CSS Präprozessoren

BACHELORARBEIT 2

StudentIn Barbara Huber, 1010601010
BetreuerIn Hannes Moser

Kuchl, 25.02.2015

Eidesstattliche Erklärung

Hiermit versichere ich, Barbara Huber, geboren am **17.01.1991** in **Kitzbühel**, dass ich die Grundsätze wissenschaftlichen Arbeitens nach bestem Wissen und Gewissen eingehalten habe und die vorliegende Bachelorarbeit von mir selbstständig verfasst wurde. Zur Erstellung wurden von mir keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Ich versichere, dass ich die Bachelorarbeit weder im In- noch Ausland bisher in irgendeiner Form als Prüfungsarbeit vorgelegt habe und dass diese Arbeit mit der den BegutachterInnen vorgelegten Arbeit übereinstimmt.

Kuchl, am **25.02.2015**

Unterschrift

Barbara Huber 1010601010

Kurzfassung

Vor- und Zuname:	Barbara HUBER
Institution:	FH Salzburg
Studiengang:	Bachelor MultiMediaTechnology
Titel der Bachelorarbeit:	Sass vs. Less vs. Stylus
Begutachter:	Hannes Moser

Die folgende Arbeit beschäftigt sich mit der Verwendung von CSS-Präprozessoren, im speziellen mit LESS, Sass und Stylus. Es wird beschrieben, was Präprozessoren im Allgmeinen sind und wie diese angewendet werden. Um die Verwendung der Präprozessoren verständlicher erklären zu können, wird in der Arbeit auch auf den Begriff Workflow eingegangen und erörtert, wie der Workflow mithilfe von Präprozessoren beeinflusst wird.
Um zu verstehen, wie Präprozessoren arbeiten, ist es notwendig zu wissen, was ein Parser ist und was unter Software Komponenten zu verstehen ist. Zwei Kapitel der Arbeit beschäftigen sich mit diesen Themen und erläutern anhand von Beispielen und den Präprozessoren LESS, SCSS bzw. Sass und Stylus, wie die jeweiligen Parser der Präprozessoren im Detail funktionieren. Auch das Kapitel zu Software Komponenten wird in Zusammenhang mit den erwähnten Präprozessoren erklärt.
Wie zu Beginn erwähnt, befasst sich die Arbeit im Speziellen mit den CSS-Präprozessoren LESS, Sass und Stylus.
Nach der Explikation von Präprozessoren, Parser, Workflow und Software Komponenten werden die, der Arbeit zu Grunde liegenden Präprozessoren im Detail beschrieben.
Nach der Theoretischen Darlegung des Themas, wird die praktische Arbeit beschrieben und erläutert, wie diese umgesetzt wurde. Im Anschluss daran, wird im letzten Kapitel der Arbeit diskutiert, welche Ergebnisse der praktische Teil der Arbeit ergeben hat und welche Relevanz das Thema und die Inhalte der Arbeit auf weitere Forschungsfragen bzw. Problemaspekte hat.

Schlagwörter: Präprozessoren, Workflow, Software Komponenten, CSS, LESS, SCSS, Sass, Stylus

Abstract

The topic of this thesis deals with the Usage of CSS-Preprocessors, especially with LESS, Sass and Stylus. The thesis describes the meaning of Preprocessors and the use of these. To explain the usage of preprocessors the thesis explains the meaning of workflow and how it could be improved with preprocessors.
Furthermore the concept of parsers and software components is demonstrated.
The theoretical part of the thesis will characterize preprocessors, parsers, workflow and software components on one hand and the three preprocessors - „LESS“, „Sass“ and „Stylus“ - on the other hand.

The practical part shows the three preprocessors in use: by developing a yeoman generator and a grunt-task functionality and use of each preprocessor will be shown.

Finally the comparisons´ relevance and further or remaining questions for following research in this area are summarized.

Keywords: Preprocessors, Workflow, software components, CSS, LESS, SCSS, Sass, Stylus

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Forschungsfrage 2

1.2 Aufbau 2

2 Was sind CSS-Präprozessoren **4**

2.1 Workflow 10

2.2 Software Komponenten 12

2.3 Superset 14

2.4 Parser 14

3 Erweiterungen von CSS Präprozessoren **18**

3.1 Variablen 18

3.2 Mixin 19

3.3 Funktionen 22

3.4 Vererbung 22

3.5 Logic / Loops 23

4 Präprozessoren LESS, Sass, Stylus **24**

4.1 LESS 24

4.2 Sass 26

4.3 Stylus 28

5 Implementierugen von CSS Präprozessoren **29**

5.1 Variablen 29

5.1.1 Less 29

5.1.2 Sass 29

5.1.3 Stylus 30

5.2 Mixins 31

5.2.1 Less 31

5.2.2 Sass 31

5.2.3 Stylus 32

5.3 Funktionen 33

5.3.1 Less . . . 33
5.3.2 Sass . . . 36
5.3.3 Stylus . . . 36
5.4 Vererbung . . . 37
5.4.1 Less . . . 37
5.4.2 Sass . . . 37
5.4.3 Stylus . . . 38
5.5 Logic/Loops . . . 38
5.5.1 LESS . . . 38
5.5.2 Sass . . . 39
5.5.3 Stylus . . . 40

6 Praktische Anwendung und Vergleiche 42
6.1 Yeoman Generator . . . 42
6.2 Grunt Task . . . 46
6.3 Ergebnisse . . . 50

7 Schluss 52
7.1 Relevanz . . . 53
7.2 Ausblick . . . 53

Abbildungsverzeichnis 55

Tabellenverzeichnis 57

Literaturverzeichnis 58

1 Einleitung

Ein wichtiger Teil von Webseiten, ist die Gestaltung und das Design. Um eine Seite nach dem Design zu gestalten, verwendet man im World Wide Web das CSS 3.
Seit ein paar Jahren, werden sogenannte Präprozessoren für CSS entwickelt, die Erweiterungen zu CSS bereitstellen.
Was sind nun Präprozessoren? Wie werden sie verwendet und welche Erweiterungen stellen Sie im Detail zur Verfügung? Diese Fragen sollen in der Arbeit behandelt werden.
Mit CSS3 ist schon eine große Vielfalt an Gestaltungen möglich. CSS3 bietet einfache Berechnungen und es gibt Funktionen, mit denen Beispielsweise die Ecken einer Box abgerundet werden können oder einem Hintergrund ein Verlauf gegeben wird. Die Notwendigkeit, der Verwendung von Präprozessoren für CSS, wird immer größer, da die Ansprüche an die Gestaltung von Webseiten immer mehr Bedeutung bekommt. Folgende Abbildung zeigt das Suchinteresse nach den Suchbegriffen Less, Sass, Styuls und Css in Google:[1]
Die Statistik in der Abbildung zeigt das Interesse im zeitlichen Verlauf von 2005 bis 2015.

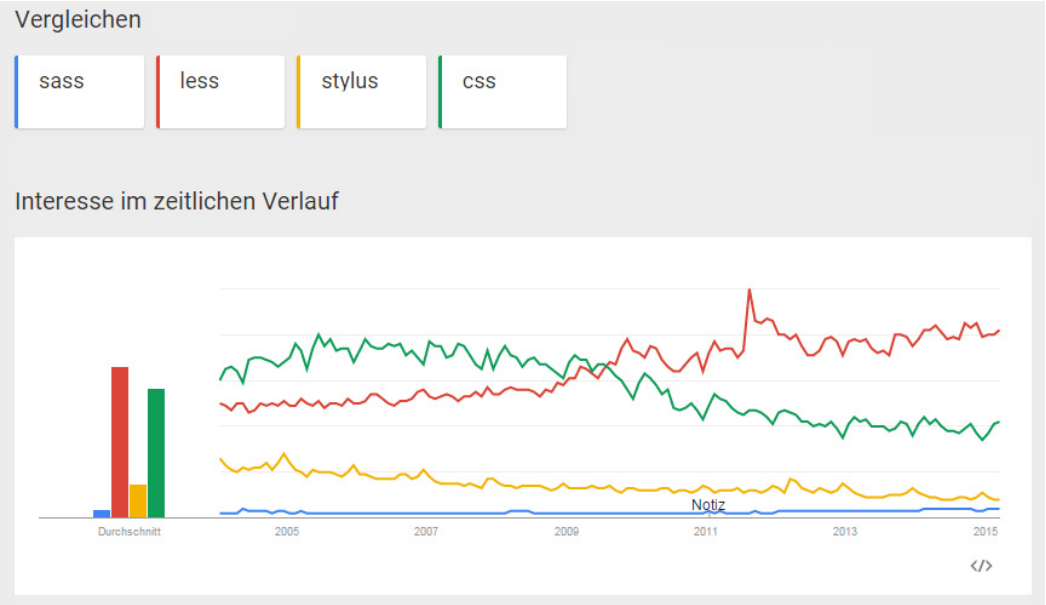

Abbildung 1: Suchtrends in Google nach Sass, Less, Styuls und Css

In der Abbildung ist gut zu erkennen, dass das Interesse nach Less gerade in den letzten 5 Jahren sehr gestiegen ist. Trotzallem wird noch sehr oft nach CSS gesucht, was darauf zurückfzuführen ist, dass, trotz den Vorteilen von CSS-Präprozessoren, noch sehr viele Programmierer/innen mit einfachem CSS3 ihre Seiten stylen.
Wie die Abbildung zeigt, werden im Vergleich zu Less und Css selten Suchanfragen zu Sass und Stylus über Google abgesetzt.

1. Google Trends zum Vergleich von Suchinteresse zu Less, Sass, Stylus und CSS: http://www.google.at/trends/explore#geo=US&q=sass,+less,+stylus,+css&cmpt=q.

Die Statistik die sich über die Abbildung ergibt, spiegelt jedoch nur das Suchinteresse über Google wieder und kann nicht als wissenschaftlich erwiesene Statistik zur Verwendung der gezeigten Programmiersprachen gesehen werden.
Die Abbildung zeigt, dass Less noch vor CSS nachgefragt wird. Less ist einer der bekanntesten CSS-Präprozessoren, der wie Sass und Stylus in dieser Arbeit behandelt wird.
CSS-Präprozessoren bieten viele Erweiterungen, welche die Erstellung eines Webseitenstylings vereinfachen und den Workflow bei der Erstellung verringern. Neben den Erweiterungen der Präprozessoren, gibt es für die CSS-Präprozessoren viele Tasks in Node.js, die dabei helfen, die Verwendung der Präprozessoren mit Hilfe von Grunt oder Gulp noch weiter zu Verbessern.
Der Praktische Teil dieser Arbeit wird mittels solcher Grunttaks realisiert, wie das im Detail umgesetzt wird, wird in Kapitel 6 beschrieben.

1.1 Forschungsfrage

In dieser Arbeit soll die Frage:

> „ Welche Präprozessoren gibt es und wie funktionieren diese? Wie funktionieren Präprozessoren im Generellen? “

beantwortet werden.
Die Relevanz dieser Frage wird im zunehmenden Interesse, Webseiten immer besser und stylischer zu gestalten, gesehen. Die Webseiten sollen immer innovativer und kreativer werden und oft reicht es nicht mehr, mit den von CSS3 gebotenen Möglichkeiten auszukommen.
In dieser Arbeit soll nun, wie schon unter Punkt 1 erläutert, untersucht werden, welche Tools bereitgestellt werden, um CSS3 zu erweitern und den Ansprüchen zu entsprechen.
Am Beginn der Arbeit wird kurz erläutert wie sich Präprozessoren definieren und welchen Anteil am Gesamten die einzelnen Bestandteile, wie z.B. Mixins und Vererbung, haben und wie diese zusammenwirken. Im nächsten Schritt wird auf die einzelnen Methoden eingegangen um im Anschluss, im praktischen Teil der Arbeit, eine Library mit den beschriebenen CSS-Präprozessoren zu erstellen.

1.2 Aufbau

Wie in Punkt 2 beschrieben wird in der Arbeit zuerst auf die Grundthematik Präprozessoren eingegangen und beschrieben, wie Präprozessoren im Gesamten funktionieren und die Bestandteile im Einzelnen verwendet und erstellt werden.
Als Nächstes wird erklärt welche Auswirkungen Präprozessoren auf die Erstellung eines Webseitenstylings haben. Das Kapitel dient zur Information der Leser/innen und um die technischen Weiterfürhungen der Präprozessoren in den weiteren Kapiteln besser verstehen zu können.

Das Unterkapitel 2.1 erläutert kurz den Begriff Workflow, da dieser in der Arbeit des öfteren verwendet wird.
Die weiteren Unterkapitel 2.3 bis 2.5 erläutern im Speziellen die genannten CSS-Präprozessoren Less, Sass und Stylus, nachdem in Kapitel 2.2 noch genauer auf Software Komponenten eingegangen wird.
In Kapitel 3 wird auch kurz auf den Begriff Superset eingegangen.
Nachdem auch die Präprozessoren Less, Sass und Stylus erläutert werden, wird im Weiteren in Kapitel 4 darauf eingegangen, was Parsing ist und wie das im Speziellen bei den erwähnten Präprozessoren funktioniert.
Nach dieser Einführung wird die Benützung und Implementierung der, in Punkt 1 genannten Tools beschrieben.
Kapitel 5 beschreibt die genaue Verwendung der in Kapitel 2.6 beschriebenen Erweiterungen. Nach der Theoretischen Einführung in das Thema, beschreibt Kapitel 6 die Umsetzung und Realisierung des praktischen Teils.
Im letzten Kapitel wird noch einmal auf die Vorteile von CSS-Präprozessoren eingegangen und diese mit dem praktischen Teil der Arbeit verglichen und so herausgearbeitet, wie sich diese Vorteile auf die Arbeit mit CSS-Präprozessoren auswirken.

