Notationen und Organisationsformen für CSS bzw. SCSS

Inhaltsverzeichnis

1 CSS

Grundsätzlich gibt es für CSS keine verpflichtende allgemeine Systematik zum Schreiben von Dateien.
Hilfreich sind die folgenden Vorgehensweisen:

- Allgemeine Regeln an den Anfang
- Gruppierung von zusammengehörigen Selektoren
- Hohe Spezifität ans Ende

https://www.mediaevent.de/css-organisieren/

1.1 Zweck von Notationen

Beim Schreiben einer CSS-Datei sammeln sich oft mehrere hundert Zeilen, die ungeordnet aufgeschrieben sind. Eine Regel wird erdacht, sequentiell unter die bereits bestehenden geschrieben und auf Grund der Spezifitätsregeln greift sie vielleicht nicht. Das Ergebnis ist oft eine ungeordnete, schlecht wartbare Datei.

Einheitliche Notationen dienen folgenden Zwecken:

- ***Wartbarkeit*** der Dateien
- Ausschalten einiger ***Spezifitätsregeln***
- Einhaltung einheitlicher ***Namenskonventionen***
- Modularisierung und ***Wiederverwendung*** einzelner Bestandteile.

Alle folgenden Konventionen sind freiwillig und keine rigiden Frameworks.
Aufgelistet sind die meist bekanntesten und es ist immer möglich eine Mischform zu verwenden oder eine neue eigene Form zu erschaffen.

2 BEM Block Element Modifier

Ziel ist ein konsistentes Konzept für die Benennung von Klassen und die ausschließliche Verwendung von Klassenselektoren, um die sequentielle Abarbeitung zu gewährleisten, sowie die Wiederverwendung von Regeln zu erreichen.

- Konzept für Klassenbezeichner
- Wiederverwendung von Regeln

BEM Regeln:

- alle Selektoren bestehen aus nur einer Klasse
- andere Selektoren, wie Typ oder id werden nicht verwendet
- Verschachtelungen und Nachfahren sind nicht erlaubt
- !important wird nicht verwendet

Daraus folgt, jedes zu stylende Element muß genau ein Klassenattribut mit mindestens einem Wert besitzen.

Nachteil: Der HTML-Code wird dadurch immens aufgebläht.

2.1 BEM-Klassenbezeichner

Block

Ein Block ist ein beliebiger Container mit einem Basis-Styling.
Er enthält mehrere **Elemente**, die zusammengehören.
Ein Block kann einen **Modifier** haben, der das Element verfeinert.
Modifier können nicht für sich alleinstehen.

Block_ _element--modifier

Block—modifier

Block

BEM-Beispiele

```
<figure class="pic">

     <img class="pic__img">

     <figcaption class="pic__caption"></figcaption>

</figure>
```

Blöcke stehen für sich allein.
.header
.nav
.main
.article
.ul, .ol
.picture, .figure
.form
.checkbox

Elemente sind an einen Block gebunden und können nicht für sich alleinstehen.
.header__title
.nav__item
.picture__caption
.checkbox__label

Modifier können ebenfalls nicht für sich alleinstehen, sie zeichnen Blöcke oder Elemente besonders aus.
.nav__item--hover
.checkbox—disabled

https://9elements.com/bem-cheat-sheet/

3 SMACSS Scalable Modular Architekture for CSS

Für die Namensgebung der Klassen kann die BEM-Notation verwendet werden.

Auch hier handelt es sich um eine Organisationsform von CSS zur besseren Wartbarkeit und Wiederverwendung. Zusätzlich zu Regeln für Selektoren werden Regeln für die Verteilung auf mehrere Dateien oder Verzeichnisse gegeben.

3.1 Kategorien

*Kategorien werden vergeben **von allgemein** und wiederverwendbar ...*

1. Base
2. Layout
3. Module
4. State
5. Theme

*... **zu spezifisch** und nur für einzelne Projekte*

Jede Kategorie ist ein Verzeichnis oder eine Datei.
In jeder Kategorie könnten sich eine oder mehr Dateien befinden.

3.2 Naming Rules

Die Namenskonvention zeichnet die jeweilige Kategorie aus.

Das kann z.B. mit einem Präfix erreicht werden:
layout ***l-*** state ***is-***

Module ohne Präfix, sondern mit Klassenname (z.B. nach BEM)

Base

- default styles
- alles was in einen reset oder eine foundation gehört

Hier werden hauptsächlich folgende Selektoren verwendet:

- einfache Typ Selektoren

aber auch:

- Attribut Selektoren
- Pseudoklassen, Pseudoelemente
- Nachfahren Selektoren und Geschwister

Keine ID-Selektoren (höchste Spezifität und deshalb nur schwierig zu überschreiben)

Defaults für:

- Überschriften
- links
- buttons
- Hintergründe

Das alles kann dann zu einem eigenen Reset führen.

```
html, body, form, a, input[type=…]
```

Layout

Auszeichnung der Hauptbereiche einer Site.

```
header, footer, article, main
```

Wiederverwendbare Bereiche,
gerade im Mobile-Bereich finden sich oftmals Wiederholungen.

