Kollaborative Frontend-Entwicklung durch webbasierte Real-Time Praktiken

Michael S. Kuss

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im September 2017

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Hagenberg, am 25. September 2017

Michael S. Kuss

Inhaltsverzeichnis

Kurzfassung

Diese Arbeit beschäftigt sich mit der Verbesserung und Erleichterung des gemeinsamen Arbeitens in einer Frontend-Build-Pipeline mittels Hinzufügens von Echtzeit-Aktualisierungen. Schon kleine Veränderungen im Code können Auswirkungen auf die visuelle Repräsentation einer Webseite haben, besonders wenn zum Entwickeln die Atomic Design Methodologie verwendet wird, da bei dieser die Komponenten aufeinander aufbauen. Beim Entwickeln von Frontend-Umgebungen im Team ist es wichtig, immer den aktuellsten Entwicklungsstand der visuellen Repräsentation begutachten zu können. Das Hinzufügen von Echtzeit-Aktualisierungen soll den EntwicklerInnen dabei helfen, die Übersicht zu behalten und den aktuellsten Zustand der Applikation abfragen zu können.

Schlagwörter: Frontend, Development, Collaboration, Real-Time, Atomic, Design, Node.js, Websockets, CSCW, StyleLab, Pimcore

Abstract

This Master thesis aims at improving and simplifying the collaborative work process with a frontend-build-pipeline by adding real-time functionality. Even little changes in the code affect the visual representation, especially when developing in the atomic design methodology, as lots of small parts depend on each other and changing one part affects the others. When developing frontend applications in teams it is important to be able to examine the latest stage of development of the visual representation. By adding real-time updates developers should be supported in getting an overall picture and in being informed about the latest state of the applications at all times.

Keywords: Frontend, Development, Collaboration, Real-Time, Atomic, Design, Node.js, Websockets, CSCW, StyleLab, Pimcore

Kapitel 1

Einleitung

Frontend-Build-Pipelines sind geregelte Abläufe, die in der Frontend-Entwicklung vorkommen und heutzutage nicht mehr wegzudenken sind. Sie beschreiben, vereinfachen und regeln den Entwicklungsprozess der ProgrammiererInnen und können bei der Automatisierung unterschiedlichster Tasks helfen. In dieser Arbeit geht es um die Optimierung und Automatisierung einer Frontend-Build-Pipeline, welche von EntwicklerInnen verwendet wird, um eine Website zu erstellen. Der praktische Teil der Arbeit wird in Kooperation mit der Firma elements[1] erstellt. Das Ziel dieser Kooperation ist es, die Frontend-Build-Pipeline und die Entwicklungsabläufe der Firma elements zu optimieren. Da beim Erstellen einer Webseite meist mehrere Personen gleichzeitig dieselbe Frontend-Build-Pipeline benutzen, wurde der Fokus auf das gemeinsame Arbeiten in einer Entwicklungsumgebung gelegt. So befindet sich die Arbeit im Spektrum des Computer Supported Cooperative Work (CSCW)[2], welches sich mit den Einflüssen von unterschiedlichen Technologien, die kollaborative Aktivitäten erleichtern, beeinträchtigen oder einfach verändern, beschäftigt [10, S. 43].

Die Partnerschaft mit der Firma elements stellt daher die Rahmenbedingungen für die praktische Arbeit dar. In Kapitel 5 wird auf diese Rahmenbedingungen, welche in Form von Entwicklungstools, wie zum Beispiel Pimcore vorliegen, genauer eingegangen. Ziel ist es, diese Tooling Pipeline mit Hilfe von Real-Time (Definition siehe Abschnitt 2.2) Aktualisierungen in ihrer Effizienz zu optimieren. Daraus resultiert die Forschungsidee.

1.1 Forschungsfrage

Ziel der Untersuchung ist es, herauszufinden, wie sich diese Real-Time-Funktionen auf die EntwicklerInnen auswirken. Es soll überprüft werden, ob und wie diese Funktion des automatischen Aktualisierens die EntwicklerInnen bei ihrer Arbeit und dem Prozess des Entwickelns und der Zusammenarbeit mit anderen EntwicklerInnen beeinflusst und unterstützt. Zur Untersuchung dieser Frage wurde eine prototypische Implementierung umgesetzt, mittels derer eine qualitative und quantitative Evaluierung der genannten Prinzipien erfolgen kann. Dieser Prototyp wurde in einer klar definierten Testumgebung

[1] elements.at New Media Solutions GmbH: www.elements.at

[2] computerunterstützte Gruppenarbeit

mit Hilfe von Probanden getestet. Voraussetzung für die Testpersonen war es, dass sie mit dem Erstellen von Webseiten vertraut und darin erprobt sind, damit sie sich leichter in dem zuvor erstellten Prototyp zurechtfinden können. Um diese Tests ausführlich auswerten zu können, fand im Anschluss an den Test eine quantitative Befragung statt.

1.2 Gliederung der Masterarbeit

In Kapitel 2 werden grundlegende Begriffe erklärt, um das Verständnis der Arbeit zu erleichtern. Kapitel 3 beschreibt den aktuellen Stand der Technik, um zu verstehen, wie an die Problemstellung herangegangen werden kann. Kapitel 4 bietet einen genaueren Einblick in die benötigten Technologien, um den Prototypen anzufertigen. In Kapitel 5 wird der abstrakte Aufbau der Umsetzung beschrieben. Im Kapitel 6 wird dann die technische Umsetzung erläutert. Das Kapitel 7 beschäftigt sich mit dem Erstellen und Auswerten des Fragebogens, den die Probanden im Anschluss an die Testphase bearbeiten sollten. Im abschließenden Kapitel 8 werden die Ergebnisse vorgestellt sowie mögliche Verbesserungen vorgeschlagen.

Kapitel 2

Begriffserklärung

In diesem Kapitel werden spezielle Begriffe erklärt, welche in der vorliegenden Arbeit verwendet werden.

2.1 Atomic Design

Atomic Design ist eine Methodik, die das Vorgehen modularer Design-Systeme beschreibt. Sie zeigt den Zusammenhang zwischen der gesamten Benutzeroberfläche und deren Einzelteilen, welche aufeinander aufbauen (siehe Abbildung 2.1). Diese Methodik wurde zuerst von Brad Frost in [3, S. 42] wie folgt beschrieben:

> Atomic design is a methodology composed of five distinct stages working together to create interface design systems in a more deliberate and hierarchical manner.

Im Atomic Design gibt es also, angelehnt an die chemischen Grundbausteine des Universums, fünf verschiedene Stufen, welche aufeinander aufbauen.