2 Was sind CSS-Präprozessoren

Präprozessoren sind Computerprogramme, welche Daten vorbereiten und zur Weiterverarbeitung an ein anderes Programm weitergeben. In den meisten Fällen, wird ein Präprozessor dazu verwendet, Eingabedaten, im Falle dieser Arbeit CSS-Styles, zu konvertieren.(Peter 2012)
Präprozessoren werden benützt um beispielsweise Variabilität zu schaffen. Im Falle dieser Arbeit wird Variabilität in Bezug auf die mehrfache Verwendung gleicher Codeabschnitte in einem Stylesheet verstanden. Bspw. die Schriftart oder Schriftfarbe. In dieser Arbeit geht es um die CSS-Präprozessoren Sass, Less und Stylus. Diese Präprozessoren werden eingesetzt, um das Schreiben des Codes sowie die Syntax zu erleichtern. Desweiteren stellen sie Funktionen und Variablen zur Verfügung.
Mithilfe der genannten Präprozessoren können Aufgaben automatisiert werden. Es gibt die Möglichkeit Variablen zu erstellen und so die Bearbeitung von Stylings wie z.B. der Farbe, um ein Vielfaches zu erleichtern. Möchte man in einem CSS file die Farbe der Schrift verändern, muss man alle Stellen suchen, an denen diese Farbe zugeordnet wird. Verwendet man beispielsweise mit less eine Variable für die Farbe, so muss nur an einer Stelle, dort wo die Farbe der Variablen zugewiesen wird, die Farbe geändert werden.
Nicht nur Variablen sondern auch Funktionen und Mixins werden von CSS-Präprozessoren bereit gestellt. Im Gegensatz zu einfachem CSS kann man mit den genannten Präprozessoren Mixins erstellen, die dabei helfen den Code übersichtlicher zu Gestalten. Mit einem Mixin kann z.B. ein Clearfix erstellt werden. Hierfür wird folgender Code, in scss syntax, siehe Kapitel 2.3 für die Definition von SCSS, erstellt:

Listing 1: erstellen eines Mixins (scss)

```
1  @mixin .clearfix {
2   content: ".";
3    display: block;
4    height: 0;
5    clear: both;
6    visibility: hidden;
7  }
8
9  .mainContent {
10   @include: .clearfix;
11 }
```

Die Zeilen 1 bis 7 in Listening 1 erstellen das Mixin *clearfix* und in Zeile 10 wird gezeigt, wie dieses dann aufgerufen werden kann. Mit diesem Mixin kann vermieden werden, dass die in dem Mixin enthaltenen Stylings jedes mal erneut erstellt werden müssen. Stattdessen kann an jeder gewünschten Stelle im Code mit dem Befehl ”include:clearfix"der gesamte Codeabschnitt eingebunden werden.

Ein weiterer Vorteil von CSS-Präprozessoren ist, dass der Code verschachtelt werden kann. Folgendes Listing zeigt, wie sich das verschachteln des Codes auf die Lesbarkeit auswirken kann. Listing 2 zeigt den Code ohne Verschachtelung und Listing 3 mit Verschachtelung:

Listing 2: Code ohne Verschachtelung (css)

```
1  #content .text .title{
2    font-size: 12px;
3    color: #dedede;
4    font-weight: normal;
5  }
6  #content .text .title h1{
7    font-size: 18px;
8    color: #ffffaa;
9    font-weight: bold;
10 }
11 #content .text .title h3{
12   color: #efefef;
13 }
```

Listing 3: Code mit Verschachtelung (scss)

```
1  #content .text{
2    .title{
3      font-size: 12px;
4      color: #dedede;
5      font-weight: normal;
6      h1{
7        font-size: 18px;
8        color: #ffffaa;
9        font-weight: bold;
10     }
11     h3{
12       color: #efefef;
13     }
14   }
15 }
```

Wie in den 2 Listings erkennbar, ist der Sinn von CSS-Präprozessoren nicht immer, den Code zu verkürzen, sondern die Schreibweise und dadurch auch die Lesbarkeit des Codes zu vereinfachen. In Listing 2 wird Zeile 1 in den Zeilen 6 und 11 wiederholt. Dies ist in Listing 3 nicht der Fall. Durch die Verschachtelung können beliebig viele Klassen, die sich innerhalb

der Klasse text befinden, angesprochen werden, ohne dass immer alle darüberliegenden Klassen aufgerufen werden müssen.
Es sollte jedoch darauf geachtet werden, dass es nicht immer sinnvoll ist, den Code zu verschachteln. Hat man z.B. einen Button, der auf der ganzen Seite immer grün sein soll, macht es keinen Sinn, dem Button die Styles in einer Verschachtelung zu geben, da man dann immer darauf achten muss, dass der Button auch wirklich in diesem Abschnitt des Quellcodes auftritt. Stattdessen ist es hier sinnvoller, den Button separat zu stylen und bei etwaigen Änderungen, diese dann in der Verschachtelung anzuführen.

Neben den genannten Erweiterungen von Less, Sass und Stylus zu CSS kann man zusätzlich den Workflow verbessern, wie der Workflow verbessert werden kann, wird in Kapitel 2.1 beschrieben. Werden die Präprozessoren richtig verwendet kann der Code sauberer und auch kürzer gehalten werden.
Durch den Einsatz der Erweiterungen, kann ein komplexer Code vereinfacht werden, wodurch der Code wartbarer und übersichtlicher wird. Auch kann der Zeitaufwand für die Erstellung der Stylesheets verkürzt werden, da Beispielsweise ein Code, der häufig verwendet wird, mithilfe eines Mixins nur einmal definiert werden muss.

Der Code von CSS-Präprozessoren ist immer valides CSS, da die Präprozessoren Erweiterungen von CSS darstellen. Die am Beginn des Kapitels genannten Präprozessoren sind drei der bekanntesten CSS-Präprozessoren, es gibt jedoch noch weitere, wie zum Beispiel Turbine oder Switch CSS (Jung 2010).

Um zu überprüfen, ob das geschriebene Stylesheet valide und von guter Qualität ist, gibt es Linting, dies ist die Überprüfung von CSS auf Validität und Codequalität. Es gibt einige OpenSource Tools, die die Überprüfung des Stylesheets einfach und schnell ermöglichen. Ein gutes OpenSource Tool ist z.B. CSSLint von Nicolas C. Zakas und Nicole Sullivan[2]
Neben der Überprüfung der Codequalität werden mit Linting auch die Browserperformance und viele weitere Punkte wie z.B.

- Parsing Fehler
- leere Anweisungen
- Nullwerte benötigen keine Einheiten
- keine IDs in Selektoren
- nicht zu viele Floats verwenden
- nicht zu viele Schriften verwenden

gecheckt.

Wie zuvor beschrieben, bieten CSS-Präprozessoren einige Vorteile, können jedoch auch

2. open source Tool zur Überprüfung von CSS: https://github.com/CSSLint/csslint/wiki/About.

Nachteile haben, wenn sie falsch verwendet werden. Diese Arbeit vergleicht die 3 bekanntesten CSS-Präprozessoren, jedoch nicht mit dem Ziel herauszufinden, welche Variante die Beste ist, da es keinen “besten Präprozessor„ gibt.
Welchen Präprozessor man verwendet bzw. benötigt liegt im Interesse jedes einzelnen. Hier einige Meinungen, von Programmierern, die mit Less, Sass bzw. Scss oder Stylus arbeiten.

> “Less, because it is intuitive and also is the engine inside Twitter Bootstrap. So if you want to edit Bootstrap CSS you edit using Less."(Stephanie Hughes, zitiert nach psdtowp 2014)

> “I use LESS as a CSS preprocessor. ... I like it because it’s closest to vanilla CSS. This way, if you find yourself in a situation where you have to fix something in pure CSS, you haven’t forgotten how to do so, by working in LESS.
>
> In general, I’m a big fan of preprocessors. They allow you to programmatically style your site or app, and they make writing clean, DRY, Object-Oriented CSS much easier."(Jamie Marcus, zitiert nach psdtowp 2014)

> “I have surrendered to SCSS . . . it has the best tooling support (including Chrome) and the most devs using it. I *really* like the others too and have zero qualms about them."(Brian Leroux, zitiert nach psdtowp 2014)

> “I use Scss because most projects use it and Stylus on my own projects. I’m really fond of the stylus syntax, the fact that it makes my stack 100% JavaScript and find that it’s by far the fastest compiler."(Wes Bos, zitiert nach psdtowp 2014)

Aus den Zitaten geht klar hervor, dass auf Grund vieler Vorlieben und Beweggründe ein bestimmter CSS-Präprozessor gewählt wird.
Die Wahl des CSS-Präprozessors hängt auch von der eigenen Arbeitsweise und den Anforderungen durch die Projekte an den CSS-Präprozessor selbst ab.

Um das Schreiben eines Stylesheets noch weiter zu vereinfachen, gibt es nicht nur die genannten Präprozessoren, sondern auch Postprozessoren wie „Autoprefixer“. Ein Autoprefixer erlaubt es, im Stylesheet auf die prefixes für die verschiedenen Browser zu verzichten und diese dann mithilfe des Autoprefixers automatisch hinzuzufügen. Ein Autoprefixer verwendet eine Datenbank mit allen gängigen browsern und fügt alle prefixes für diese Browser automaitsch hinzu. Nachfolgender Code zeigt ein Beispiel, wie ein Autoprefixer funktioniert.

Listing 4: Verwendung Autoprefixer (css)

```
1 //css code without prefixes
2 a {
```

```
3   transition: transform 1s
4 }
5
6 //css code after autoprefixer
7 a {
8   -webkit-transition: -webkit-transform 1s;
9   transition: -ms-transform 1s;
10  transition: transform 1s
11 }
```

Ein Autoprefixer kann mithilfe von Grunt, welches in dieser Arbeit im praktischen Teil verwendet wird, installiert und auf das geschriebene Stylesheet angewendet werden. Ein Autoprefixer kann sowohl auf css-code als auch auf scss, less oder stylus Code angewendet werden.
Um den Autoprefixer in grunt zu verwenden, wird folgender Code in das Gruntfile.js geschrieben:(Sitnik 2013)

Listing 5: Code in Gruntfile für Autoprefixer

```
1 autoprefixer: {
2   dist: {
3     files: {
4       'build/style.css': 'style.css'
5     }
6   }
7 },
8 watch: {
9   styles: {
10    files: ['style.css'],
11    tasks: ['autoprefixer']
12  }
13 }
14 ...
15 grunt.loadNpmTasks('grunt-autoprefixer');
```

Wie beschrieben, kann der Autoprefixer mit Grunt verwendet werden. Grunt ist ein Tool, mithilfe dessen, Tasks automatisch, also ohne zutun des Benutzers, der Benutzerin ablaufen. Grunt wird mit npm, dem node package manager, installiert.
Die Installation erfolgt über die Kommandozeile. Bevor Grunt installiert werden kann, muss jedoch npm installiert sein.
Um die Verwendung von grunt zu erleichtern, benötigt man die Kommandozeilenschnittstelle von Grunt. Wird diese Schnittstelle verwendet, kann grunt von jedem Ordner aus aufgerufen werden. Es ist aber auch möglich grunt von der Shell, die von Windows zur Verfügung gestellt wird, aufzurufen.

Um Grunt zu verwenden, benötigt man eine package.json und ein Gruntfile.js.
Ein Beispiel für eine package.json und ein Gruntfile.js:

Listing 6: Beispiel package.json

```
{
  "name": "my-project-name",
  "version": "0.1.0",
  "devDependencies": {
    "grunt": "~0.4.5",
    "grunt-contrib-jshint": "~0.10.0",
    "grunt-contrib-nodeunit": "~0.4.1",
    "grunt-contrib-uglify": "~0.5.0"
  }
}
```

Im Abschnitt der „devDependencies“ wird angegeben, welche grunt-tasks installiert sein müssen.

Listing 7: Beispiel Gruntfile.js

```
module.exports = function(grunt) {

  // Project configuration.
  grunt.initConfig({
    pkg: grunt.file.readJSON('package.json'),
    uglify: {
      options: {
        banner: '/*! <%= pkg.name %> <%= grunt.template.today
           ("yyyy-mm-dd") %> */\n'
      },
      build: {
        src: 'src/<%= pkg.name %>.js',
        dest: 'build/<%= pkg.name %>.min.js'
      }
    }
  });

  // Load the plugin that provides the "uglify" task.
  grunt.loadNpmTasks('grunt-contrib-uglify');

  // Default task(s).
  grunt.registerTask('default', ['uglify']);

};
```

Nachdem grunt installiert wurde und die package.json und das Gruntfile.js richtig befüllt wurden, müssen die packages, welche in der package.json unter „devDependencies“ angegeben wurden, installiert werden.
Hat man das CLI, also die Kommandozeilenschnittstelle, von Grunt installiert, werden alle benötigten packages automatisch beim aufruf von grunt installiert, da das CLI bei jedem Aufruf von grunt überprüft, ob nicht installierte packages benötigt werden.
Verwendet man die normale windows shell, muss man vor dem grunt-Aufruf noch ‚npm install‘ ausführen. Mit diesem Aufruf von npm werden alle benötigten Packages mit einem Durchlauf installiert.
Im praktischen Teil der Arbeit wird grunt verwendet und das Kapitel 6 beschreibt die Verwendung von Grunt noch genauer.
Das gelesene Kapitel beschreibt den Nutzen und die Verwendung von CSS-Präprozessoren. Es soll erläutern, welche Vorteile und Nachteile so ein Präprozessor bietet. Im nächsten Kapitel wird der Begriff Workflow genauer definiert.

2.1 Workflow

Was ist Workflow? Wie wird Workflow definiert?
Diese Fragen werden in folgendem Kapitel erörtert und beantwortet.

> "Workflow ist ein Arbeitsablauf... Mit dem Workflow können Geschäftsprozesse, an denen Mitarbeiter aus mehreren Abteilungen in einer vorgegebenen Reihenfolge beschäftigt sind, informationstechnisch realisiert werden. Die einzelnen Aktivitäten eines Workflows stehen in einem logischen Zusammenhang und einer zeitlichen Abfolge zueinander. "(ITWissen.info 2014)

Das Zitat beschreibt, dass ein Workflow eine Abfolge von Aktivitäten ist. In Zusammenhang mit dieser Arbeit, bedeudet Workflow die Ausführung verschiedener Arbeitsschritte in einer definierten Abfolge.
Der erste Arbeitsschritt bei der Erstellung eines Webseitenstylings ist die Erstellung eines Stylesheets. In dieser Arbeit werden less, sass und stylus behandelt. Es wird also entweder eine style.less, eine style.sass bzw. style.scss oder eine style.styl erstellt.
Durch die Verwendung von Präprozessoren, kann schon bei der Erstellung des Stylesheets der Workflow vereinfacht und verbessert werden, da durch Erweiterungen wie bspw. Variablen und Mixins, das schreiben des Codes erleichtert wird.
Neben der erwähnten Erweiterungen kann der Workflow zusätzlich noch durch die Verwendung von Molekülen verbessert werden.

Unter einem Molekül wird, wie auch in der Chemie, eine Zusammensetzung von Atomen verstanden.
In Bezug auf CSS sind Atome einzelne Stylings wie bspw. ein Inputfeld oder ein Submitbutton. Durch die Zusammensetzung der Atome kann ein Molekül, bei diesem Beispiel ein

Formular, erstellt werden.(Frost 2013)
Mithilfe der Moleküle, können Abschnitte von Stylings einmal für die gesamte Seite erstellt werden, und müssen so nicht wiederholt definiert werden.

