Minimal- / Maximalwerte
- **maxlength** für die maximale Anzahl an Ziffern oder Buchstaben
- **multiple** für die Eingabe meherer Werte in einem Inputfeld
- **pattern** zur Eingabevalidierung mit „regular Expression“ (RegEx)
- **placeholder** für eine Textanzeige. Wird bei Eingabe überschrieben.
- **readonly** für externe Nicht-HTML-Elemente
- **required** für Pflichtfelder
- **step** für die Angabe eines Intervalls bei numerische Feldern

Installationsvorbereitungen

Ein CodeEditor Ihrer Wahl sollte installiert sein. Wenn noch kein Code-Editor installiert ist, reicht der Visual Studio Code von Microsoft (kostenlos) aus.

Für einen Computer mit Linux oder Mac OS X ist Git bereits vorinstalliert. Windowsbenutzer können diesen GIT-Installer verwenden.

Auswahl der Webapp aus dem Repository

Für Testzwecke kann direkt aus dem Master-Branch der Webapp ein eigener Branch für die eigene Webentwicklung abgezweigt werden.

> **Achtung**
>
> Es werden keine Pull-Requests aus diesem Branch angenommen, wenn der Entwicklungszweig nicht vorher abgestimmt wurde.

Wir weitergehende Entwicklungszwecke, besonders mit eigener API sollte ein eigenständiges Repository angelegt werden.

Installation der API

Für Entwicklungstests kann die verfügbare API von demo.mozaikoo.com verwendet werden.

Wenn Sie eine eigenständige API benötigen, wenden Sie sich bitte an den Support.