2.1.1 Atoms

Atoms, also Atome, bilden die erste Stufe des Atomic Designs und sind die Grundbausteine aller Benutzeroberflächen. Atome beinhalten grundlegende HTML-Tags wie `<input>`, `<button>` oder `<label>`. Sie können aber auch eigene elementare Bausteine sein, wie eine Kombination aus verschiedenen `<div>`-Tags, welche eine Grundeinheit bilden.

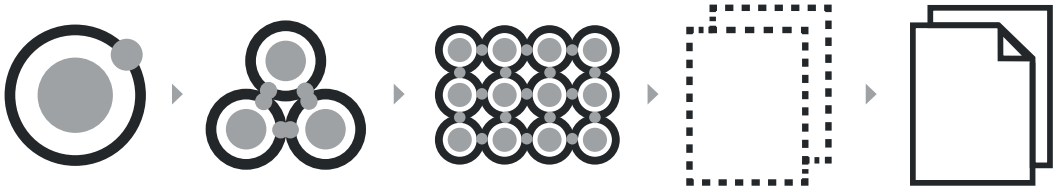


Abbildung 2.1: Atomic Design: Atome, Molekühle, Organismen, Templates und Pages bauen aufeinander auf, um ein Interface-Design-Systeme zu schaffen [3].

2.1.2 Molecules

Molecules, also Moleküle, sind die zweite Stufe in der Atomic Design Methodik. Dabei handelt es sich um relativ einfache Gruppen von UI-Elementen, die zusammen als Einheit funktionieren. Sie bestehen zu einem großen Teil aus Atomen und können zum Beispiel eine Kombination aus einem Input-Feld, einem dazugehörigen Label und einem Button bilden. Diese Kombination der Grundbausteine gibt ihnen einen größeren Sinn. In dem oben erwähnten Beispiel wäre das Label eine Beschreibung des dazugehörigen Input-Feldes und der Button hätte eine Bestätigungsfunktion dieses Feldes. Diese nun fertige Komponente kann jetzt auf den verschiedensten Benutzeroberflächen platziert und wiederverwendet werden. Das Erstellen solcher simplen Komponenten, die wiederverwendbar sind und eine gewisse Konsistenz bieten, lehnt sich an dem Single-Responsibility-Prinzip [7] an:

> A class should have only one reason to change.

2.1.3 Organisms

Organisms, also Organismen, können sowohl aus verschiedenen oder aber auch aus gleichen Molekülen bestehen als auch aus Atomen. Organismen sind eigenständige und wiederverwendbare Komponenten, die als eine eigene Einheit meist mit einer gewissen Funktion ausgestattet sind. Beispiele für Organismen sind ein Formular, eine Seiten-Navigation oder der Footer-Bereich.

2.1.4 Templates

Templates folgen nicht mehr der ursprünglichen Analogie zum chemischen Aufbau. Templates sind meist eine Ansammlung von Organismen oder auch Molekülen. Ein Template spiegelt eine fertige Seite eines Webauftrittes wider. Hier ist bereits das finale Design in Grundzügen zu erkennen.

2.1.5 Pages

Pages sind die Erweiterung von Templates. Bei Pages geht es darum, diese Templates mit richtigen Inhalten zu füllen, wie zum Beispiel mit Bildern oder Texten in der richtigen Länge. Pages spiegeln das fertige Design wider.

2.2 Web Real-Time

Real-Time, also Echtzeit, ist ein strittiger Begriff, der im Web eine andere Bedeutung als beim klassischen Software Engineering hat. In dem Buch *Hardware-dependent Software – Principles and Practice* [2] wird sie wie folgt beschrieben:

> Time means that the correctness of the system depends not only on logical results but also on the time the results are produced.

Das bedeutet also, wenn es um Real-Time Tasks geht, dass die Korrektheit einer Implementierung nicht nur von den vom Programm erzeugten Werten abhängt, sondern auch von der Zeit, zu der diese Werte erzeugt werden. Bei Real-Time geht es also darum, dass eine Aufgabe, die nicht zu einer bestimmten Zeit erfüllt wird, eine Fehlfunktion im System verursacht.

Web Real-Time hingegen sind Technologien und Praktiken, die es BenutzerInnen ermöglichen, Informationen zu erhalten, sobald sie von ihren AutorInnen oder Systemen veröffentlicht werden, anstatt zu verlangen, dass die UserInnen oder ihre Software regelmäßig bei der Quelle nach Updates nachfragen und überprüfen. Ein Beispiel für diese Real-Time-Technologien und -Praktiken ist WebRTC[1], das in seiner Standardisierung auch Real-Time genannt wird. Wenn es in dieser Arbeit um Real-Time geht, ist Web Real-Time gemeint.

[1] https://www.w3.org/TR/webrtc/, https://tools.ietf.org/html/draft-ietf-rtcweb-overview-18

Kapitel 3

Stand der Technik

Im folgenden Kapitel wird der Stand der Technik in Bezug auf diese Arbeit beschrieben. Zunächst wird das Prinzip von Style-Guide-Tools und Frontend-Build-Pipelines beschrieben, welche die Erstellung von Websites erleichtern sollen. Ebenso wird auf grundlegende Elemente der Kollaboration in diesem Bereich eingegangen. In Kapitel 4 werden die technischen Details zu diesen Bereichen beschrieben.

3.1 Style Guide Tools

Durch die rapide Entwicklung von Webseiten und deren immer größer werdenden Komplexität werden Tools benötigt, um trotz dieses Umfangs eine Konsistenz durch das Design und die Funktionalität, die eine einheitliche Linie und Darstellung beinhalten und eine Übersicht während der Entwicklung bringen, bieten zu können. Außerdem soll diese zu Beginn erstellte Linie auch für weitere Entwicklungen eine solide Grundbasis bieten. Wie Edward Tufte[1] sagte:

> Great design is not democratic; it comes from great designers. If the standard is lousy, then develop another standard.

Tufte sagt somit, dass großartiges Design nicht demokratisch ist. Es kommt von großartigen Designern. Wenn der Standard miserabel ist, dann soll ein anderer Standard kreiert werden. Style Guides helfen dabei, eine Übersicht über alle Komponenten zu bieten und einen soliden Standard zu entwickeln. Im besten Fall können sie sich während des Entwicklungsprozesses der Webseite anpassen und verschiedene Kombinationen des Interfaces widerspiegeln. Sie sind also eine Sammlung aller Komponenten und folgen oft der Atomic Design Methodik (siehe Abschnitt 2.1).

Während des Wartungsprozesses einer Webseite können durch zuvor sorgfältig erarbeitete Style Guides neue Komponenten aus den schon bestehenden Komponenten erstellt werden. Style Guides helfen außerdem, diese neu erstellten Komponenten zu überprüfen, um evaluieren zu können, ob sie der ursprünglichen Linie entsprechen. Sie tragen somit deutlich zur Qualitätssicherung bei.