Um diese Styles richtig auf der Webseite ausgeben zu können, folgt der nächste Arbeitsschritt. Die Kompilierung der Dateien in CSS.
Für die Kompilierung gibt es für jeden der genannten CSS-Präprozessoren einige Third-PartyLibraries wie z.B. Koala für Sass und Less oder CodeKit, welche less, stylus und sass kompiliert.
Wie im vorherigen Absatz erwähnt, kann durch die Verwendung der Erweiterungen der Workflow verbessert werden, da der erste Arbeitsschritt schneller und verbessert durchgeführt werden kann.
Die 2 genannten Libraries Koala und CodeKit ermöglichen es, die erstellten Stylesheets in Echtzeit zu kompilieren, so kann der Workflow ein weiteres mal verbessert werden, da die Erstellung des Stylesheets und dessen Kompilierung parallel ablaufen.

Bei der Verwendung von CSS-Präprozessoren, wird in den meisten Fällen, wie auch bei der Benützung von CSS, nur die erstellte css Datei im HTML-Code eingebunden. Um die Stylings zu verändern, muss bspw. bei Stylus, die Stylesheetdatei .styl bearbeitet und erneut kompiliert werden um die Änderungen zu sehen.
Um direkt in den Developer Tools, das CSS bearbeiten und ändern zu können, wird von den Präprozessoren die Möglichkeit für CSS-SourceMaps bereitgestellt. Sogenannte SourceMap-Dateien, sind json-Dateien, welche ein Mapping zwischen den kompilierten CSS Deklarationen und dem Sourcefile, also der ursprünglichen Stylesheetdatei, definiert. Durch diese Datei wird es ermöglicht, dass im Developer Tool von Chrome direkt der CSS Code verändert werden kann. Um die SourceMap-Datei im Chrome verwenden zu können, muss die Verwendung in den Developer Tools aktiviert werden. Nach der Aktivierung, wird die generierte CSS-Datei bei jeder Änderung in den Developer Tools automatisch neu geladen. Durch die Verwendung von SourceMaps kann der Workflow noch einmal verbessert werden, da nicht für jede Änderung die Quelldatei bearbeitet werden muss.(*Working with CSS Preprocessors* 2013)

Nachdem in den vorangegangenen Absätzen die Vor- und Nachteile von Präprozessoren erläutert wurden, wird im nächsten Kapitel genauer auf den Begriff „Software Komponenten“ eingegangen.

2.2 Software Komponenten

Software Komponenten (im weiteren Verlauf als Komponenten bezeichnet) sind wiederverwendbare Bausteine einer Applikation, welche aus Software-Code bestehen. Komponenten implementieren spezifische Funktionalitäten gemeinsam mit vordefinierten Schnittstellen. Da es sich um wiederverwendbare Teile eines Codes handelt, sind Komponenten sogenannte Software-Bausteine, die einen bestimmten Bereich eines Geschäftsfeldes kapseln, jedoch keine abgeschlossene Applikation darstellen, und nicht für sich alleine ablaufen können. (Andresen 2003, 1)
In Bezug auf diese Arbeit, sind bspw. Mixins Komponenten von CSS-Präprozessoren. Diese Komponente kann wiederverwendet werden und bildet einen eigenschändigen Bereich des CSS-Präprozessors. Wie beschrieben, ist sie jedoch keine abgeschlossene Applikation und kann aus diesem Grund nicht alleine Ablaufen.

Im Buch von Andreas Andresen, Komponentenbasierte Softwareentwicklung, (2) werden auf Seite 2 einige Vorteile von Komponenten aufgezählt. Die wichtigsten für diese Arbeit sind folgende:

1. überschaubare Größenordnung
2. trennen Zuständigkeiten
3. einfach einsetzbar und kombinierbar
4. einfach wiederverwendbar
5. fördern eine schnelle Applikationsentwicklung
6. einfach austauschbar

Dieselben Vorteile haben teilweise auch die Erweiterungen von CSS-Präprozessoren. Beispielsweise Mixins. Mixins können, wie schon erwähnt, im weitesten Sinne als Software Komponenten bezeichnet werden, da die Punkte aus der Aufzählung genau auf Mixins übertragen werden können. Mixins haben eine überschaubar Größenordnung, da sie meist nur eine Aufgabe erfüllen, wie z.B. einen Hintergrund mit Farbverlauf oder eine CSS-Transition zu erstellen.
Ein Mixin erfüllt auch die Punkte 3, 4 und 6. Ein Mixin kann an jeder Stelle des Stylesheets verwendet werden und ist austauschbar, da einfach ein anderes Mixin verwendet werden kann. Ebenso kann es kombiniert werden, in dem 2 Mixins beim selben Selector aufgerufen werden.
Somit hat ein Mixin alle bereits aufgezählten Vorteile einer Software Komponente. Jedocch gibt es auch Anforderungen an Komponenten (6). Folgende Aufzählung zeigt einige Anforderungen an die Architektur:

1. die Ermöglichung einer einfachen Kommunikation von Stytem und Komponente untereinander

2. komplexe Zusammenhänge innerhalb von Stystemen und zwischen Komponenten und Stystemen müssen auf übersichtliche und einfache Weise strukturiert werden können.
3. Komponenten und Stysteme müssen effizient dimensioniert werden
4. bestehende Komponentnen müssen einfach integrierbar sein
5. eine einfache Wiederverwendung der Komponenten muss ermöglicht werden
6. die Zuständigkeit muss klar getrennt sein.
7. das System muss die Anforderungen in Bezug auf Robustheit, Zuverlässigkeit, Performance, Sicherheit und Skalierbarkeit erfüllen.

Bei den Anforderungen an die Architektur, kann ein Mixin nicht alle Punkte erfüllen. Die Punkte 4, 5, 6 können von Mixins erfüllt werden, da diese auch die größten Vorteile eines Mixins darstellen. Bei den anderen Punkten kann dies jedoch nicht ganz deutlich definiert werden.
Ein Mixin als eine Erweiterung eines CSS-Präprozessors, muss diese Punkte nicht erfüllen, es gibt keine direkten Anforderungen an die Erweiterungen in diesem Sinne.
Diese Kriterien legen nahe, dass ein Mixin nur im weiteren Sinne als eine Komponente bezeichnet werden können. Auch die anderen Erweiterungen von CSS-Präprozessoren können, wenn dann nur im weiteren Sinne als solche bezeichnet werden.

Nach dieser allgemeinen Einführung in CSS-Präprozessoren und der Erläuterung von Workflow und Software Komponenten, wird in den nächsten Kapiteln näher auf die, für diese Arbeit verwendeten, Präprozessoren im Speziellen eingegangen.

2.3 Superset

Ein Superset ist eine sogenannte Obermenge. Der Begriff Superset kommt aus der Mengenlehre und bedeutet, dass B ein Superset von A ist, sobald A in B enthalten ist. Bezogen auf diese Arbeit heißt das, dass Beispielsweise Scss, eine Syntax-Variante von Sass, ein Superset von CSS3 ist.
Sobald das CSS3 valide ist, hat man auch ein valides, also gültiges SCSS. Umgekeht gilt das jedoch nicht. Vorteilhaft ist, dass ein schon vorhandener CSS3-Code einfach weiterverwendet werden kann und mit SCSS erweiterbar ist. Um also eine vorhandene CSS3-Datei in SCSS umzuwandeln, muss nur die Dateiendung von .css auf .scss umgeschrieben werden.

Von den, in dieser Arbeit genannten, Programmiersprachen Less, Sass und Stylus ist nur die Syntax-Variante SCSS von Sass bzw. Less jeweils ein Superset von CSS.
Der Begriff Superset darf nicht mit dem Begriff Erweiterung verwechselt werden.
Bei einer Obermenge ist dessen Untermenge immer Valide für die Obermenge, das heißt bei einem Superset wie SCSS ist das CSS, die Untermenge, immer valides SCSS.
Bei den Erweiterungen Sass und Stylus gilt dies jedoch nicht, da die Syntax dieser Präprozessoren nicht dieselbe von CSS3 ist. Auf die Syntax und die Unterschiede wird in den Kapiteln 4.1, 4.2 und 4.3 näher eingegangen.

Nach den Erläuterungen zu Less, Sass, Stylus und den Erweiterungen in den letzten Kapiteln, wird im nächsten Kapitel auf das Parsing im Allgemeinen und im Speziellen von Less, Sass und Stylus eingegangen.

2.4 Parser

Ein Parser ist ein Programm zur Zerlegung und Umwandlung einer beliebigen Eingabe, welche zur Weiterverarbeitung in ein brauchbares Fromat umgewandelt wird. Ein Parser erzeugt zusätzliche Strukturbeschreibungen und bedeutet Syntaxanalyse.
Der Parser verwendet zur Analyse eines Textes einen lexikalischen Scanner, auch Lexer genannt. Ein Lexer zerlegt die Eingabe in sogenannte Tokens (beispielsweise Wörter oder Eingabesymbole) die der Parser versteht.
Bei einem HTML Code würde der Lexer die Datei in HTML-Tags und Fließtext zerteilen und so an den Parser weiterleiten. Den Lexer interessiert nur das Aussehen der Syntaxelemente wie z.b. die spitzen Klammern eines Tags, während der Parser dann die syntaktischen Zusammenhänge verarbeitet. Er untersucht also, welche Paare von Tags zusammengehören oder wie diese verschachtelt sind. Die inhaltliche Bedeutung der Tags interessiert den Parser nicht, dafür ist dann die Weiterverarbeitung zuständig.

Im Falle dieser Arbeit würde also der Lexer die Datei welche in Less, Stylus oder Sass bzw. Scss geschrieben wurde, zerlegen. In diesem Fall sind also die Klammern und Strichpunkte bzw. wie in Sass die Abstände und Einrückungen für den Lexer von Bedeutung.

Der Parser baut eine Datenstruktur bsbw. einen Parsingbaum oder einen Syntaxbaum

und überprüft die Korrektheit der Syntax im Prozess. Er kann entweder selbst oder mit einem Generator programmiert werden. Der Input eines Parsers, kann sowohl Text in einer Programmiersprache als auch normaler Text sein. Eine wichtige Klasse, eines einfachen Parsers ist die „ Regular Expression “.

Wie ein Parser verwendet wird, hängt vom Input ab. Bei einer Markup Sprache wie HTML oder XML wird der Parser zum Lesen der Datein verwendet, bei Programmierpsrachen wie C++, ist ein Parser eine Komponente eines Kompilierers oder Interpreters, welcher den Source Code des Programmes parsed.

Jedes Format, welches geparsed werden kann, muss eine deterministische Grammatik, also eine Grammatik, welche aus Syntaxregeln und Vokabeln besteht, haben. Diese Art von Grammatik wird als „ Kontextfreie Grammatik “ bezeichnet. Die menschliche Sprache hat keine deterministische Grammatik und kann somit nicht so einfach mit einem konventionellen Parser geparsed werden. (Garsiel 2015)

Als eine kontextfreie Grammatik, wie im vorigen Abschnitt beschrieben, wird eine Grammatik bezeichnet, welche Ersetzungsregeln enthält, die nur genau ein Nichtterminalsymbol enhält. Nichtterminalsymbol bedeutet, dass ein Symbol nicht in den endgültigen Wörtern vorkommt. Nichtterminalsymbole kommen nur in Zwischenschritten vor und werden durch das Anwenden von Regeln nach und nach ersetzt, bis nur noch Terminalsymbole vorhanden sind.

Es gibt 2 Hauptarten von Parsern: den „Top down “ und den „ Bottom Up “ Parser. Der Top Down parser geht von Oben nach Unten, in diesem Fall ist das Level der Struktur in der Syntax gemeint. Bottom up geht, wie der Name schon sagt, von Unten nach Oben, also vom niedrigsten Level zum Höchsten.
Ein einfaches Beispiel für einen Parser ist das parsen folgender mathematischer Rechnung: 2+3-1
Der Top Down Parser beginnt bei 2+3 als Expression und geht weiter zu 2+3-1 als Expression.
Der Bottom Up Parser scannt den Input bis eine Regel stimmt und ersetzt den Input mit der Regel. Dieses Vorgehen wird wiederholt, bis kein Input mehr vorhanden ist. Tabelle 1 zeigt, wie der Stack des Parsers aussieht: (Garsiel 2015)

Der Top Down Parser wird auch als LL Parser bezeichnet, welcher von Links nach rechts parst und eine Linksableitung der Eingabe bildet. Bei einer Linksableitung wird das am weitesten Links stehende Nichtterminalsymbol durch Anwendung einer Regel ersetzt. Im Gegensatz dazu, gibt es noch den LR Parser, welcher eine Rechtsableitung bildet und gleich dem Bottom Up Parser ist.
Neben dem LL Parser, gibt es noch den LL(k) parser welcher beim parsen des Inputs mehrere Tokens vorausschaut.

Nachdem im bisherigen Kapitel der Begriff Parser etwas näher betrachtet wurde, wird im

Stack	Input
	2+3-1
term	+3-1
term operation	3-1
expression	-1
expression operation	1
expression	

Tabelle 1: Parser Stack: Bottom Up Parser

Folgenden auf die unterschiedlichen Parser der beschriebenen CSS-Präprozessoren näher eingegangen.

Less
Im letzten Absatz wurde gemeinsam zum Parser vom Tokenizer oder auch Lexer gesprochen. Less zeigt, dass der Tokenizer nicht zwingend verwendet werden muss. Less verwendet beim Parsen keinen Tokenizer, also keinen Lexer.
Der Parser von Less durchläuft die Eingabe, den Less-Code, einmal und parsed alles. Das heißt es gibt eine Funktion in der für alle Spezialfälle wieder eigene Funktionen geschrieben werden.
Bei Less werden nur jene Werte geparst, die eine Variable, Operationen oder dynamische referenzen aufweisen, wohingegen andere Werte übersprungen werden.
Ein Beispiel für einen Wert, der übersprungen werden kann, ist: '1px solid #000'.
Dieser Wert sieht in CSS gleich aus und muss somit nicht verändert werden.
Beinhaltet der Wert eine Variable wie bspw. @color statt #000, muss geparst werden.
CSS kennt keine Variablen, somit muss an jene Stelle im Parser gesprungen werden, an der die Variable geparst wird.
An dieser Stelle, wird dann mit einer regular expression gesucht, welche Variable enthalten ist, um anschließend diese Variable und den dazugehörigen Wert zu finden.