Vorwiegend ID-Selektoren, aber auch Klassen Selektoren

Für die ID’s keinen Präfix verwenden

#article

#main

Für die Klassen mit Präfix *l-* arbeiten

Beispiel

```
<header id="header"></header>

<nav id="primarynav"></nav>

<main id="maincontent"></main>

#header { … }

#primarynav { … }

#maincontent { … }
```

Beispiel

```
article ul {

<ul class=„l-grid“>

    <li><a href="…">…</a></li>

    <li><a href="…">…</a></li>  …

</ul>

.l-grid {

    margin: 2%;

    padding: 0;

    display: flex;

    flex-flow: row wrap;

    justify-content: space-between;

}

.l-grid > li {

    flex: 1 1 25em;

}
```

Layouts bestehen aus einem oder mehreren Modules.

Modules

- reusables, wiederverwendbarer Code, projektübergreifend
- Listen, Sidebars, Navigation
- Unterbereiche der großen Hauptbereiche wie main, article

Styling von Unterbereichen, die kleinteiliger als Layout sind, aber immer noch Gruppierungen.

nav, buttons, Dialogfelder, Bildergalerien, …

Ein Module befindet sich immer in einem Layout und steht nicht für sich allein.

Module sind Projektübergreifend wiederverwendbar.

Vorzugsweise werden Klassen verwendet und Nachfahren

Typ Selektoren vermeiden, wenn die Elemente nicht immer gleich gestylt werden,

außerdem können die Elemente nicht im DOM-Baum verschoben werden, ohne den Selektor zu verändern.

ID Selektoren vermeiden, um keine Spezifitätsprobleme zu bekommen.

Beispiel

```
<nav class=„nav“>

     <ul class=„nav__ul“> //s. BEM-Notation

          <li>

               <a>


.nav__ul a{}
```

State

Layout oder Modules in einem bestimmten State z.B. hidden oder expanded, mobile oder large screen, clicked, active, optional, … success, error

States überschreiben die alpgemeingültigen Regeln

States und Modules sind sich sehr ähnlich, hier gilt es abzuwägen (bei allen anderen Kategorien übrigens auch).

- Einfache Klassenselektoren,
- keine Nachfahren oder ID-Selektoren,
- !important ist erlaubt

Theme

Theme Dateien kommen nicht immer vor, ähnlich zu State, und nur für spezielle Projekte.

4 ITCSS – Inverted Triangle CSS.

Hier werden die CSS-Anweisungen in aufsteigender Sortierung nach Spezifität geordnet.

Von unspezifisch ...

1. Universal Selektor
2. Typ Selektor und Pseudo-Elemente
3. Klassen, Pseudo-Klassen und Attribut Selektor
4. Id Selektor (grundsätzlich sollten id Selektoren vermieden werden.
5. Inline styles
6. !important Anweisungen

... bis spezifisch

Der Code wird in „layer“ aufgeteilt, die durch Dateien oder Verzeichnisse repräsentiert werden können.

1. **Settings**
   fonts-Variablen, colors-Variablen, sonstige Definitionen (Deklarationen)

2. **Tools**
   globale mixins und functions

Erst ab hier wird erster CSS-Code produziert.

3. **Generic**
   reset, normalize oder sanitize
   box-sizing
   Verwendung von Typ Selektoren

4. **Elements**
   Typ-Selektoren, ggf. werden hier browser-styles überschrieben

5. **Objects**
   Klassen Selektoren für wiederkehrende styles.
   Objects fallen oftmals ganz weg.

6. **Components**
   Spezielle Komponenten, die den eigentlichen Style ausmachen.
   Hier finden sich die meisten CSS-Regeln.
   Jede einzelne Komponente bekommt in der Regel eine eigene Datei.
   Klassen Selektoren

7. **Utilities oder Trumps**
   Hilfsklassen, die alles vorherige überschreiben können.
   Hier kann !important verwendet werden

Einzelne Gruppen oder Elemente werden also nicht in einer Datei beschrieben, sondern Elemente können durch alle layer hindurchgereicht werden und so immer spezifischer gestyled werden. Die unteren layer werden oftmals projektübergreifend verwendet.
Ab objects und components werden sie projektspezifischer.

5 OOCSS – Object Oriented CSS

Ziel ist auch hier gute Wartbarkeit und Wiederverwendung einzelner Komponenten.

Als ein Objekt wird in dieser Organisationsform jedes sich möglicherweise wiederholendes visuelles Pattern spezifiziert.

Prinzipien:

- Separation of structure and skin
- Separation of container and content

Separation of structure and skin

Unter structure (layout) fällt alles, was für den Anwender „unsichtbar“ ist, also z.B. Größe und Positionierung.

Properties:
height, width, position, margin, padding, overflow,
grid, flex, etc.

Skin (Gestaltung) ist danach alles für den Anwender sichtbar, also z.B. Farben.

Properties:
colors, fonts, shadow, gradient, etc.

Hat ein Element sowohl structure als auch skin, dann bekommt es zwei Regeln.
Als Selektoren werden hauptsächlich Klassen verwendet.
Eine Klasse kann dann einem Element zugefügt werden und der entsprechende Regelsatz ist so wieder verwendbar.

Separation of container and content

Content sind i.d.R. Elemente, die in anderen Elementen geschachtelt sind, z.B. <img>, <p>, <li>, etc.
Container sind übergeordnete Elemente, die i.d.R. keinen Flow-Content enthalten, sondern weitere Elemente, z.B. <nav>, <main>, <article>, <header>, <footer>, <ul>, etc.
Container haben fast immer eine structure-Klasse.

Für die Klassen können zwar die BEM – Regeln zur Benennung verwendet werden, allerdings wird die Regel nur eine Klasse pro Element gebrochen. Auch OOCSS bläht das HTML-Dokument auf.

https://www.keycdn.com/blog/oocss