Es gibt viele Beispiele für solche Style Guides, welche sich sehr in ihrer Anwen-

[1] Edward Tufte, zitiert im Smashing Magazine von Kat Neville [13].

dung unterscheiden. Styleguides.io[2] hat eine Liste dieser unterschiedlichen Style Guides erstellt. Bootstrap[3] ist wohl einer der meist verbreitetsten Style Guides, jedoch wird er oft nicht als solcher wahrgenommen. Aber auch Patternlab.io[4] gehört zu den wohl bekannteren. Patternlab.io nutzt die Atomic Design Methodik als Grundlage für den Aufbau der Komponenten und richtet sich vor allem an selbst erstellte und sich noch im Entwicklungsstadium befindende Komponenten, im Gegensatz zu Bootstrap, welches eine schon fertige Bibliothek an Bausteinen bietet, die nach Vorlieben angepasst werden können.

3.2 Frontend-Build-Pipeline

Der Bereich der Web Frontend Entwicklung verändert sich rapide. Durch das Verschieben der Aufgabenbereiche in der Webentwicklung wird dem Frontend immer mehr Arbeit zugemutet. Ob es komplexe Precompiler zum Entwickeln von kompatiblen CSS-Codes sind oder das Verwalten und Rendern von Website-Komponenten, das Entwickeln von Webseiten ist mit immer mehr Arbeit im Frontend verbunden.

Täglich erscheinen neue Tools und Frameworks, welche das Zusammenbauen und Verwalten von Abhängigkeiten, das Kompilieren, Zusammenfügen oder Minimieren von Codes und Assets besser machen und so den EntwicklerInnen, so scheint es, das Leben leichter machen sollen. Tools wie jQuery[5], Bootstrap[6], Sass[7], Bower[8] oder Gulp[9] zählen heutzutage zum Stand der Technik und sollen es einfacher machen, die neuen Aufgaben für das Entwickeln des Frontends zu meistern. Im Abschnitt 4.3 werden die technischen Grundlagen der Frontend-Build-Pipelines beschrieben und es wird genauer auf den Vorgang der Automatisierung des Bereiches eingegangen.

3.3 Kollaboration in der Frontend Entwicklung

Hier ist mit Kollaboration bei der Entwicklung nicht die Absprache über Kommunikationsdienste wie Slack[10] oder Skype[11] oder das Projektmanagement gemeint, sondern das gemeinschaftliche Entwickeln eines Quellcodes.

3.3.1 Versionsverwaltungssysteme

Wenn es um Kollaboration bei der Web-Entwicklung geht, steht meistens das Versionsverwaltungssystem Git[12] im Fokus. Git ist ein kostenloses und freies System zur Versionsverwaltung von Dateien in allen Größen, welches von Linus Torvalds initiiert

[2] http://styleguides.io/tools.html
[3] http://getbootstrap.com/
[4] http://patternlab.io/
[5] https://jquery.com/
[6] http://getbootstrap.com/
[7] http://sass-lang.com/
[8] https://bower.io/
[9] http://gulpjs.com/
[10] https://slack.com/
[11] https://skype.com/
[12] https://git-scm.com/

wurde. Es unterscheidet sich von typischen Versionsverwaltungssystemen. Grundsätzlich könnte man den kollaborativen Entwicklungsprozess einer Frontend-Umgebung in zwei verschiedene Typen unterteilen:

1. **Remote**: Bei einem zentralisierten System liegt die Entwicklungspipeline auf einem Server. Änderungen durch EntwicklerInnen werden auf diesen Server über FTP oder mit Hilfe eines Versionsverwaltungssystems hochgeladen und die Pipeline übernimmt, wie zuvor beschrieben, das Zusammenbauen und Verwalten von Abhängigkeiten, Kompilieren, Zusammenfügen oder Minimieren von Codes und Assets. Die visuelle Repräsentation lässt sich dann von allen EntwicklerInnen über diesen Webserver, der dies rendert, analysieren.
2. **Lokal**: Das Problem bei Systemen wie Git ist, dass sie immer nur die lokale Version zeigen. Wenn man nun als FrontendentwicklerIn ein Interface erstellt, sieht man die lokale Repräsentation, falls man auf einem lokalen Entwicklungsserver arbeitet.

3.3.2 Real-Time Frontend-Entwicklungstools

Es gibt einige Tools, die man als Frontend-Kollaborationstools bezeichnen kann. Sie unterscheiden sich jedoch sehr stark voneinander. Wie anfangs beschrieben, sind keine Kommunikationsdienste gemeint. Die meisten dieser Tools beinhalten einen Code-Editor, welcher das gemeinschaftliche Bearbeiten von Dateien ermöglicht, jedoch keine visuelle Repräsentation oder ein automatisches Aktualisieren dieser Repräsentation des HTML/PHP-Dokuments bietet. Andere wiederum bieten ein automatisches Aktualisieren des HTML/PHP-Dokuments mit CSS an, aber keine Möglichkeit der Repräsentation. Floobits ist eines der Tools, welches mit leichten Modifikationen beides anbieten könnte.

Floobits

Flootbits[13] ist ein kollaboratives Online Real-Time Codebearbeitungstool. Dieses Tool erlaubt es, im Browser eine IDE zu starten. Flootbits ermöglicht es NutzerInnen, gleichzeitig dieselben Dateien zu bearbeiten. Da dieses Tool nicht nur für das Entwickeln von Web-Applikationen geeignet ist, sondern auch für Java-Anwendungen oder Programme in anderen Programmiersprachen, ist es nicht darauf ausgelegt, die zu rendernde Webpage anzuzeigen. Daher ist auch kein Mechanismus des automatischen Aktualisierens dieses Views integriert. Das bedeutet, dass Flootbits nur die grundsätzliche Codebase zwischen Nutzern synchronisiert und somit ein Ersatz für Git ist. Flootbits besitzt außerdem eine große Anzahl von Plugins für die meisten gängigen Desktop-IDEs. Mit Hilfe dieser Plugins kann Flootbits auch außerhalb der Browserumgebung genutzt werden. Dies macht die Nutzung der Tools sehr attraktiv, denn hier können wiederum andere Tools verwendet werden, um das automatische Aktualisieren der Views im Browser zu ermöglichen. Dies ist jedoch durch das kontinuierliche Aktualisieren der Datei bei Tastendruck eines Benutzers eher unpraktisch.

[13] https://floobits.com

Kapitel 4

Technische Grundlagen

In diesem Kapitel werden die technischen Grundlagen erklärt, die zum weiteren Verstehen der Arbeit benötigt werden. Neben Pimcore sind dies Node.js, Frontend-Build-Pipelines, also Build Tools, und Websockets.