Wie beschrieben, ist der Hauptparser von Less lediglich für die Delegation zuständig. Das heißt der Hauptparser überprüft, welche Funktion, Variable oder Operation im Selektor vorkommt und delegiert die Aufgabe dann an die jeweilige Parsingfunktion weiter. (Sellier 2015b)

Sass/Scss
Sass bzw. Scss ist wie in Kapitel 4.2 beschrieben, in Ruby geschrieben. Somit ist auch der Parser in Ruby geschrieben.
Der Parser von Sass ist ähnlich dem von Less. Es gibt eine Funktion, in der die möglichen Aufrufe im Stylesheet initialisiert werden, bspw. für ein Mixin. Anschließend werden alle unterschiedlichen Aufrufe durchgegangen und anschließend werden die Tokens mit einem Lexer gescannt und umgewandelt. Der Unterschied zwischen den Parsern von Sass und

Scss liegt vor allem im Lexer, da die Syntax unterschiedlich ist und so verschiedene Aufrufe divergente Tokens enthalten.
Der Parser ist für Sass und Scss gleich aufgebaut, allerdings gibt es zwei verschiedene Lexer, die in getrennten Dateien definiert werden. (Yard 2014a)

Stylus
Im Gegensatz zu Less verwendet Stylus, wie auch Sass einen Lexer. Der Parser von Stylus ist so aufgbaut, dass zu Begin alle möglichen Selektoren in ein Array gespeichert werden. Anschließend wird der Parser mit den mitgegebenen Optionen und Strings sowie der Lexer initiasiliert.
Im weiteren Verlauf des Parsers, wird mit 2 Funktionen der letzte und aktuelle Stand des Parsers ermittelt und der Input geparsed.
Der Parser von Stylus ist ein LL(k) Parser, dieser wurde schon zu Begin des aktuellen Kapitels erklärt. Bei Stylus gibt es beim Parser eine Funktion, welche den nächsten Token zurückgibt. Die darauffolgende Funktion betrachtet den Input mit einem Lookahead(1), das heißt er schaut um 1 Token voraus. Wird bei dieser Funktion kein Error ausgegeben, geht der Parser weiter und überprüft den Input mit Lookahead(n).
Die nächsten Funktionen des Parsers überprüfen, ob das Token ein Selector oder ein pseudo Selector, welcher zu Beginn des Parsers jeweils in ein Array gespeichert wurden, ist. Wenn das der Fall ist, folgen Überprüfungen auf Validiät und um welche Art von Selector es sich beim Token handelt. Der Parser überprüft an dieser Stelle auch, ob es sich um eine Funktion, ein Leerzeichen oder ähnliches handelt und führt im weiteren die dementsprechenden Funktionen aus. Bei einem Leerzeichen bspw. wird dieses Übersprungen und weitergegangen.
Bei Stylus sind auch if-statements möglich, somit überprüft der Parser auch, ob solche im Input enthalten sind und bearbeitet diese. (LearnBoost 2010a)

3 Erweiterungen von CSS Präprozessoren

In diesem Kapitel wird auf Variablen, Mixins, Funktionen und Vererbung von Css-Präprozessoren eingegangen. Es wird erklärt wie diese Komponenten generell verwendet werden. Bei den einzelnen Codebeispielen wird angegeben mit welchem Css-Präprozessor dieser umgesetzt wurde. In Kapitel 5 wird die Verwendung jeder Komponente für jeden CSS-Präprozessor im Detail beschrieben.

3.1 Variablen

Variablen, sind eine Erweiterung der CSS-Präprozessoren gegenüber CSS3 und sind gänzlich gleich wie Variablen in anderen Programmiersprachen. Bei einem großen Projekt mit einigen Grundfarben oder Schriftarten, welche an vielen Stellen im Code verwendet werden, wird somit ermöglicht am Beginn der Datei oder in einer separaten Datei die Variable zu erstellen und ihr einen Wert zuzuweisen. Somit muss in der restlichen Datei nur die Variable aufgerufen werden und bei einer Änderung muss diese nur bei der Zuweisung der Variable geschehen und nicht, wie bei CSS, an allen Stellen, wo diese verwendet wird.(Yard 2014b)
Im Zusammenhang der Erstellung einer Variablen in einer eigenen Datei, ist die Variable **@import**, welche es ermöglicht, in der Entwicklung so viele Datein zu haben, wie man möchte und diese dann in der Produktion zu einer einzigen Datei zusammen zu fügen, sehr interessant(Giraudel 2014b).
In der Entwicklung ist es sehr hilfreich die CSS-Datein aufzuteilen, um einen guten Überblick zu schaffen und auch die Größe der Datei beschränken zu können.

> “Multiple files in dev, a single file in prod."(Bruce Lee, zitiert nach Giraudel 2014a)

Nachfolgene Listings zeigen eine Datei für die Variablen und eine Datei, in der die Variablen verwendet werden. Die Listings werden in scss Syntax geschrieben.

Listing 8: variables.scss

```
1 $primaryFont: normal 13px 'condensed light';
2 $primaryColor: #efefef;
```

Listing 9: style.scss

```
1 @import 'variables.scss';
2
3 #content .text{
4   .title{
5     font: $primaryFont;
6     color: $primaryColor;
```

```
7   }
8 }
```

3.2 Mixin

Ein Mixin ist eine Classe in CSS, welche viel Ähnlichkeit mit einer Funktion in einer anderen Programmiersprach, z.B. PHP, hat.
In dieser Verwendung ist ein Mixin eine Gruppe von CSS Anweisungen in einer Klasse. Mixins erlauben es, sämtliche Eigenschaften der erstellten Klasse in einer anderen Klasse aufzurufen.
Beispielsweise hat man ein Mixin mit dem Namen RoundBorders, welches die Klasse .RoundBorders erstellt. Diese Klasse .RoundBorders kann man nun ganze einfach in einer anderen Klasse oder auch einer id, z.B. #menu, aufrufen.(Gerchev 2012)
Folgender Code veranschaulicht das aufrufen eines Mixins (alle Folgenden Codebeispiele zum Mixin sind in Less geschrieben):

Listing 10: Mixin

```
1  //Mixin RoundBorders
2  .RoundBorders {
3    border-radius: 5px;
4    -moz-border-radius: 5px;
5    -webkit-border-radius: 5px;
6  }
7
8  #menu {
9    color: gray;
10   .RoundBorders;
11 }
```

(Gerchev 2012) In Listing 10 wird in den Zeilen 2 bis 6 das Mixin RoundBorders erstellt. In Zeile 9 wird dieses Mixin aufgerufen. Somit erhält die ID menu die Eigenschaften aus der Klasse RoundBorders. Die Ausgabe von Listing 10 wird in Listing 11 dargestellt:

Listing 11: Mixin Ausgabe

```
1  //Mixin RoundBorders
2  .RoundBorders {
3    border-radius: 5px;
4    -moz-border-radius: 5px;
5    -webkit-border-radius: 5px;
6  }
7
8  #menu {
```

```
9     color: gray;
10    border-radius: 5px;
11    -moz-border-radius: 5px;
12    -webkit-border-radius: 5px;
13  }
```

(Gerchev 2012)

Wenn das Mixin in der Ausgabe nicht angezeigt werden soll, kann man dies mit Klammern bewerkstelligen, wie in folgender Listing, in Zeile 2, gezeigt.(the core less team 2014)

Listing 12: Mixin und Ausgabe ohne Mixin

```
1  //Mixin RoundBorders
2  .RoundBorders() {
3    border-radius: 5px;
4    -moz-border-radius: 5px;
5    -webkit-border-radius: 5px;
6  }
7
8  #menu{
9    color: gray;
10   .RoundBorders;
11 }
12
13 //Ausgabe:
14 #menu {
15   color: gray;
16   border-radius: 5px;
17   -moz-border-radius: 5px;
18   -webkit-border-radius: 5px;
19 }
```

Das erstellte Mixin kann somit im gesamten Code verwendet werden. Werden die im Mixin festegelegten Anweisungen in einem Projekt oft benötigt, können damit viele Codezeilen und vorallem Codeduplikationen vermieden werden.

Mixins können auch Argumente oder Selectoren beinhalten.
Soll beispielsweise in einem Mixin für einen abgerundeten Rahmen der Radius variabel bleiben, kann dieser als Argument übergeben werden. Der Code für das Mixin und für die Einbindung in eine Klasse sieht in Less folgendermaßen aus:

Listing 13: Mixin mit Argument

```
1  //Mixin
2  .border-radius(@radius) {
```

```
3     -webkit-border-radius: @radius;
4        -moz-border-radius: @radius;
5             border-radius: @radius;
6  }
7  .button {
8    .border-radius(6px);
9  }
```

(Gerchev 2012) Wie in Listing 13 zu sehen, wird in Zeile 9 der Radius von 6px mitübergeben. (the core less team 2014)
Eine besondere Variable im Zusammenhang mit Mixins ist @arguments. Mit dieser Variable werden alle Argumente, die dem Mixin mitgegeben werden, angewendet. (Gerchev 2012)

Listing 14: Mixin mit @arguments

```
1  //Mixin
2  .BoxShadow(@x: 0, @y: 0, @blur: 1px, @color: #000) {
3    box-shadow: @arguments;
4    -moz-box-shadow: @arguments;
5    -webkit-box-shadow: @arguments;
6  }
7
8  .BoxShadow(2px, 5px);
9
10 //Ausgabe
11 box-shadow: 2px 5px 1px #000;
12 -moz-box-shadow: 2px 5px 1px #000;
13 -webkit-box-shadow: 2px 5px 1px #000;
```

(Gerchev 2012) Wie schon erwähnt können Mixins auch Selectoren beinhalten. Das heißt, es kann in einem Mixin auch ein hover Effekt oder ein Aktivstatus angegeben werden.

Listing 15: Mixin mit Selector

```
1  //Mixin
2  .my-hover-mixin() {
3    &:hover {
4      border: 1px solid red;
5    }
6  }
7  button {
8    .my-hover-mixin();
9  }
10
11 //Ausgabe
```

```
12 button:hover {
13   border: 1px solid red;
14 }
```

(Gerchev 2012) Mixins können bei jedem der, in dieser Arbeit vorgestellten, CSS-Präprozessoren verwendet werden, die Schreibweise unterscheidet sich jedoch. Darauf wird in Kapitel 5.2 noch genauer eingegangen. Erwähnenswert ist auch noch, dass beispielsweise in Less eine Schleife mit einem Mixin gelöst wird. Sass bzw. Scss und Stylus hingegen erlauben das Iterieren durch eine Schleife.

3.3 Funktionen

Funktionen bei CSS-Präprozessoren sind denen in anderen Programmiersprachen sehr ähnlich. Mit Funktionen, kann man beispielsweise zwei Pixelwerte addieren, subtrahieren, dividieren und multiplizieren. Wie die Implementierung der Funktionen bei den, in dieser Arbeit verwendeten, CSS-Präprozessoren funktioniert, wird in Kapitel 5.3 erklärt.

3.4 Vererbung

Verberbung bedeuted, dass die Eigenschaften einer Klasse in einer anderen Klasse vererbt werden können. Im Falle dieser Arbeit hat zum Beispiel die Klasse .message dieselben Eigenschaften wie die Klasse .warning mit ein paar zusätzlichen Eigenschaften. Durch die Möglichkeit der Vererbung bei CSS-Präprozessoren kann nun vermieden werden den Code zu duplizieren. Folgender Code zeigt, wie mit der Variable extend eine Klasse innerhalb einer anderen Klasse aufgerufen und so deren Eigenschaften vererbt werden können:

Listing 16: Vererbung mit extend (SCSS)

```
1 .message {
2     padding: 10px;
3     border: 1px solid #eee;
4   }
5
6   .warning {
7     @extend .message;
8     color: #E2E21E;
9   }
```

Der Code in Listing 16 ist in SCSS syntax geschrieben. In Zeile 7 werden mit @extend die Eigenschaften der Klasse .message vererbt. In den vorgestellten CSS-Präprozessoren Sass und Stylus, wird zur Vererbung dieselbe Variable verwendet. In Less gibt es wie bereits erwähnt, keine Vererbung in diesem Sinne. Wie eine Vererbung in Less implementiert werden kann, wird in Kapitel 5.4 behandelt.

3.5 Logic / Loops

Dieses Kapitel befasst sich mit Schleifen und if-else statements.
Schleifen und if-else statements in CSS-Präprozessoren funktionieren im Großen und Ganzen auf die selbe Weise wie in PHP oder einer anderen Programmiersprache. Alle 3 der relevanten CSS-Präprozessoren ermöglichen einfache if-else statements und for schleifen. Jedoch funktionieren in Sass und Stylus kompliziertere for schleifen, wie bspw. for-each schleifen, welche in LESS nicht unterstützt werden. Auch bei den if-konditionen ist mit Sass und Stylus mehr möglich als mit LESS, da in Sass und Stylus ebenso if/then/else statments möglich sind und nicht nur, wie bei LESS, einfache if/then statements.
Folgender Code zeigt eine Schleife und ein if/hten statement in LESS:(Coyier 2012)

Listing 17: if/then statement und loop in LESS

```
1  //mixin mit if/then
2  .set-bg-color (@text-color) when (lightness(@text-color) >=
       50%) {
3    background: black;
4  }
5  //aufruf des mixins
6  .box-1 {
7    color: #BADA55;
8    .set-bg-color(#BADA55);
9  }
10
11 //schleife
12 .loop (@index) when (@index > 0) {
13   .myclass {
14     z-index: @index;
15   }
16   // Call itself
17   .loopingClass(@index - 1);
18 }
19 // Stop loop
20 .loopingClass (0) {}
21
22 // Outputs stuff
23 .loopingClass (10);
```

In Kapitel 5.5 wird näher erklärt, wie Schleifen und if/then statements in LESS, Sass und Stylus funktionieren und worin die Unterschiede liegen.

4 Präprozessoren LESS, Sass, Stylus

4.1 LESS

Less ist einer der genannten CSS-Präprozessoren und wird in diesem Kapitel genauer erklärt.
Seit dem Jahr 2010 arbeitet Alexis Sellier an der Entwicklung von Less.
Less ist wie auch die Syntax-Variante SCSS von Sass ein Superset, siehe Kapitel 3, und kann somit ohne Probleme in eine CSS Datei eingefügt werden. Um eine funktionstüchtige Less-Datei zu erhalten ist es, bei einer generellen Verwendung von Less, nur notwendig die Endung einer CSS Datei in .less zu ändern. In die so erhaltene Less-Datei kann man die zusätzlichen Eigenschaften von Less einbauen und problemlos verwenden.
Less verwendet dieselbe Syntax wie css und bietet wie auch Stylus und Sass die Möglichkeit der Verschachtelung und der Verwendung von Features wie Variablen, Mixins, Vererbung und Funktionen.
Allerdings gibt es auch Erweiterungen, die in Sass und Stylus jedoch nicht in Less möglich sind. Beispiele dafür sind Defaultwerte für Variablen oder Vererbung von Klassen.
Im Gegensatz zu Sass, ist Less eine Javascript Library, die wie jede andere Javascript Library im HTML Head eingebunden wird. Zu Beachten ist, dass vor der Einbindung der js Datei die Stylesheets geladen werden.
Um mit Less arbeiten zu können benötigt man weder eine Commandozeile noch Tools wie Rhino oder Nodejs. Beispiel Einbindung:

Listing 18: Einbindung Less

```
1 //erst das Stylesheet, dann die js datei.
2 <link rel="stylesheet/less" type="text/css" href="styles.less"
     >
3 <script src="less.js" type="text/javascript"></script>
```

Diese Einbindung ist die Einfachste Art Less zu verwenden, jedoch gilt das nur für die Client-seitige Verwendung desweiteren läuft Less mit dieser Einbindung nur mit modernen Browsern wie z.B. den letzten Versionen von Chrome oder Firefox.
Der Watch mode wird entweder durch ein erneutes Laden der Datei nachdem „#!watch“ an die URL im headbereich geschrieben wurde oder wenn in der Konsole „less.watch()“ aufgerufen wird, aktiviert.
Für eine Serverseitige Verwendung wird less mit nodejs installiert. Somit kann nach der installation die Datei styles.less mit dem Compiler in Node in eine css datei kompiliert werden.(Sellier 2014)

Für die Kompilierung der Less Dateien gibt es einige andere Third Party Tools, wie zum Beispiel die kostenlosen Tools SimpLESS oder WinLESS.
In einem späteren Kapitel wird noch auf die Verwendung der genannten Erweiterungen mit Less eingegangen.

4.2 Sass

Sass ist eine Erweiterung von CSS3, welche Variablen, Mixins, Selectoren, Funktionen und andere Erweiterungen anbietet. Somit ist Sass, wie schon erwähnt, ein Präprozessor von CSS.
Einer der größten Vorteile eines CSS-Präprozessors ist, dass man in der Entwicklung verschiedene Dateien verwenden kann, ohne Performance, beim Laden der Seite, einzubüßen.
Um Sass in CSS umzuwandeln, verwendet man entweder die Kommandozeile oder eines von verschiedenen web-frameworks welche die notwendigen Funktionen bereitstellen. Darauf wird später noch genauer eingegangen.
Ursprünglich wurde Sass in ruby geschrieben. Zur Installation von Sass muss voher Ruby installiert werden. Auf Max OS ist ruby standardmäßig vorhanden, somit kann hier Sass sofort installiert werden. Mittlerweile gibt es sass auch ohne Ruby. Libsass zum Beispiel kann mit node.js verwendet werden.
Um jedoch alle Vorteile von Sass verwenden zu können, bspw. Compass muss Ruby installiert sein, da Compass auf Sass basiert und ebenso in Ruby geschrieben wurde.
Für Sass gibt es zwei verschieden Syntaxen. Die Ursprüngliche Syntax verwendet die Dateiendung .sass und verwendet Einrückungen statt der geschwungenen Klammern um die Verschachtelung der Selektoren anzuzeigen und Zeilenumbrüche statt eines Semicolons um die Eigenschaften zu trennen.
Die neuere Syntax verwendet die Dateiendung .scss. Hier werden im Gegensatz zur alten Syntax wieder Klammern und Semicolon verwendet. (Yard 2014b)
Folgender Codeausschnitt zeigt eine CSS-Datei mit der ursprüngliche Sass Syntax:

Listing 19: Code in ursprünglicher Syntax

```
1  #content .text
2      .title
3          font-size: 12px
4          color: #dedede
5          font-weight: normal
6          h1
7              font-size: 18px
8              color: #ffffaa
9              font-weight: bold
10
11     h3
12         color: #efefef
```

Nachfolgene Listing zeigt nun einen Codeausschnitt einer CSS-Datei mit der scss Syntax:

Listing 20: Code mit in scss Syntax

```
1  #content .text{
2    .title{
3      font-size: 12px;
```

```
4       color: #dedede;
5       font-weight: normal;
6       h1{
7         font-size: 18px;
8         color: #ffffaa;
9         font-weight: bold;
10      }
11      h3{
12        color: #efefef;
13      }
14    }
15  }
```

Wie schon erwähnt bietet Sass die Möglichkeit von Variablen, Funktionen, Mixins und vielem Mehr. In Kapitel 5 wird näher darauf eingegangen. Wie im vorangegangenen Kapitel beschrieben, ist Less ein Superset von CSS. Auch die neuere Syntax von Sass, Scss, ist ein Superset. Die Bedeutung von Superset wird in Kapitel 3 beschrieben.

4.3 Stylus

Stylus ist ein CSS-Präprozessor der mit Nodejs läuft und dessen Dateiendung .styl ist.
Mit dem Befehl “npm install stylus -g„ lässt sich stylus über nodejs installieren und verwenden.
Die Syntax von Stylus ist ähnlich der von Sass. Wie auch bei Sass, können bei Stylus die Klammern und die Semicolons weggelassen werden. Verwendet man diese Syntax ist Stylus kein Superset von CSS und daher nicht rückwärtskompatibel.
Die Schreibweise der Syntax ohne Klammern und Semicolons ist optional und es kann auch die normale CSS-Syntax angewandt werden.
Desweiteren ist es auch möglich, die Syntaxen zu vermischen. Folgender Code zeigt beide Syntaxen in einer Datei:

Listing 21: style.styl

```
1  //Mixin in Stylus
2  border-radius()
3     -webkit-border-radius: arguments;
4     -moz-border-radius: arguments;
5     border-radius: arguments;
6  //Verwendung der normalen CSS-Syntax
7   body a {
8     font: 12px/1.4 "Lucida Grande", Arial, sans-serif;
9     background: black;
10    color: #ccc;
11  }
12 //Verwendung der Stylus Syntax ohne Klammern, aber mit
      Semicolons
13  form input
14    padding: 5px;
15    border: 1px solid;
16    border-radius: 5px;
```

(LearnBoost 2010b)

In diesem Codebeispiel wird zuerst ein Mixin erstellt. Hier ist zu sehen, dass dies mit Stylus anders funktioniert als mit Less oder Sass. Darauf wird in Kapitel 5 noch genauer eingegangen.
Die codezeilen 8 bis 12 sind normales CSS und der letzte Abschnitt ist sowohl Stylus-Syntax als auch CSS-Syntax. Es werden keine Klammern verwendet, jedoch Semicolons.
Wie der Code aus Listing 7 zeigt, ist Stylus in der Benützung sehr einfach, da, wie schon am Begin des Kapitels erwähnt, die Verwendung von Klammern und Strichpunkten optional ist. Auch die anderen genannten Erweiterungen, können in Stylus angewendet werden.
Auf die genaue Verwendung wird im nächsten Kapitel eingegangen.

5 Implementierugen von CSS Präprozessoren

5.1 Variablen

Variablen sind wie schon in Kapitel 2.6.1 beschrieben, eine Erweiterung der CSS-Präprozessoren. Mithilfe der Variablen können Werte am Beginn des Stylesheets definiert werden und an jeder Stelle im Code wiederverwendet werden. In diesem Kapitel wird die Verwendung von Variablen mit Less, Stylus und Sass erläutert. Im Großen und Ganzen ist die Verwendung sehr ähnlich und es werden hier nur die Unterschiede näher betrachtet.

5.1.1 Less

In Less wird die Variable mit dem Zeichen „@“ gekennzeichnet. Listing 17 zeigt die Initialisierung der Variable und die anschließende Verwendung im Code:

Listing 22: Verwendung Variable in less

```
1 @color: #4D926F;
2
3 #header {
4   color: @color;
5 }
6 h2 {
7   color: @color;
8 }
```

Wie in Zeile 4 zu sehen, muss die Variable wieder mit dem „@“ Zeichen aufgerufen werden, da ohne dem “@„ Zeichen keine Farbe mitgegeben wird.

5.1.2 Sass

Wie auch in Less gibt es für Sass eine eigene Schreibweise für Variablen. Diese Schreibweise gilt für beide Syntaxvarianten von sass.

Listing 23: Verwendung Variable in sass

```
1 $color: #4D926F;
2
3 #header {
4   color: $color;
5 }
6 h2 {
7   color: $color;
8 }
```

Der Code in Listing 18 zeigt die Verwendung einer Variable in Sass. Zeile 1 zeigt, dass die Variable mit „$“ dargestellt werden muss. Auch bei der Verwendung muss, wie bei less das Zeichen „@“, das Zeichen „$“ vorangestellt werden, damit die Variable richtig erkannt wird.

5.1.3 Stylus

Im Gegensatz zu Less und Sass muss bei Stylus kein Sonderzeichen vor die Variable gesetzt werden, jedoch kann die Schreibweise von Sass verwendet und vor die Variable das Zeichen „$“ geschrieben werden. Nachfolgender Code zeigt beide Schreibweisen von Variablen in Stylus:

Listing 24: Verwendung Variable in stylus

```
1  \\Initialisierung Variable ohne Sonderzeichen
2  font-size = 14px
3  font = font-size "Lucida Grande", Arial
4  \\Verwendung Variable
5  body
6    font font, sans-serif
7
8  \\Initialisierung Variable mit Sonderzeichen
9  $font-size = 14px
10 \\Verwendung Variable
11 body {
12   font: $font-size sans-serif;
13 }
```

(LearnBoost 2010b) Wie das Listing zeigt, wird in Zeile 2 und 3 die Variable ‚font-size’ bzw. ‚font’ ohne voransetzen eines Sonderzeichens initialisiert. Somit wird auch beim Aufruf der Variable kein Sonderzeichen vorangestellt. Wie in Zeile 3 zu sehen, wird die Variable ‚font-size’ direkt in der Variable ‚font’ aufgerufen. Dadurch muss in Zeile 6 bei der Verwendung nur die Variable ‚font’ aufgerufen werden.
In Zeile 9 wird die Variable wie bei Sass mit dem Zeichen „$“ initialisiert. Dadurch ist auch die Verwendung ident mit der von Sass.

Die letzten 3 Unterkapitel zeigen, dass eine Variable in den 3 beschriebenen CSS-Präprozessoren bis auf die verwendeten Sonderzeichen gleich erstellt und verwendet werden. Im nächsten Kapitel wird die Verwendung von Mixins näher betrachtet.

5.2 Mixins

Mixins sind eigene Klassen, welche am Beginn des Stylesheets erstellt werden und fortlaufend im ganzen Stylesheet wiederverwendet werden können.
In Kapitel 2.6.2 wurde beschrieben, was Mixins sind und wie sie im generellen funktionieren. In diesem Kapitel, wird für jeden Präprozessor erklärt, wie ein Mixin formuliert und angewendet wird.

5.2.1 Less

Mit Less wird eine Mixin Klasse erstellt wie jede andere Klasse auch. Folgender Code erstellt ein Mixin und verwendet dieses anschließend:

Listing 25: Verwendung Mixin in Less

```
1  .rounded-corners (@radius: 5px) {
2    border-radius: @radius;
3    -webkit-border-radius: @radius;
4    -moz-border-radius: @radius;
5  }
6
7  #header {
8    .rounded-corners;
9  }
10 #footer {
11   .rounded-corners(10px);
12 }
```

(Sellier 2014) Die Zeilen 1 bis 5 aus Listing 20 erstellen die Mixin-Klasse “rounded-corners„, welche auch die Mitgabe eines Parameter, in diesem Fall den Radius, ermöglicht.
In Zeile 8 und 11 wird die Klasse dann aufgerufen und verwendet. Wie erwähnt, ermöglicht das Mixin auch Parameter. Als Defaultwert wird dem Mixin in Zeile 1 ein Radius von 5px zugewiesen und in Zeile 11 wird der Defaultwert überschrieben und auf 10px geändert.

5.2.2 Sass

Im Gegensatz zu Less, wird bei Sass nicht einfach eine Klasse definiert. Listing 21 zeigt, wie mit Sass ein Mixin erstellt wird:

Listing 26: Verwendung Mixin in Sass

```
1  @mixin large-text {
2    font: {
3      family: Arial;
4      size: 20px;
5      weight: bold;
6    }
7    color: #ff0000;
8  }
9
10 .page-title {
11   @include large-text;
12   padding: 4px;
13   margin-top: 10px;
14 }
```

(Yard 2014b) Der Code zeigt in Zeile 1, dass in Sass für die Erstellung eines Mixins "@mixin„ vor den Namen des Mixins gesetzt werden muss. Trotz der differenten Schreibweise, wird ein Mixin auch in Sass wie eine Klasse behandelt.
Für die Verwendung des Mixins wird in Zeile 11 mit "@include„ und dem Mixin-Namen ‚large-text' aufgerufen.

5.2.3 Stylus

In Stylus ist der Aufbau eines Mixins sehr ähnlich dem mit Less.

Listing 27: Verwendung Mixin in Stylus

```
1 rounded-corners (n)
2   border-radius  n
3   -webkit-border-radius  n
4   -moz-border-radius  n
5
6 #footer
7   rounded-corners(10px)
```

In den Zeilen 1 bis 4 wird das Mixin erstellt. In Zeile 1 wird die Variable n als Parameter mitgegeben, welcher beim Aufruf des Mixins in Zeile 11 ausgefüllt wird. Somit beträgt der Radius 10px.
In Stylus ist die Syntax sehr variabel und es kann von dem Entwickler, der Entwicklerin selbst entschieden werden, ob Klammern, Strichpunkte und Doppelpunkte geschrieben werden oder nicht. Verwendet man die Syntax mit den Zeichen, sieht der Code aus Listing 22 wie folgt aus:

Listing 28: Verwendung Mixin in Stylus

```
1 .rounded-corners (n){
2   border-radius: n;
3   -webkit-border-radius: n;
4   -moz-border-radius: n;
5 }
6
7 #footer{
8   .rounded-corners(10px);
9 }
```

Listing 23 zeigt, dass die Erstellung des Mixins mit dieser Syntax gänzlich der Erstellung mit Less gleicht.

5.3 Funktionen

Dieses Kapitel behandelt die Verwendung von Funktionen und Operatoren mit den CSS-Präprozessoren. Funktionen und Optionen helfen die Strutkur und Lesbarkeit des Stylesheets zu optimieren und es können komplexe Strukturen innerhalb der CSS Datei erstellt werden.
Bei Sass und Stylus können Funktionen eigens definiert werden und im ganzen Stylesheet angewendet werden.
In Less gibt es sehr viele vordefinierte Funktionen, die Beispielsweise Farbmischungen ermöglichen.