4.1 Pimcore

Pimcore ist ein Open-Source-Projekt, das Content-Management-System (CMS), Product-Information-Management (PIM), Digital-Asset-Management (DAM) und E-Commerce[1] Funktionen in einem System zusammenfasst. Es baut auf PHP und MySQL auf und ist somit eine Webanwendung. Pimcore selbst basiert auf dem Symfony-Framework und arbeitet mit ext-Komponenten. Für diese Arbeit wird Pimcore in Version 4.4.1 verwendet. Pimcores Dependency-Management wird mit Composer verwaltet.

4.1.1 Pimcore Module

Beim Aufbau von Lösungen mit Pimcore beginnt man normalerweise mit dem Konfigurieren eines Objektdatenmodells, dem Erstellen von benutzerdefinierten Regeln, Aktionen und Vorlagen, dem Erstellen von Dokumenten und so weiter. Vieles ist möglich, ohne Pimcore selbst zu erweitern. Aber je nach gewünschtem Ergebnis ist es manchmal notwendig, die Standardfunktionalität von Pimcore zu erweitern. Das Erstellen von Pimcore-Plugins ist notwendig, wenn man komplexe und umfangreiche Funktionen zu Pimcore hinzufügen möchte. Ein Plugin kann einerseits eine Bibliothek von Funktionen sein, welche wiederverwendbar sind. Andererseits kann es dazu verwendet werden, die Pimcore-Backend-Funktionalität zu erweitern. Außerdem kann auch die Pimcore-Backend-Benutzeroberfläche mit Plugins verändert und erweitert werden, indem JavaScript UI-Hooks verwendet werden.

[1] Transaktion des Kaufens oder Verkaufens online

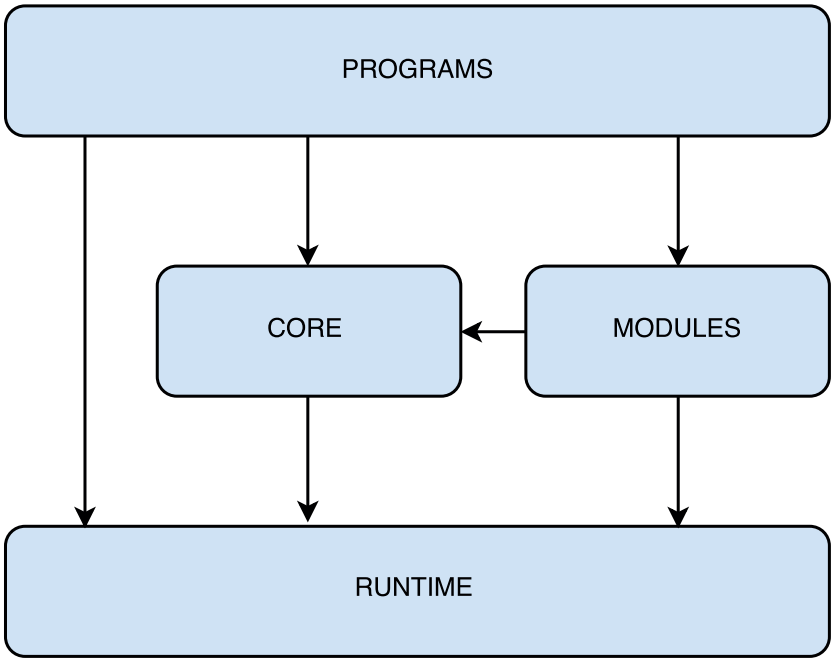


Abbildung 4.1: Node.js Ecosystem [1].

4.2 Node.js

Node.js[2] ist eine Plattform, die auf Googles V8 JavaScript-Engine basiert. Diese Plattform ist geeignet für das Erstellen von schnellen, skalierbaren Netzwerk-Applikationen. Node.js verwendet ein Event basiertes, nicht blockendes I/O Modell, welches schnelle und datenintensive Echtzeit-Anwendungen auf unterschiedlichsten Geräten ermöglicht. Node.js macht es also möglich, den JavaScript Code, welcher normalerweise nur in einer Browserumgebung läuft, als eigenständige Anwendung laufen zu lassen. Das gibt Front-end-EntwicklerInnen den Komfort, serverseitige Anwendungen zu schreiben, ohne eine weitere Programmiersprache zu lernen. Das Node.js Ecosystem besteht aus folgenden Komponenten:

- **Runtime** bildet die Basis von Node.js (siehe Abbildung: 4.1) und besteht, wie schon erwähnt, aus einer Verzweigung (fork) der Google V8 JavaScript-Engine[3]. Diese Basis bietet zusätzlich grundlegende API Funktionalitäten, exklusive den Komponenten, die für den DOM geschrieben wurden.
- Der Node.js **Kern** erweitert das Grundsystem um I/O Funktionalitäten nahe des Betriebssystems und fügt zusätzliche APIs hinzu, wie **Dateizugriff**, **Speichermanagement** und **Netzwerkprotokolle**.
- **Module** sind das Grundkonzept, auf dem Node.js aufbaut. Jedes Projekt, welches meistens wieder auf andere Module aufbaut, kann wieder als ein Modul gebündelt werden, welches andere EntwicklerInnen für ihre Programme und Module wei-

[2] https://nodejs.org/
[3] https://chromium.googlesource.com/v8/v8.git

ter verwenden können. Für das einfache Wiederverwenden und Installieren dieser Module und ihrer Abhängigkeiten gibt es NPM (siehe Abschnitt 4.2.1).

- **Programme** sind der eigentliche Teil der Anwendung, die entwickelt wird. Dieses Programm kann dann auch wieder als Modul weiterverwendet werden.

4.2.1 NPM

NPM[4] ist ein Paket Manager, welcher Abhängigkeiten verwaltet und installiert. Mit NPM können die zuvor genannten Module (siehe Abschnitt 4.2) über das Internet bezogen werden. Wenn die Module von weiteren Submodulen abhängig sind, werden auch diese bezogen. NPM wird meistens in Verbindung mit Node.js Projekten verwendet.