5.3.1 Less

Wie erwähnt, gibt es in Less viele verschiedene vordefinierte Funktionen zum Erstellen von komplexen Strukturen. Einige Beispiele, auf die im Anschluss noch genauer eingegangen wird, sind:

- floor
- argb
- saturation
- fadein
- mix
- average

Alle Codebeispiele sind von (Sellier 2015a) **floor**
Floor ist eine mathematische Funktion, die zur Berechnung von Integern dient. Mit floor kann ein Integer abgerundet werden. Die Funktion ist wie bei PHP, allerdings kann hier nicht festgelegt werden auf wie viele Stellen abgerundet wird. Bei der Funktion in Less wird automatisch auf die nächst niedrigere ganze Zahl abgerundet.

argb
Argb ist wie ‚rgb' eine Farbfunktion und berechnet eine hexadezimale Version der angegebenen Farbe im #AARRGGBB Format. Anders als bei rgba wird hierbei der Alphawert zu Beginn verwendet. Beispielsweise:

Listing 29: ARGB in Less

```
1  rgb(90, 23, 148);
2  //Ausgabe:
3  #5a1794
4
5  rgba(90, 23, 148, 0.5)
6  //Ausgabe:
7  #5a179480
8
9  argb(rgba(90, 23, 148, 0.5));
10 //Ausgabe
11 #805a1794
```

Wie das Listing zeigt, wird der Alphawert, welcher in der rgba-Funktion als letzter Parameter mitgegeben wird, bei der Berechnung des Hexadezimalwertes am Beginn verwendet.

saturation
Saturation ist wie ‚argb' eine Farbfunktion, die jedoch nicht wie argb die übergebenen Parameter in einen Hexadezimalwert umrechnet sondern den Sättigungskanal aus dem Farbobjekt extrahiert. Die Ausgabe dieser Funktion ist ein Prozentwert zwischen 0 und 100 und wird aus einem hsl-Wert ermittelt.

Listing 30: Saturation in Less

```
1  saturation(hsl(90, 100%, 50%))
2  //Ausgabe
3  100%
```

fadein
Die Farboperation ‚fadein' erhöht die Deckkraft der übergebenen Farbe. Als Mitgabeparameter stellt die Funktion einen hsla-wert und eine Prozentzahl zwischen 0 und 100 bereit. Nachfolgender Code zeigt, wie der übergebene Farbwert mit der übergebenen Prozentzahl berechnet wird.

Listing 31: Fadein in Less

```
fadein(hsla(90, 90%, 50%, 0.5), 10%)
//Ausgabe
rgba(128, 242, 13, 0.6) (hsla(90, 90%, 50%, 0.6))
```

Die gegenteilige Funktion zu ‚fadein' ist ‚fadeout', welche die Deckkraft nicht erhöht sondern vermindert.

mix
Mix ist eine Funktion, welche zwei Farben miteinander vermischt. Hierbei wird auch die Transparenz beachtet. Mitgegeben werden der Funktion 2 Farbwerte und die gewünschte Transparenz.

Listing 32: Mix in Less

```
mix(#ff0000, #0000ff, 50)
mix(rgba(100,0,0,1.0), rgba(0,100,0,0.5), 50)
//Ausgabe
#800080
rgba(75, 25, 0, 0.75);
```

average
Average ist ebenso, wie die gleichnamige Funktion in PHP, eine Funktion, welche den Durchschnitt berechnet. In Less berechnet die Funktion aber nicht das Mittel von Integerwerten sondern den Durchschnitt von Farbwerten. Die mitgegebenen Parameter sind zwei Farbobjekte, aus denen ein drittes Farbobjekt, der Medianwert ermittelt wird.

Listing 33: Average in Less

```
average(#ff6600, #000000);
//Ausgabe
#ff6600 #000000 #803300
```

(Sellier 2014)

Neben den vorgestellten Funktionen, die von Less zur Verfügung gestellt werden, kann man auch einfache mathematische Funktionen erstellen wie z.B.

Listing 34: Funktionen in Less

```
@base: 5%;
@filler: (@base * 2);

height: (100% / 2 + @filler);
```

5.3.2 Sass

In Sass wird eine Funktion auf folgende Weise erstellt:

Listing 35: Verwendung Funktion in Sass

```
1 $grid-width: 40px;
2 $gutter-width: 10px;
3
4 @function grid-width($n) {
5   @return $n * $grid-width + ($n - 1) * $gutter-width;
6 }
7
8 #sidebar { width: grid-width(5); }
```

(Yard 2014b) Wie bei der Verwendung von Variablen beschrieben, werden in Zeile 1 und 2 die Variablen ‚grid-width' und ‚gutter-width' erstellt. Die Zeilen 4 bis 6 erstellen die Funktion, der eine Variable mitgegeben werden kann. In der Funktion wird dann mithilfe des mitgegebenen Parameters und der zuvor definierten Werte ‚grid-width' und ‚gutter-width' die Breite berechnet, welche in Zeile 8 aufgerufen wird.
Die Funktion in Listing 30 zeigt, wie Funktionen in Sass im Generellen erstellt und verwendet werden.

5.3.3 Stylus

Wie in Sass, sieht in Stylus die Erstellung einer Funktion sehr ähnlich aus. Da jedoch bei Stylus auf Klammern, Strichpunkte und Doppelpunkte verzichtet werden kann, wird hierbei eine Funktion wie in Listing 31 gezeigt, formuliert:

Listing 36: Verwendung Funktion in Stylus

```
1 grid-width: 40px;
2 gutter-width: 10px;
3
4 function grid-width(n)
5   return n * grid-width + (n - 1) * gutter-width
6
7 #sidebar
8    width: grid-width(5)
```

Wie bereits im Bezug auf die Erstellung einer Variablen mit Stylus erwähnt, wird auch hier kein Sonderzeichen benötigt, um eine Variable zu definieren. Ebenso wird für die Funktion nichts dergleichen benötigt.

5.4 Vererbung

Kapitel 2.6.4 beschreibt, was Vererbung bei CSS-Präprozessoren bedeutet und wie diese im generellen funktioniert. Dieses Kapitel beschreibt für jeden behandelten CSS-Präprozessor, wie Vererbung zustande kommt und verwendet wird.

5.4.1 Less

In Less funktioniert die Vererbung durch Verschachtelung.

Listing 37: Vererbung in Less

```
1  #header {
2    h1 {
3      font-size: 26px;
4      font-weight: bold;
5    }
6    p { font-size: 12px;
7      a { text-decoration: none;
8        &:hover { border-width: 1px }
9      }
10   }
11 }
```

Auch mit Hilfe von Mixins kann in Less vererbt werden. Ein Mixin ist wie erwähnt eine eigene Klasse und diese kann in jeder anderen Klasse durch das Aufrufen vererbt werden. Wie ein Mixin erstellt und aufgerufen wird, wurde bereits in Kapitel 5.2.1 erläutert.

5.4.2 Sass

Sass behandelt Vererbung anders als Less, da Sass für die Vererbung von Klassen die Anweisung „@extend “, welche an jeder Stelle im Code aufgerufen werden kann, verwendet. Listing 33 zeigt ein kurzes Beispiel, wie die Vererbung in Sass funktioniert.

Listing 38: Vererbung in Sass

```
1  .error {
2    border: 1px #f00;
3    background-color: #fdd;
4  }
5  .seriousError {
6    @extend .error;
7    border-width: 3px;
8  }
```

Die Listing zeigt, wie in der Klasse ‚.seriousError’ die Klasse ’.error’ vererbt wird.

5.4.3 Stylus

Stylus vererbt Klassen auf genau dieselbe Weise wie Sass.

Listing 39: Vererbung in Stylus

```
1 .error {
2   border: 1px #f00;
3   background-color: #fdd;
4 }
5 .seriousError {
6   @extend .error;
7   border-width: 3px;
8 }
```

5.5 Logic/Loops

Wie in Kapitel 3.5 beschrieben, werden in allen relevanten CSS-Präprozessoren Schleifen und if/then statements unterstützt.
In diesem Kapitel wird genauer erklärt, worin die Unterschiede bestehen.

5.5.1 LESS

In LESS gibt es nur einfache if/then statements und nicht wie in Sass und Stylus if/then/else statements. Es kann also nur eine einfache Abfrage erstellt werden.

Listing 40: Logic in LESS

```
1 //mixin if/then
2 .set-bg-color (@text-color) when (lightness(@text-color) >=
     50%) {
3   background: black;
4 }
5 .set-bg-color (@text-color) when (lightness(@text-color) <
     50%) {
6   background: #ccc;
7 }
```

Wie das Listing zeigt, muss für jede Variante, die Abgefragt werden soll, ein eigenes Statement gemacht werden.

In LESS ist auch eine einfach Schleife möglich. Es gibt aber keine Möglichkeit für eine each-Schleife wie in Sass.
Für die Erstellung einer Schleife wird in LESS eine, auf sich selbst refernzierende Rekursion formuliert, in der ein Mixin rekursiv mit angepassten Werten aufgerufen wird.

Listing 41: Loop in LESS

```
.loop (@index) when (@index > 0) {
  .myclass {
    z-index: @index;
  }
  // Call itself
  .loopingClass(@index - 1);
}
// Stop loop
.loopingClass (0) {}

// Outputs stuff
.loopingClass (10);
```

(Coyier 2012)

5.5.2 Sass

Sass unterstützt zusätzlich zum normalen if/then Statement noch das if/then/else Statement, somit kann jede Bedingung in einem Statement zusammengefasst werden und es muss nicht für jede Bedingung ein einzelnes Statement geschrieben werden.

Listing 42: if/then/else Schleife Sass

```
@if $type == ocean {
    color: blue;
  } @else if $type == matador {
    color: red;
  } @else if $type == monster {
    color: green;
  } @else {
    color: black;
  }
```

Wie schon in Kapitel 3.5 erwähnt, unterstützt Sass auch bei den Schleifen mehrere Möglichkeiten. Für die Umsetzung von Schleifen gibt es nicht nur die for-Loop sondern zusätzlich noch die While-Schleife und die each-Schleife.

Listing 43: for Schleife Sass

```
@for $i from 1 through 3 {
  .item-#{$i} { width: 2em * $i; }
}
```

Listing 44: while Schleife Sass

```
$i: 6;
@while $i > 0 {
  .item-#{$i} { width: 2em * $i; }
  $i: $i - 2;
}
```

Listing 45: each Schleife Sass

```
@each $animal in puma, sea-slug, egret, salamander {
  .#{$animal}-icon {
    background-image: url('/images/#{$animal}.png');
  }
}
```

(Yard 2014b)

5.5.3 Stylus

Wie auch in Sass, gibt es in Stylus die Möglichkeit eines if/then/else Statements.

Listing 46: if-else Statement Stylus

```
box(x, y, margin-only = false)
  if margin-only
    margin y x
  else
    padding y x
```

Anders als in LESS und in Sass bietet Stylus noch das unless-Statement, welches Analog zum ‚!=‘ von PHP oder Javascript, funktioniert.

Listing 47: if-else Statement Stylus

```
disable-padding-override = true

 unless disable-padding-override is defined and
    disable-padding-override
   padding(x, y)
     margin y x

 body
   padding 5px 10px
```

Neben den if/else Statements bietet Stylus wie LESS und Sass ebenso die Möglichkeit von Schleifen. In Stylus wird eine Schleife ähnlich der for-schleife bei Sass ausgeführt.

Listing 48: for-in Schleife Stylus

```
for <val-name> [, <key-name>] in <expression>

//for ohne key
body
  for num in 1 2 3
    foo num

//for mit key
body
    fonts = Impact Arial sans-serif
    for font, i in fonts
      foo i font
```

(LearnBoost 2010b)

Nach den Erläuterungen zur Funktionalität der Erweiterungen iin LESS, Sass und Stylus wird im Anschluss die Umsetzung der praktischen Arbeit beschrieben um im in der Folge die Ergebnisse zu präsentieren.

6 Praktische Anwendung und Vergleiche

Der praktische Teil dieser Arbeit besteht im Großen und Ganzen aus einer index.html, einer Sass Datei, einem Gruntfile und einer package.json. Das Ergebnis der praktischen Arbeit wird eine Library, die mit yeoman vom Enduser erstellt werden kann, und dann entweder die enthaltene Sass-Datei oder eine daraus generiert Less, Css oder Stylus Datei erstellt und zur Verfügung stellt.
Zusätzlich zum Yeoman Generator wird auch ein extra Grunttask entwickelt, der dem Verwender, der Verwenderin ermöglicht, ein beliebiges Stylesheet in LESS, Sass oder Stylus zu convertieren. Die möglichen Zieldateien können entweder css, oder eines der anderen beiden Präprozessor varianten sein. Wie dieser Grunttask entwickelt wird, wird an einer späteren Stelle des Kapitels erläutert.

Die enthaltene Sass Datei ist die wichtigste Komponente für die Arbeit. Die Library, die am Ende zur Verfügung gestellt wird, ermöglicht es dem User in möglichst wenig Schritten eine fertige Less, Sass, Stylus oder Css Datei zu erhalten, die den Code für Css-Animationen enhält. Für den User werden diese Dateien mit Hilfe dieser, im Projekt enthaltenen, Sass Datei gebildet.
Um die Library gut vorstellen zu können, werden die mit Sass erstellten Animationen noch auf einer Seite grafisch dargestellt.

6.1 Yeoman Generator

Bereits in vorangegangenen Kapiteln wurde beschrieben, dass Less oder Sass oder Stylus geparst werden muss, um daraus Css zu erhalten. Nun wird beim praktischen Teil nicht nur für die Kompilierung der Sass Datei in eine Css Datei ein Parser verwendet sondern auch für die Kompilierung in Less und Styulus.

Für die Umsetzung des pratkischen Teils fiel die Entscheidung auf Yeoman mit Grunt und Node.js.
Am Beginn der Arbeit steht die Erstellung der Sass Datei. Die Animationen werden mit Mixins erstellt und in bestimmten Klassen aufgerufen. Somit wird dem Enduser die Verwendung der Datei erleichtert, da er in seinem div-Container nur noch die richtige Klasse aufrufen muss. Hier ein kurzer Codeausschnitt aus der Sass Datei:

Listing 49: main.scss

```
43 @mixin rotate{
44     .box:hover{
45         transform:rotate(30deg); /* W3C */
46     }
47 }
48 ...
49
```

```
50 .rotate{
51     @include rotate;
52 }
```

Der Code in Listing 34 zeigt, wie in der neuen Syntax von Sass, welche schon Kapitel 2.4 erläutert wurde, ein Mixin erstellt und verwendet wird. Durch die Verwendung von autoprefixer, kann im Stylesheet auf die prefixes verzichtet werden.
Das Mixin rotate dient dazu, den div container, auf den das Mixin angewendet wird, um 30 Grad im Uhrzeigersinn zu drehen . Das in der Listing dargestellten Mixin ist nur eines von mehreren, welche in der praktischen Arbeit angewendet werden. Das vollständige Stylesheet und die dazugehörige HTML-Datei befindet sich im Anhang.

Das Erstellen der Sass Datei ist ein wesentlicher Teil, jedoch muss auch ermöglicht werden, diese Sass Datei in Css, Less oder Stylus umzuwandeln. Hierfür gibt es das Gruntfile und die package.json. In der package.json werden alle benötigten Grunt-Libraries als Abhängigkeiten angegeben. Das heißt, bevor das Gruntfile richtig ablaufen kann, müssen alle Libraries installiert sein, die in der package.json stehen.
Auszug aus der package.json:

Listing 50: package.json

```
25 "dependencies": {
26     "yeoman-generator": "^0.18.0",
27     "chalk": "^0.5.0",
28     "yosay": "^0.3.0",
29     "fs": "*",
30     "grunt": "~0.4.5",
31     "load-grunt-tasks": "*",
32     "grunt-cli": "0.1.13",
33     "grunt-sass": "0.8.1",
34     "grunt-scss2less":"*",
35     "grunt-contrib-watch": "~0.6.1",
36     "grunt-contrib-clean": ">=0.4.0",
37     "jit-grunt": "~0.7.0"
38   },
```

Die Libraries in Zeile 30, 32 und 37 werden benötigt, damit das Gruntfile richtig ausgeführt und die anderen Libraries richtig interpretiert werden. Wie die Listing zeigt, gibt es für jeden Schritt, der benötigt wird um das Endergebnis der praktischen Arbeit zu erhalten, eine eigene Grunt-Library. Mit der *grunt-contrib-sass* bspw. wird die Sass Datei in eine Css Datei kompiliert.
Die Library in Zeile 34 ist für die Kompilierung von Sass in Less. Für die Umwandlung von Scss in Stylus, gibt es keinen vorgefertigten Grunt-Task, daher wurde dieser eigens erstellt.
Zur Verwendung dieser Libraries wird nun ein Gruntfile benötigt, in dem genau definiert

wird, wann welche Grunt-Library ausgeführt wird und was genau gemacht werden soll. Die nächste Lisiting zeigt einen Ausschnitt eines der, in dieser Arbeit verwendeten, Gruntfiles.

Listing 51: Gruntfile.js (Scss zu LESS)

```
30   scss2less: {
31       options: {
32         sourceMap: 'none'
33       },
34       dist: {
35         files: {
36           'less/main.less': 'sass/main.scss'
37         }
38       }
39     },
40    autoprefixer: {
41       no_dest: {
42         src: 'less/main.less'
43       },
44     }
45   });
46   grunt.registerTask('default', ['scss2less:dist', '
       autoprefixer', 'watch']);
47   grunt.loadNpmTasks('load-grunt-tasks');
48   grunt.loadNpmTasks('grunt-autoprefixer');
49   grunt.loadTasks('tasks');
```

Die Listing zeigt jenen Auschnitt, in dem definiert wird, wie die sass Datei in Less umgewandelt werden soll, um anschließend mit dem autoprefixer die prefixes hinzuzufügen.
Im hier verwendeten Gruntfile, sehen für alle festgelegten Kompilierungen die Codeabschnitte in etwa so aus wie jener aus Listing 36. Unterschiedlich ist lediglich die Bezeichnung der verwendeten Library, in Listing 36 scss2less, und das Ziel, in Listing 36 der Ordner less und die datei main.less.

In den Zeilen 30 bis 39 wird der Task scss2less genau definiert. Hier wird angegeben, dass die Quelldatei 'main.scss' im Ordner 'sass' in die Zieldatei 'main.less' im Ordner 'less' konvertiert wird. Der Task wird in Zeile 45 mit dem Befehl 'scss2less:dist' aufgerufen.

Wie zu Beginn des Kapitels angeführt wurde, stellt der praktische Teil dieser Arbeit eine Library in Form eines Yeoman Generators dar. Um einen Yeoman Generator zu erstellen, benötigt es nicht nur eine Package.json und ein Gruntfile.js sondern auch eine index.js in der definiert wird, was beim Aufruf des Generators ausgeführt wird. Nachfolgende Listing zeigt einen Ausschnitt dieser index.js.

Listing 52: index.js

```
6  module.exports = yeoman.generators.Base.extend({
7    initializing: function () {
8      this.pkg = require('../package.json');
9    },
10
11   prompting: function () {
12     var done = this.async();
13
14     // Have Yeoman greet the user.
15     this.log(yosay(
16       'Welcome to the doozie ' + chalk.red('Animations') + '
           generator!'
17     ));
18
19     var prompts = [{
20       name: 'someOption',
21       message: "Waehlen Sie eine Stylesheetvariante? (CSS/
           Stylus/Less/Scss)",
22       default: 'Scss'
23     }];
24
25     this.prompt(prompts, function (props) {
26       this.someOption = props.someOption;
27
28       done();
29     }.bind(this));
30   },
```

Die aufgeführte Listing zeigt, wie der Generator aufgerufen und initialisiert wird. Zeile 8 der Listing bindet die package.json, aus welcher ein Auschnitt gezeigt wurde, ein.
Somit können die benötigten Node Packages durch den Aufruf der Funktion in den Zeilen 9 bis 13 installiert werden.
Die Funktion 'prompting' in den Zeilen 16 bis 35 bietet die Möglichkeit individuelle Ausgaben in der Console zu schreiben. Der Prompt in Zeile 24 bis 28 bspw. ist jener Prompt der dem User anzeigt, eine Stylesheetvariante zu wählen. Der Userinput auf diese Frage, entscheidet den weiteren Verlauf der Library. Im weiteren Code der index.js, der im Anhang zu sehen ist, wird für jede Eingabe definiert, was anschließend passiert. Wählt der User z.B. Less, sieht der Code folgendermaßen aus:

Listing 53: index.js

```
56 case 'Less':
57   this.mkdir('sass');
58   this.fs.copy(
```

```
59      this.templatePath('main.scss'),
60      this.destinationPath('/sass/main.scss')
61    );
62    this.fs.copy(
63      this.templatePath('_lessGruntfile.js'),
64      this.destinationPath('Gruntfile.js')
65    );
66  break;
67  ...
68  this.fs.copy(
69    this.templatePath('index.html'),
70    this.destinationPath('index.html')
71  );
72  this.fs.copy(
73    this.templatePath('_package.json'),
74    this.destinationPath('package.json')
75  );
```

Durch den Code in Listing 38 wird die ursprüngliche Scss Datei in einen Ordner kopiert. Ebenso werden das Gruntfile und die Package.json in den richtigen Ordner kopiert. Anschließend, kann der user mit dem Befehl 'grunt' das Ausführen der Tasks im Gruntfile aufrufen und erhält nach erfolgreicher Durchführung das gewünschte Stylesheet.

6.2 Grunt Task

Für den bereits erwähnten GruntTask wurden für die Umsetzung einige vorgefertigte gruntasks als Grundlage verwendet. Nachfolgende Listing zeigt, wie ein Grunt-Task im Generellen erstellt wird.

Listing 54: Erstellung Grunt Task

```
1  'use strict';
2
3  module.exports = function(grunt){
4    var warnReal = grunt.fail.warn;
5    var warnFake = function () {
6      arguments[0] = 'Warning '.cyan + arguments[0];
7      grunt.log.writeln.apply(null, arguments);
8    };
9
10   grunt.registerMultiTask('less2sass2stylus2css', 'Convert
       Less to CSS, SCSS or Stylus; convert SCSS to Less, CSS
```

```
      or Stylus;  convert Stylus to Less, SCSS or CSS',
      function() {
11      //do somethin awesome
12    });
13  }
```

Nachdem der Grunttask erstellt wird, werden einige Variablen definiert, welche für die Verwendung der benötigten packages notwendig sind. Wie bereits erwähnt, werden für ein paar mögliche Konvertierungen bereits vorhandene grunt-tasks als Vorlage verwendet. Hierfür werden im Grunttask auch bestimmte Optionen festgelegt.
Listing 55 zeigt, wie der Code nach diesen Erweiterungen aussieht.

Listing 55: Erstellung Grunt Task

```
1  'use strict';
2
3  var async = require('async');
4  var path = require('path');
5  var _ = require('lodash');
6  var chalk = require('chalk');
7  var less = require('less');
8
9  module.exports = function(grunt){
10   var warnReal = grunt.fail.warn;
11   var warnFake = function () {
12     arguments[0] = 'Warning '.cyan + arguments[0];
13     grunt.log.writeln.apply(null, arguments);
14   };
15   var fs = require('fs');
16
17   grunt.registerMultiTask('less2sass...', '...', function() {
18     var done = this.async();
19     var options = this.options({
20       report: 'min',
21       includePaths: [],
22       outputStyle: 'nested',
23       sourceComments: 'none',
24       separator: grunt.util.linefeed,
25       banner: ''
26     });
27     var banner = options.banner;
28     var filesCreatedCount = 0;
29     if (options.basePath || options.flatten) {
30       grunt.fail.warn('Wildcards are no longer supported.');
```

```
31      }
32      if (this.files.length < 1) {
33        grunt.verbose.warn('No source files were provided.');
34      }
35    });
36  }
```

Wie das Listing zeigt, wird in den definierten Variablen angegeben, welche packages zusätzlich benötigt werden. Diese Packages müssen in der package.json angegeben werden, damit der Grunttask funktioniert.
Im Anschluss an das letzte if-Statement wird eine Schleife erstellt, in welcher die Files, welche im Gruntfile angegeben werden, ausgelesen werden. In dieser Schleife wird dann für jede Variante des Sourcefiles ein If-Statement verfasst. Die möglichen Sourcefiles sind entweder Less, Sass oder Stylus.
In jedem If-Statement für das Sourcefile wird wiederum für jede Variante der Zieldatei ein If-Statement erstellt. In diesem If-Statement wird dann der Code zur Konvertierung angegeben. Nachfolgende Listing zeigt das Beispiel anhand von Scss als Quelldatei und css als Zieldatei:

Listing 56: Konvertierung Scss in CSS

```
1  async.eachSeries(this.files, function(f, nextFileObj) {
2    var destFile = f.dest;
3    var files = f.src.filter(function(filepath) {
4      if (!grunt.file.exists(filepath)) {
5        grunt.log.warn('"'+filepath + '" not found.');
6        return false;
7      } else {
8        return true;
9      }
10   });
11   if(f.src[0].split(".").pop() == 'scss'){
12     if(destFile.split(".").pop() == 'css'){
13       var sass = require('node-sass');
14       sass.render({
15         file: f.src[0],
16         success: function (css) {
17           grunt.file.write(destFile, css);
18           grunt.log.writeln('"' + destFile + '" created.');
19           nextFileObj();
20         },
21         error: function (err) {
22           grunt.warn(err);
23         },
```

```
24            includePaths: options.includePaths,
25            outputStyle: options.outputStyle,
26            sourceComments: options.sourceComments
27          });
28      }
29    }
30  }, done);
```

Für dieses Beispiel wird der, bereits bestehende, grunt-Task „grunt-contrib-sass“ als Grundlage verwendet.
Ebenso für die Konvertierung von Scss in LESS, LESS in CSS und Stylus in CSS gibt es bereits grunt-tasks die für diese Arbeit als Grundlage dienen.
Alle weiteren Konvertierungen werden selbst entwickelt und mit regular expressions umgesetzt.[3]

Das gelesene Kapitel erläutert, wie der praktische Teil dieser Arbeit aufgebaut ist und durchgeführt wird.
Die Entwicklung dieser Library und des Grunttasks wurde auf Grund der Verwendung von Präprozessoren und der Darstellung der Umwandlung dieser in Css, als praktischer Teil gewählt. Desweiteren bietet die Library die Möglichkeit, die Anwendung und Handhabung der vorgestellten CSS-Präprozessoren, realitätsnahe zu verwirklichen und zusätzlich eine einfache Lösung für CSS-Animationen zur Verfügung zu stellen.

Im weiteren Verlauf dieser Arbeit werden mit Hilfe der, im praktischen Teil, erstellten Stylesheets, die Vorteile von Präprozessoren überprüft.
Diese Überprüfung gilt im Besonderen dem Workflow und der Codequalität.

3. gesamter Code zum Grunttask: https://raw.githubusercontent.com/Babsi2/grunt-scss-less-stylus-css/master/tasks/less2sass2stylus2css.js

6.3 Ergebnisse

In den letzten 2 Kapiteln wurde beschrieben, wie der praktische Part der Thesis durchgeführt wurde und wie die Verwendung von Präprozessoren umgesetzt werden kann. Für die Erstellung des Yeoman Generators wurde als Basis ein Stylesheet in SCSS-Syntax verfasst. Dieser Präprozessor wurde gewählt, da hierbei die Syntax, jender von CSS3 am Ähnlichsten ist.
Wie bereits beschrieben, wurden die Animationen für den Generator mit Mixins erzeugt. Durch die Verwendung der Mixins konnte das Schreiben des Codes übersichtlicher und einfacher gestaltet werden. Bei der SCSS-Syntax, werden zu beginn des Stylesheets alle benötigten Mixins erzeugt und im Anschluss in den Klassen aufgerufen. Somit kann der Code auch kürzer gehalten werden als bei CSS.
Ein weiterer Vorteil, ergab sich durch die Verwendung des Autoprefixers mit Grunt. Hiermit konnte im Basisstylesheet auf die Prefixes verzichtet werden.
Nachfolgende Tabelle veranschaulicht die Unterschiede zwischen den verschiedenen Stylesheets in Hinblick auf die Anzahl der Codezeilen:
Die Tabelle zeigt, dass CSS am Meisten Codezeilen benötigt um das selbe Resultat wie

Unterteilung Code	**CSS**	**SCSS**	**LESS**	**Stylus**
Leerzeilen	41	21	23	80
Codezeilen	182	150	148	116
Kommentarzeilen	5	2	2	2
Zeilen gesamt	228	173	173	198

Tabelle 2: Codeaufteilung der Stylesheets

SCSS zu erreichen. Sieht man sich die Anzahl der gesamten Zeilen an, steht Stylus an zweiter Stelle, was darauf fußt, dass hierbei durch das entfernen der Geschwungenen Klammern, die Mehrheit an Leerzeilen entsteht.
Die Tabelle kann jedoch lediglich für den Code, der sich aus dem pratkischen Teil der vorliegenden Arbeit ergibt, angenommen werden, da sich die Leerzeilen bspw. in Stylus erheblich ändern können, wenn das Stylesheet direkt in der Stylus-syntax verfasst wird.
Zumal in dieser Arbeit, die Konvertierung von SCSS in Stylus mit Regular Expressions umgesetzt wurde, entstehen einige Leerzeilen durch das Entfernen der Klammern und nachfolgendem einfügen einer Leerzeile.
Einige Leerzeilen werden jedoch für die Erstellung eines validen Stylesheets mit Stylus nicht benötigt.

Wie erläutert, konnte durch die Verwendung von CSS-Präprozessoren die Übersichtlichkeit des Codes verbessert werden. Neben der Übersichtlichkeit, wurde auch der zeitliche Aufwand bei der Verfassung des Stylesheets verkürzt. Durch die Verwendung von Mixins, mussten Abschnitte des Codes nicht wiederholt formuliert werden.