`package.json`

Alle NPM-Packages beinhalten eine Datei, `package.json`, welche die Metadaten und relevante Projektinformationen beinhaltet sowie die Abhängigkeiten von anderen Modulen beschreibt. Diese Datei befindet sich im Normalfall im Projektroot.

```
1
2 {
3   "name": "node-sass", //Name des Packages
4   "version": "4.5.3", // Version des Packages
5   ...
6   //Modulabhängigkeiten zu anderen NPM-Packete:
7   "dependencies": {
8     "glob": "^7.0.3",
9     "lodash.assign": "^4.2.0",
10    "mkdirp": "^0.5.1",
11    "node-gyp": "^3.3.1",
12    "request": "^2.79.0",
13    "sass-graph": "^2.2.4",
14    ...
15  },
16  //Modulabhängigkeiten zu anderen NPM-Packete während der Entwicklung:
17  "devDependencies": {
18    "eslint": "^3.4.0",
19    "fs-extra": "^0.30.0",
20    "mocha": "^3.1.2",
21    "object-merge": "^2.5.1",
22    "rimraf": "^2.5.2",
23    "sass-spec": "3.5.0-1",
24    "unique-temp-dir": "^1.0.0"
25    ...
26  }
27 }
```

[4] https://www.npmjs.com/

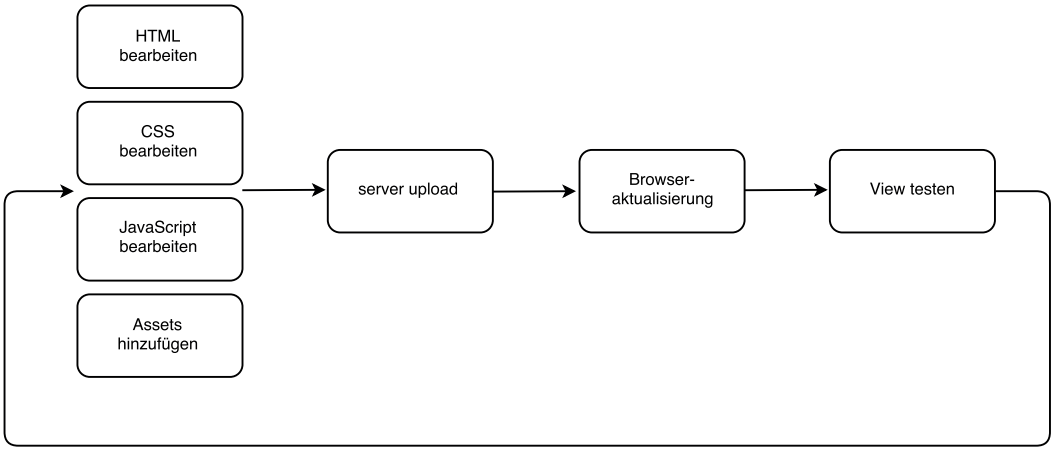


Abbildung 4.2: Klassische Frontend Pipeline. Abbildung inspiriert von [4]. Sie zeigt den zyklischen Entwicklungsablauf beim Entwickeln eines Frontends.

4.2.2 Middlewares

Eines der Kernelemente beim Entwickeln von Node.js Applikationen sind Middlewares. Middlewares sind Funktionen, die Zugriff auf das Request-Objekt, das Response-Objekt und die nächste Middlewarefunktion im Request/Response-Zyklus der Applikation haben. Die eingehängten Middlewares sind verkettet. Jede Middleware kennt die nächste und ruft diese beim Beenden oder mit dem Befehl `next();` auf.

4.3 Frontend Build Tools

Eine Frontend Pipeline beschreibt den Ablauf des Erstellens eines Frontends. In diesem Ablauf können auch Tools verwendet werden, um diesen zu vereinfachen. Diese Tools sind meist in Node.js geschrieben. Hier gibt es eine große Anzahl an vorgefertigten Hilfsmodulen, welche über NPM installiert werden können. Im Normalfall ist diese Pipeline dem Workflow des Entwicklers bzw. der Entwicklerin angepasst und daher werden die über NPM bezogenen Node.js Module um den eigenen JavaScript Code erweitert.

4.3.1 Klassische Frontend Pipeline

Abbildung 4.2 zeigt einen klassischen Frontend-Workflow. Wie in dieser Abbildung zu sehen ist, werden beim Aufbauen einer Website immer die gleichen Schritte wiederholt. Doch die moderne Frontend-Entwicklung wird immer komplexer und es werden Tools genutzt, um die vorhandenen Technologien um zusätzliche Funktionen zu erweitern.

Bei HTML kommen immer öfter Templating-Sprachen zum Einsatz, um die verschiedensten HTML-Grundbausteine zu erstellen und miteinander zu verknüpfen. Auch bei der Erstellung von CSS-Codes werden immer mehr Zusatztools, wie SASS oder LESS, verwendet, um die Grundfunktionen von CSS zu erweitern und das Warten des Codes zu vereinfachen. Auch beim Erstellen von JavaScript-Codes muss zum Beispiel der ES6-Code auf den ES5-Code transpiliert werden. Bei einem Transcompiler (auch als Transpiler oder Quer-Übersetzer bezeichnet) handelt es sich um einen speziellen

Tabelle 4.1: Frontend tooling survey 2016 von Ashley Nolan aus dem Jahr 2016 [15].

Task Runner	Number of Votes	Percentage	% Diff (to 2015)
Gulp	2.060	43.69%	-0.1%
NPM Scripts	1.223	25.94%	+22.78%
Grunt	554	11.75%	-15.81%
Make	54	1.15%	N/A
GUI Application (i.e. Codekit)	93	1.97%	N/A
Others	214	4.54%	-0.34%
I don't use a task runner	517	10.97%	-8.56%

Compiler, der den Quellcode einer Programmiersprache in den Quellcode einer anderen Programmiersprache übersetzt, zum Beispiel von Pascal in C Fenton [11]. Hier sollten auch alle Konsolen- und Debugger-Statements aus den Skripten entfernt werden. Meist wird für jede Komponente eine eigene CSS- und JavaScript-Datei erstellt. Um die Ladegeschwindigkeit der Website zu verbessern, sollten diese Dateien auch noch zusammengeführt werden, um bei einem Aufruf nur jeweils eine CSS- und JavaScript-Datei auszuliefern. Am besten sollte der daraus resultierende Code auch noch validiert werden. Der Prozess der Validierung wird oft `linting` genannt. Mehr dazu folgt im Abschnitt 4.3.2. Als letzten Schritt vor dem Upload auf den Server müssen die Assets noch für Webseiten optimiert werden. Hierbei handelt es sich meistens um Bilder, die in Qualität, Größe und Codierung optimiert werden können. Danach werden die Dateien auf den Entwicklungs-, Staging- und Produktionsserver hochgeladen. Im Normalfall werden anschließend die bearbeiteten Dateien über einen Browser vom Server geladen und evaluiert, um so zu ermitteln, ob das gewünschte Ergebnis erzielt wurde. Dieser Prozess wird immer wieder wiederholt. Anfangs kann dies noch ein leicht überschaubarer Prozess sein, doch nach einer gewissen Zeit wird er ermüdend und es können sich Fehler einschleichen. Um diesen Prozess zu automatisieren, gibt es sogenannte Task-Runner oder Build-Tools.