Durch die Verwendung von Grunt und einem CSS-Präprozessors, kann die konvertierte Datei jeweils als minified Datei konvertiert werden, wodurch die Performance der Webseite erhöht wird. Ein großer Vorteil von CSS-Präprozessoren ist, dass beim Kompilieren mehrere Dateien zu einer Datei zusammengefasst werden können. Dieser Umstand verbessert die Performance, durch den Umstand, dass nur eine CSS-Datei geladen werden muss. Trotzallem, kann in der Entwicklung mit mehreren Stylesheets gearbeitet werden.

Im Anschluss an das Erstellen des SCSS-Stylesheets für den praktischen Teil dieser Arbeit, wurde noch getestet wie die anderen CSS-Präprozessoren, welche für diese Arbeit relevant sind, in der Verwendung funktionieren.
Für die Verwendung von LESS, wurde in Kapitel 4.1 beschrieben, dass dieses auch direkt im Browser kompiliert werden kann. Bei der Überprüfung dieser Aussage, wurde das Ergebnis erzielt, dass die Kompilierung in neuen Browsern funktioniert, jedoch auch Nachteile mit sich bringt.
Bei der Kompilierung im Browser, dauert das Rendern durchschnittlich 1 sekunde. Im Vergleich dazu, benötigt eine HTML Datei in der schon das kompilierte Stylsheet eingebunden ist, ca. 200ms.
Natürlich muss beachtet werden, dass die CSS-Datei welche in die HTML-Datei eingebunden wird im Voraus kompiliert werden muss. Dieser Durchgang dauert mit Grunt in etwa 150ms, was jedoch davon abhängt wie groß die zu kompilierende LESS Datei ist.

Neben der längeren Renderzeit, birgt die Kompilierung im Browser noch den Nachteil, dass wenn ein Fehler auftritt, dieser nicht mit der Zeilennummer im Quellsourcecode ausgegeben wird.

Der allgemeine Vergleich von LESS und SCSS brachte den Schluss, dass beide CSS-Präprozessoren sowohl Vor- als auch Nachteile haben. Beide Präprozessoren helfen dabei, den Code übersichtlicher und im Falle dieser Arbeit auch kürzer zu halten.
Betrachtet man aber die Erstellung von Mixins, bringt LESS den Vorteil mit sich, dass keine Sonderzeichen verwendet werden müssen und das Mixin wie eine Klasse definiert wird. Bei SCSS ist es jedoch durch die Verwendung von „@mixin“ leichter erkennbar, dass es sich hierbei um ein Mixin handelt.

Auch der Vergleich zwischen SCSS und Stylus wurde durchgeführt. Dieser Vergleich erbrachte das Resultat, dass zwar der Quellcode in Stylus meist kürzer ist als jener in SCSS, jedoch muss der Verfasser des Stylesheets mit der Syntax von Stylus zurechtkommen.
Durch das Entfernen von den Semicolons und den geschwungenen Klammern, wird das Lesen des Stylesheets, nach Ansicht der Autorin, erschwert, da nur die Einrückungen und Leerzeilen Aussage darüber treffen, wo eine Klassendeklaration aufhört und die Nächste beginnt.
Jedoch kann dieser Umstand den Zeitaufwand des Verfassens verkürzen und so den Workflow verbessern.

Neben den genannten Vor-und Nachteilen, kam jedoch auch die Autorin dieser Arbeit zu dem Schluss, dass es keinen „Besten“ Präprozessor gibt und die Auswahl des richtigen Präprozessors jeder für sich treffen muss.

7 Schluss

Die vorliegende Arbeit befasste sich mit dem Thema „CSS-Präprozessoren“. Im Detail ging es in der Arbeit darum, die drei Präprozessoren „LESS“, „Sass“ und „Stylus“ genauer zu erläutern und zu beschreiben, wie diese funktionieren.
Um das Verständnis für CSS-Präprozessoren zu erleichtern, wurden zu Beginn der Thesis auch technische Begriffe wie „Workflow“, „Software Komponenten“ und „Parsing“ präzisiert.

In Kapitel 2 wurde ausführlich beschrieben, wie sich CSS-Präprozessoren definieren und wozu sie verwendet werden. Wie in dem Kapitel dargelegt, bereitet die Verwendung von Präprozessoren einige Vorteile in Bezug auf die Syntax und auf den Workflow, bei der Verfassung, des Stylesheets.

Desweiteren wurde im genannten Kapitel expliziert, was Workflow bedeutet, und wie dieser durch die Benützung eines CSS-Präprozessor gegenüber von herkömmlichen CSS, verringert werden kann.

Wie zu Beginn dieses Kapitels erwähnt, wurden ebenso die Begriffe „Software Komponenten“ und „Parsing“ beschrieben.
Unter dem Begriff „Parsing“ versteht man, im Zusammenhang mit der vorliegenden Arbeit, die Methode, mit der die CSS-Präprozessoren in CSS kompiliert werden.
Das Kapitel 2.4 erörterte sowohl Parsing im Allgemeinen als auch das Parsing bei den relevanten CSS-Präprozessoren „LESS“, „Sass“ und „Stylus“.

Nachfolgend wude in Kapitel 2.5 auf Software Komponenten eingegangen und beschrieben, in welchem Aspekt CSS-Präprozessoren als Software Komponenten zu sehen sind. Hierbei wurde das Hauptaugenmerk auf Mixins, welche eine Erweiterung von CSS-Präprozessoren darstellen, gelegt.

Im Anschluss an die Schilderungen zur Funktion von CSS-Präprozessoren, beinhaltet das Kapitel 3 eine Übersicht der relevantesten Erweiterungen zu den genannten Präprozessoren. Dazu wurden die Erweiterungen Variablen, Mixins, Funktionen, Vererbung und Schleifen bzw. logische Konditionen im Generellen verdeutlicht.
Diese Erweiterungen wurden in Kapitel 5 anhand der bereits genannten CSS-Präprozessoren „LESS“, „Sass“ und „Stylus“ erläutert.

Um die Implementierungen der Erweiterungen verständlicher darstellen zu können, wurde zuvor in Kapitel 4 genauer auf die Präprozessoren eingegangen und erklärt wie diese aufgebaut sind und umgesetzt werden.

Schließlich wurde nach den theoretischen Ausführungen zum Thema „CSS-Präprozessoren“, die Umsetzung, des praktischen Teils der Thesis erklärt.

7.1 Relevanz

Das Thema dieser Arbeit ist zum jetzigen Zeitpunkt im Bereich Web von großer Bedeutung.
Wie die Ergebnisse des praktischen Teils zeigen, kann mit CSS-Präprozessoren der Workflow beim Erstellen eines Stylesheets verbessert werden. Daraus ergibt sich, dass für die Umsetzung des Webdesigns weniger Zeit in Anspruch genommen wird. Dies wiederum ermöglicht es dem Entwickler, der Entwicklerin das Hauptaugenmerk auf andere Themen zu legen.

In weiterer Folge besteht die Relevanz des Themas in der Möglichkeit, mihilfe der CSS-Präprozessoren, Funktionen und logische Konditionen direkt im Styelsheet anzugeben und dadurch mit wenig Aufwand auf Browserunterschiede eingehen zu können.

7.2 Ausblick

Das zentrale Forschungsgebiet dieser Arbeit stellt die Verwendung von CSS-Präprozessoren dar. Es wurde auf die Forschungsproblematik der Erstellung von Stylesheets mit LESS, Sass und Stylus eingegrenzt.

In weiterer Folge ließe sich noch eruieren, welche weiteren CSS-Präprozessoren bereitgestellt werden und wie diese funktionieren. Hierbei könnte auch auf die Unterschiede zu den, in dieser Arbeit, beschriebenen Präprozessoren eingegangen werden.

In Hinblick auf die Thematik könnte in weiterer Folge eine spezifischere Auseinandersetzung mit dem Kapitel „parsing“ durchgeführt und die bestehenden Parser verglichen werden.

Neben den genannten theoretischen Weiterentwicklungen, bietet es sich an, auch für den praktischen Teil weitere Forschungen durchzuführen und bspw. zu analysieren ob und in welcher Weise der entwickelte Grunt-Task refaktoriert werden könnte.

Die hier erläuterten Forschungsfragen und Weiterentwicklungen stellen nur einige wenige dar, die sich aus dem Forschungsgebeit dieser Arbeit ergeben. Das Themengebiet wird auch in Zukunft von großer Bedeutung sein, da immer weitere CSS-Präprozessoren entwickelt und die bestehenden weiterentwickelt werden.

Abkürzungsverzeichnis

Abb.	Abbildung
z.B.	zum Beispiel
ca.	cirka
bzw.	Beziehungsweise

Abbildungsverzeichnis

1 Suchtrends in Google nach Sass, Less, Styuls und Css 1

Listings

1 erstellen eines Mixins (scss) 4
2 Code ohne Verschachtelung (css) 5
3 Code mit Verschachtelung (scss) 5
4 Verwendung Autoprefixer (css) 7
5 Code in Gruntfile für Autoprefixer 8
6 Beispiel package.json 9
7 Beispiel Gruntfile.js 9
8 variables.scss 18
9 style.scss 18
10 Mixin 19
11 Mixin Ausgabe 19
12 Mixin und Ausgabe ohne Mixin 20
13 Mixin mit Argument 20
14 Mixin mit @arguments 21
15 Mixin mit Selector 21
16 Vererbung mit extend (SCSS) 22
17 if/then statement und loop in LESS 23
18 Einbindung Less 24
19 Code in ursprünglicher Syntax 26
20 Code mit in scss Syntax 26
21 style.styl 28
22 Verwendung Variable in less 29
23 Verwendung Variable in sass 29
24 Verwendung Variable in stylus 30
25 Verwendung Mixin in Less 31
26 Verwendung Mixin in Sass 32
27 Verwendung Mixin in Stylus 32

28 Verwendung Mixin in Stylus . . . 33
29 ARGB in Less . . . 34
30 Saturation in Less . . . 34
31 Fadein in Less . . . 35
32 Mix in Less . . . 35
33 Average in Less . . . 35
34 Funktionen in Less . . . 35
35 Verwendung Funktion in Sass . . . 36
36 Verwendung Funktion in Stylus . . . 36
37 Vererbung in Less . . . 37
38 Vererbung in Sass . . . 37
39 Vererbung in Stylus . . . 38
40 Logic in LESS . . . 38
41 Loop in LESS . . . 39
42 if/then/else Schleife Sass . . . 39
43 for Schleife Sass . . . 39
44 while Schleife Sass . . . 40
45 each Schleife Sass . . . 40
46 if-else Statement Stylus . . . 40
47 if-else Statement Stylus . . . 40
48 for-in Schleife Stylus . . . 41
49 main.scss . . . 42
50 package.json . . . 43
51 Gruntfile.js (Scss zu LESS) . . . 44
52 index.js . . . 44
53 index.js . . . 45
54 Erstellung Grunt Task . . . 46
55 Erstellung Grunt Task . . . 47
56 Konvertierung Scss in CSS . . . 48

Tabellenverzeichnis

1 Parser Stack: Bottom Up Parser 16
2 Codeaufteilung der Stylesheets 50

Literaturverzeichnis

Andresen, Andreas. 2003. *Komponentenbasierte Softwareentwicklung: mit MDA, UML und XML.* München und Wien: Carl Hanser. ISBN: 3-446-22282-0.

Coyier, Chris. 2012. *Sass vs. Less.* Besucht am 26. Oktober 2014. http://css-tricks.com/sass-vs-less/.

Frost, Brad. 2013. *Atomic Design.* Besucht am 28. März 2015. http://bradfrost.com/blog/post/atomic-web-design/.

Garsiel, Tali. 2015. *How Browsers Work: Parsing General.* Besucht am 23. Februar. http://taligarsiel.com/Projects/howbrowserswork1.htm#Parsing_general.

Gerchev, Ivaylo. 2012. *A Comprehensive Introduction to Less: Mixins.* Besucht am 27. November 2014. http://www.sitepoint.com/a-comprehensive-introduction-to-less-mixins/.

Giraudel, Hugo. 2014a. *Architecture for a Sass Project.* Besucht am 27. März 2015. http://www.sitepoint.com/architecture-sass-project/.

———. 2014b. *What's the Difference Between Sass and SCSS?* Besucht am 2. November 2014. http://www.sitepoint.com/whats-difference-sass-scss/.

ITWissen.info. 2014. *Workflow.* Herausgegeben von DATACOM Buchverlag GmbH. Besucht am 8. Februar 2015. http://www.itwissen.info/definition/lexikon/Workflow-workflow.html.

Jung, Jean-Baptiste. 2010. *8 CSS preprocessors to speed up development time.* Besucht am 21. Dezember 2014. http://www.catswhocode.com/blog/8-css-preprocessors-to-speed-up-development-time#disqus_thread.

LearnBoost. 2010a. *Stylus: Expressive, dynamic, robust CSS.* Besucht am 21. Februar 2015. https://github.com/stylus/stylus/blob/master/lib/parser.js.

———. 2010b. *Stylus: Expressive, dynamic, robust CSS.* Besucht am 21. Dezember 2014. http://learnboost.github.io/stylus/.

Peter, Christian. 2012. *C-Präprozessoren.* http://wr.informatik.uni-hamburg.de/_media/teaching/wintersemester_2012_2013/epc-1213-peter-praeprozessor-praesentation.pdf.

psdtowp. 2014. *CSS preprocessors.* Besucht am 21. Dezember 2014. https://psdtowp.net/css-preprocessors.html.

Sellier, Alexis. 2014. *Less die dynamische stylesheet sprache.* Besucht am 21. Dezember. http://www.lesscss.de/.

———. 2015a. *Less Functions.* Besucht am 21. März. http://lesscss.org/functions/.

Sellier, Alexis. 2015b. *Less Parser.* Besucht am 21. März. https://raw.githubusercontent.com/less/less.js/master/lib/less/parser/parser.js.

Sitnik, Andrey. 2013. *Autoprefixer: A Postprocessor for Dealing with Vendor Prefixes in the Best Possible Way.* Besucht am 28. März 2015. https://css-tricks.com/autoprefixer/.

the core less team. 2014. *Language Features: Features of the Less language.* Besucht am 27. November. http://lesscss.org/features/.

Working with CSS Preprocessors. 2013. https://developer.chrome.com/devtools/docs/css-preprocessors.

Yard. 2014a. *Sass: Documentation.* Besucht am 24. März 2015. http://sass-lang.com/documentation/Sass/SCSS/Parser.html.

———. 2014b. *Sass: Documentation.* Besucht am 24. November 2014. http://sass-lang.com/documentation/file.SASS_REFERENCE.html#syntax.

Anhang