4.3.2 Build Tools

Das Erstellen einer automatisierten Build-Pipeline mit Hilfe von Build-Tools ist anfangs ein zeitaufwendiger Prozess. Dadurch, dass dieser aber oft wiederholt wird, kann im Endeffekt jedoch viel Zeit gespart werden. Des Weiteren wird durch die Automatisierung des Prozesses die Fehleranfälligkeit drastisch reduziert. Dieser Bereich der Frontend-Entwicklung ist wegen der sich wiederholenden Tätigkeiten eine der frustrierendsten Arbeiten. Es gibt mittlerweile die verschiedensten Build-Tools, die alle ihre individuellen Vor- und Nachteile besitzen. Die zwei bekanntesten sind sicher Gulp und Grunt. Diese beiden Tools bieten eine Grundlage für das Bauen einer Pipeline. Im Jahr 2015 haben nach Umfragen von Ashley Nolan über 70% der Frontend-WebentwicklerInnen eines dieser beiden Tools benutzt [14]. Da Gulp etwas später erschienen ist, hat es einige Probleme von Grunt schon ausgemerzt, wie zum Beispiel Schwächen beim Speichern von

Programm 4.1: Ausschnitt aus einem typischen gulpfile.js. Definition der Module und Pfade.

```
1 var gulp = require('gulp'),
2     concat = require('gulp-concat'),
3     uglify = require('gulp-uglify'),
4     ...;
5
6 var paths = {
7     scripts : 'build/js',
8     scriptsSrc : [
9         src/js/**/*.js',
10        src/libs/js/**/*.js',
11    ],
12    ...
13 }
```

Dateien oder beim Managen der Abhängigkeiten. Mittlerweile sind fast alle Abhängigkeiten und Plugins für Build-Tools auf NPM zu finden. Natürlich sind Gulp und Grunt nur zwei von vielen Build-Tools. Wie Tabelle 4.1 zeigt, sind viele EntwicklerInnen relativ schnell von Grunt abgesprungen. Auch NPM-Skripte ohne ein vorgefertigtes Build-Tool zu benützen, wird immer populärer. In dieser Arbeit wurde Gulp verwendet.

Gulp

Gulp besitzt, wie die meisten Build-Tools, eine Konfigurations-Datei, welche im Normalfall im Root-Verzeichnis des Projekts liegt. Es ist an dem Namen `Gulpfile.js` zu erkennen. In dieser Konfigurations-Datei wird zuerst Gulp instanziert, wie im Programmbeispiel 4.1 zu sehen ist. Danach werden die benötigten Module geladen. Als Nächstes werden meistens Variablen angelegt, um die Projektstruktur zu beschreiben und verschiedene Pfade anzulegen, in welchen Dateien verändert oder kopiert werden können. Nun werden Tasks angelegt, welche die Dateien verarbeiten können. Mehr zu Tasks folgt ab Abschnitt 4.3.2.

Um Gulp verwenden zu können, muss die Gulp CLI installiert werden. Sie macht Gulp als globale Variable und ausführbaren Befehl verfügbar. Zunächst müssen Node.js und NPM auf dem System installiert sein. Detaillierte Informationen zur Installation von Node.js und NPM sind auf der Website www.nodejs.org zu finden. Um zu überprüfen, ob diese installiert und dem System bekannt sind, kann der Befehl `node -v` oder `npm -v` auf einer Bash, Terminal oder der Windows command Promt eingegeben werden. Sollten diese beiden Befehle verfügbar sein, kann nun die Gulp CLI mit dem Befehl `npm -install gulp-cli -g` global installiert werden. Über den Parameter `-g` installiert npm das Paket global für das System. Sobald NPM die Installation beendet hat, kann über `gulp -v` überprüft werden, ob Gulp nun global verfügbar ist.

Wenn Gulp global verfügbar ist, wird für jedes Projekt eine lokale Installation benötigt, denn die globale CLI sucht nur nach der lokalen Installation und führt diese, wenn benötigt, aus. Die lokale Gulp Installation liest und führt die Instruktionen, welche im Gulpfile.js angelegt wurden, aus. Um es zu installieren, wird im Projektverzeichnis das

Programm 4.2: Ausschnitt aus einem typischen gulpfile.js. Dieses Beispiel zeigt einen typischen Gulp Task.

```
14 gulp.task('scripts', function() {
15  return gulp.src(paths.scriptSrc)
16  .pipe(concat('bundle.js'))
17  .pipe(uglify())
18  .pipe(gulp.dest(scripts));
19 })
```

NPM-Plugin mit dem Befehl `npm install gulp -save-dev` installiert. Der optionale Parameter `-save-dev` schreibt die Entwicklungs-Abhängigkeiten in die package.json-Datei des Projektes (siehe Abschnitt 4.2.1).

Tasks

Meist wird eine Datei durch mehrere dieser Tasks hintereinander geschickt. Jeder Task erfüllt genau eine Aufgabe und schickt das Ergebnis weiter an den nächsten Schritt. Häufig wird innerhalb eines Tasks nur ein Plugin aufgerufen. Da ein Gulpfile purer JavaScript-Code ist, kann innerhalb der Datei jeglicher JavaScript kompatible Code ausgeführt werden, was das System noch viel dynamischer gestaltet als andere Build-Tools. Gulp kann außerdem mehrere dieser Tasks parallel ausführen. EntwicklerInnen haben dadurch die Möglichkeit, von einander unabhängige Tasks nebeneinander auszuführen und somit Zeit zu sparen und zusätzliche Performance herauszuholen. Außerdem kann ein Task wiederum von anderen Tasks abhängen. Diese müssen dann in der richtigen Reihenfolge ausgeführt werden.

- `default`-Task ist der Task, der von Gulp aufgerufen wird. Er ist sozusagen die Main-Methode. Hier kann entweder der Task ausprogrammiert werden oder andere Tasks aufgerufen werden. Im Normalfall wird hier die gesamte Pipeline initialisiert und gestartet und läuft mindestens einmal durch. Die Nomenklatur der weiterführenden Tasks obliegt dem Programmierer bzw. der Programmiererin und ist reine Definitionssache, folgt aber meist der Konvention, um als Leitfaden dienen zu können.
- `build`-Task ist jener Task, welcher im Normalfall einmal einen Build-Prozess durchführt, um alle benötigten Dateien im *build*-Ordner für das Projekt zu erstellen. Dieser Task wird meistens vom `default`-Task aufgerufen und beinhaltet die meisten anderen Tasks.
- Der `clean`-Task wird benötigt, um den *build*-Ordner zu leeren. Dieser Task stellt sicher, dass es zu keinen falschen Zuständen im Ordner kommt, falls in diesem fälschlicherweise Dateien bearbeitet, gelöscht oder hinzugefügt wurden.
- `scripts`-Task ist, wie schon der Name sagt, der Task, in dem die JavaScript Aktionen ausgeführt werden. Hier kann zum Beispiel ES6 in ES5 mit Babel[5] transpiliert werden. Es kann auch das Tool Uglyfy[6] verwendet werden, um den

[5] www.babeljs.io/
[6] www.npmjs.com/package/uglify-js

JavaScript-Code anschließend zu parsen oder zu verkleinern. Danach werden noch alle JavaScript-Dateien in der richtigen Reihenfolge, meist mit dem NPM-Plugin concat[7], zusammengeführt und auf eine einzelne Datei geschrieben, welche dann im *build*-Ordner erstellt wird.

- Mit dem `styles`-Task werden alle .css-Dateien verkleinert. Zusammengeführt werden sie meist schon selbst in einer Hauptdatei mit `@import`-Statements. Danach wird diese Datei in den *build*-Ordner verschoben. Oft kommt hier auch ein Preprozessor, wie SASS oder LESS, zum Einsatz. Mit diesen Preprozessoren wird das ursprüngliche CSS um Funktionen wie Variablen oder Mixins sowie einfachere Verschachtelungen erweitert.
- Der `lint`-Task ist einer der selten verwendeten Tasks. Er ist jedoch einer der hilfreichsten, um qualitativ hochwertigen und effizienten und vor allem semantisch richtigen Code zu erzeugen, was bei JavaScript nicht immer gegeben ist. Meist wird das *jshint*[8] *package* von NPM verwendet.
- *Watcher*-Tasks sind eine spezielle Form von Tasks, welche in Gulp integriert sind. Genau wie bei allen Tasks wird hier ein Event-Emitter zurückgegeben, welcher auf Änderungen reagiert. Einfach ausgedrückt bedeutet dies, wenn irgendwelche Dateien geändert wurden, wird die Aufgabe ausgeführt.

  ```
  gulp.watch('source/javascript/**/*.js', ['lint']);
  ```

  Wie in dem obigen Codeblock zu sehen ist, wird der `lint`-Task ausgeführt, wenn sich Dateien in den Unterordnern von `source/javascript` ändern und diese die Dateiendung `.js` beinhalten.
- Es können natürlich auch andere Tasks entstehen, je nachdem, welche Aufgaben das Projekt oder welche Anforderungen der Entwickler bzw. die Entwicklerin an die Pipeline hat. Ein Beispiel wäre eine Template-Sprache für die HTML-Templates oder ein Task für das automatische Erstellen einer Iconfont aus einem Ordner voll mit SVG-Bilddateien.

4.3.3 Automatisierte Frontend Pipeline

Mit obig beschriebenen Gulp-Tasks lässt sich nun die in Abschnitt 4.3.1 beschriebene klassische Frontend-Pipeline mit automatisierten Komponenten erweitern. Abbildung 4.3 zeigt die klassische Frontend-Pipeline, erweitert mit Gulp-Tasks, welche blau eingefärbt sind. Die Bereiche mit Blau-zu-weiß-Verlauf zeigen Schritte in der Pipeline, welche durch Tasks automatisiert werden könnten, auf welche aber zuvor noch nicht eingegangen wurde.

Der Upload auf den Server ist nicht bei jedem Projekt zwingend notwendig. Wenn jedoch ein Server benötigt wird, kann dieser Upload über einen Gulp Task gemacht werden. Hier wird meist auch ein *Watcher*-Task benötigt, welcher auf Dateiänderungen reagiert und dann den Upload auf einen zuvor konfigurierten Webserver startet. In vielen Fällen wird der automatische Upload auch von der IDE selbst abgehandelt.

Das automatische Aktualisieren ist etwas diffiziler als in den Bereichen davor. Da das HTTP-Protokoll auf REST basiert, ist es ein zustandsloses Protokoll und kann von

[7] https://www.npmjs.com/package/gulp-concat
[8] www.npmjs.com/package/gulp-jshint

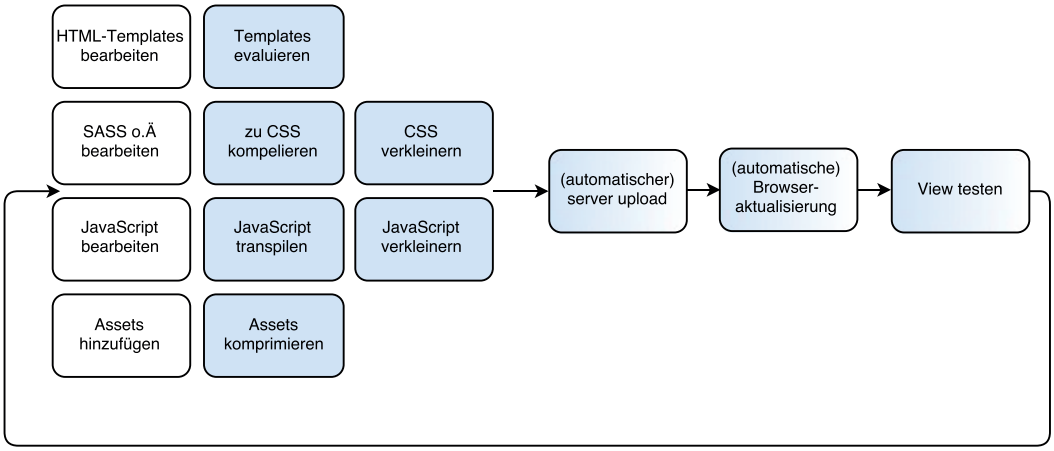


Abbildung 4.3: Automatisierte Frontend-Pipeline. Diese Abbildung zeigt die klassische Frontend Pipeline erweitert um automatisierte Komponenten, welche blau dargestellt werden.

Grund auf nicht auf veränderte Zustände am Server reagieren. Hier gibt es verschiedene Lösungsansätze, welche im Abschnitt 5.3.1 beschrieben werden. Diese Lösungen funktionieren meist entweder durch Polling, also durch dauerndes Abfragen am Server durch den Client, ob neue Daten bestehen oder mit Websockets (siehe Abschnitt 4.4).

Das Testen der Views ist eine der wichtigsten Komponenten in der Frontend-Entwicklung. Meist ist es nicht automatisierbar, denn es geht viel mehr um die visuelle Repräsentation der Komponenten, was schwer und nur bis zu einem gewissen Ausmaß von einem Tool zu erkennen ist. Was bei JavaScript zu testen ist, wurde bereits mit dem `lint`-Task beschrieben. Was auch getestet wird, ist die Repräsentation der Komponenten in verschiedensten Auflösungen bei unterschiedlichsten Geräten. Hier kann ein Style Guide helfen, welcher diesbezüglich unterstützende Funktionen zur Verfügung stellt.

Abbildung 4.4 zeigt ein Beispiel für ein Ablaufdiagramm des `default`-Tasks in einem Gulpfile, welches die Automatisierung der Frontend-Pipeline zeigt. Dieser Task wird zu Beginn der Entwicklung aufgerufen. Zuerst wird der `clean`-Task aufgerufen, welcher, wie zuvor beschrieben, für die Konsistenz der Daten im `/src`-Ordner zuständig ist. Danach werden der `scripts`- und `styles`-Task parallel zueinander ausgeführt. Dies ist möglich, da diese Tasks nicht von einander abhängen. Mit Hilfe dieser Tasks werden nun die Dateien in den `/src`-Ordner geschrieben. Nun wird ein `server`-Task gestartet, welcher mehrere *Watcher*-Tasks initialisiert, die auf Änderungen der JavaScript- und CSS-Dateien warten und bei Änderung den jeweiligen Task ausführen.

4.4 Websockets

Für eine bidirektionale Verbindung und stetigen Austausch von Daten zwischen Client und Server wurden Websockets geschaffen. Die vom *World Wide Web Consortium* (*W3C*) entwickelte Websocket-API basiert auf der Spezifikation der Internet Engineering Task Force (IETF) [12]. Das Websocket-Protokoll ist ein auf TCP-basierendes Protokoll, dessen Verbindungsaufbau wie bei HTTP vom Server als Upgrade-Request interpretiert wird. Der Client stellt die Verbindung zum Server über URL-Adressierung (z.B. ws://example.com/chat) her. Bei erfolgreichem Verbindungsaufbau können Daten als Frames gesendet und empfangen werden. Die gesendeten Daten werden als *messages* bezeichnet. *Messages* können aus mehreren Frames bestehen. Diese Frames bestehen aus einem Header und Payload. Um eine aktive Websocket-Verbindung zu schließen, gibt es einen Closing-Handshake.

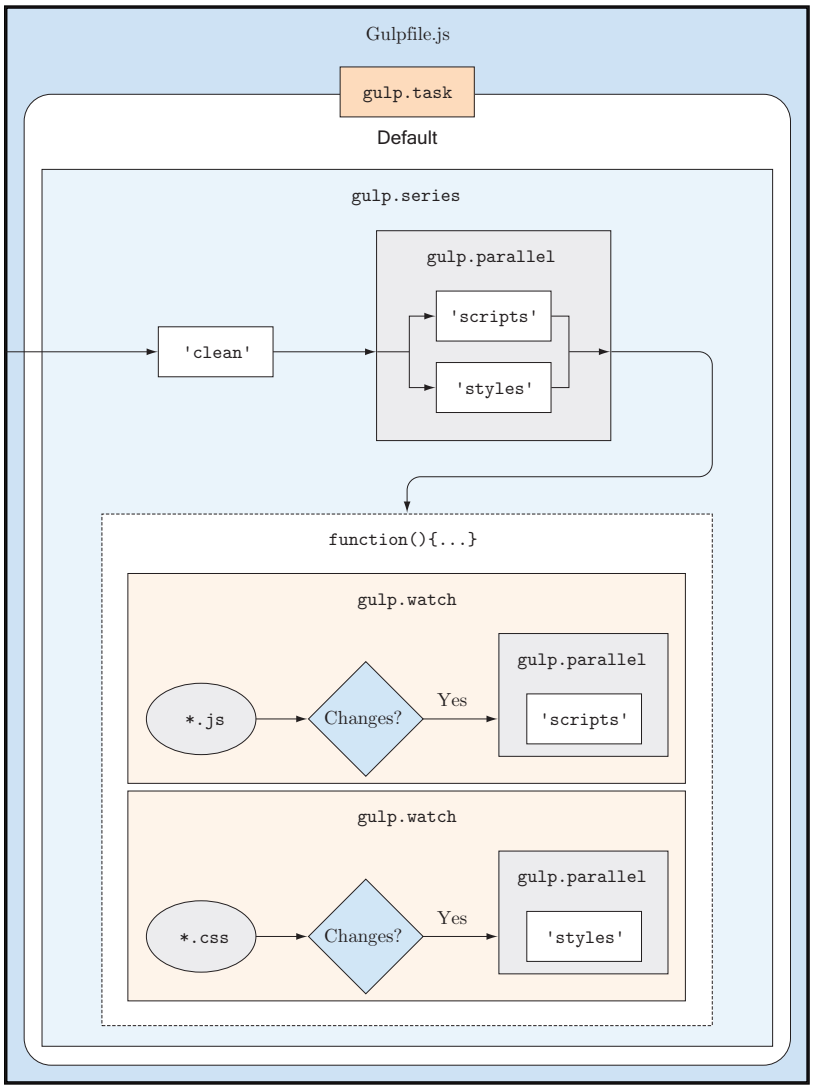


Abbildung 4.4: Gulpfile.js Task Ablaufdiagramm [1].

Kapitel 5

Konzept und Technisches Design

Die Aufgabe bestand darin, die vorhandene Frontend-Pipeline der Firma elements zu evaluieren und im Hinblick auf kollaboratives Arbeiten zu verbessern, ohne deren Workflow komplett zu ändern oder Tools zur Gänze umzuschreiben. Um diesen Task zu erfüllen, musste zuerst die bestehende Frontend-Pipeline der Firma elements analysiert werden.

5.1 Frontend-Pipeline der Firma elements

Alle Projekte des Unternehmens elements basieren auf Pimcore (siehe Abschnitt 4.1). Für das Entwickeln eines Frontends bei elements wurde daher ein eigenes Styleguide-Tool entwickelt. Dieses Tool nennt sich StyleLab und kann einfach als Pimcore-Plugin installiert werden.

5.1.1 StyleLab

Für die Vorlage zur Entwicklung des Tools wurde Patternlab verwendet. Dieses Tool richtet sich nach der Atomic Design Methodologie (siehe Abschnitt 2.1). Es ist also aufgeteilt in Atome, Moleküle, Organismen, Templates und Pages. Diese Komponenten bauen aufeinander auf. So gibt es für jede der Unterteilungen einen Ordner, in dem Module erstellt werden können. In den jeweiligen Modulordnern können weiter Unterordner erstellt werden, um Unterkategorien in den Modulen zu schaffen. Die Modulordner liegen im *html*-Ordner, welcher sich im Rootdirectory des Pimcore-Projekts befindet. Die Module darin sind in PHP geschrieben. Jedes Modul kann im anderen aufbauend geladen werden. So können Atome in Moleküle geladen werden. Außerdem können Attribute, welche im aufgerufenen Modul deklariert sind, überschrieben werden, wie im folgenden Codeabschnitt zu sehen ist.

```
<?= $this->pattern('atoms-input', ['placeholder' => 'EMAIL']) ?>
```

Das StyleLab gibt nun eine Übersicht über all diese Module und ihren Zustand. Es liest alle Dateien aus dem `html`-Ordner aus und listet sie auf. Abbildung 5.1 zeigt einen Screenshot der StyleLab-Anwendung. Hier sind die Unterteilungen in die verschiedenen Atomic Design Kategorien und deren Unterordner deutlich zu sehen. Um den Entwicklungsstand der Komponente anzuzeigen, können über ein einfaches PHP-Array im Sty-

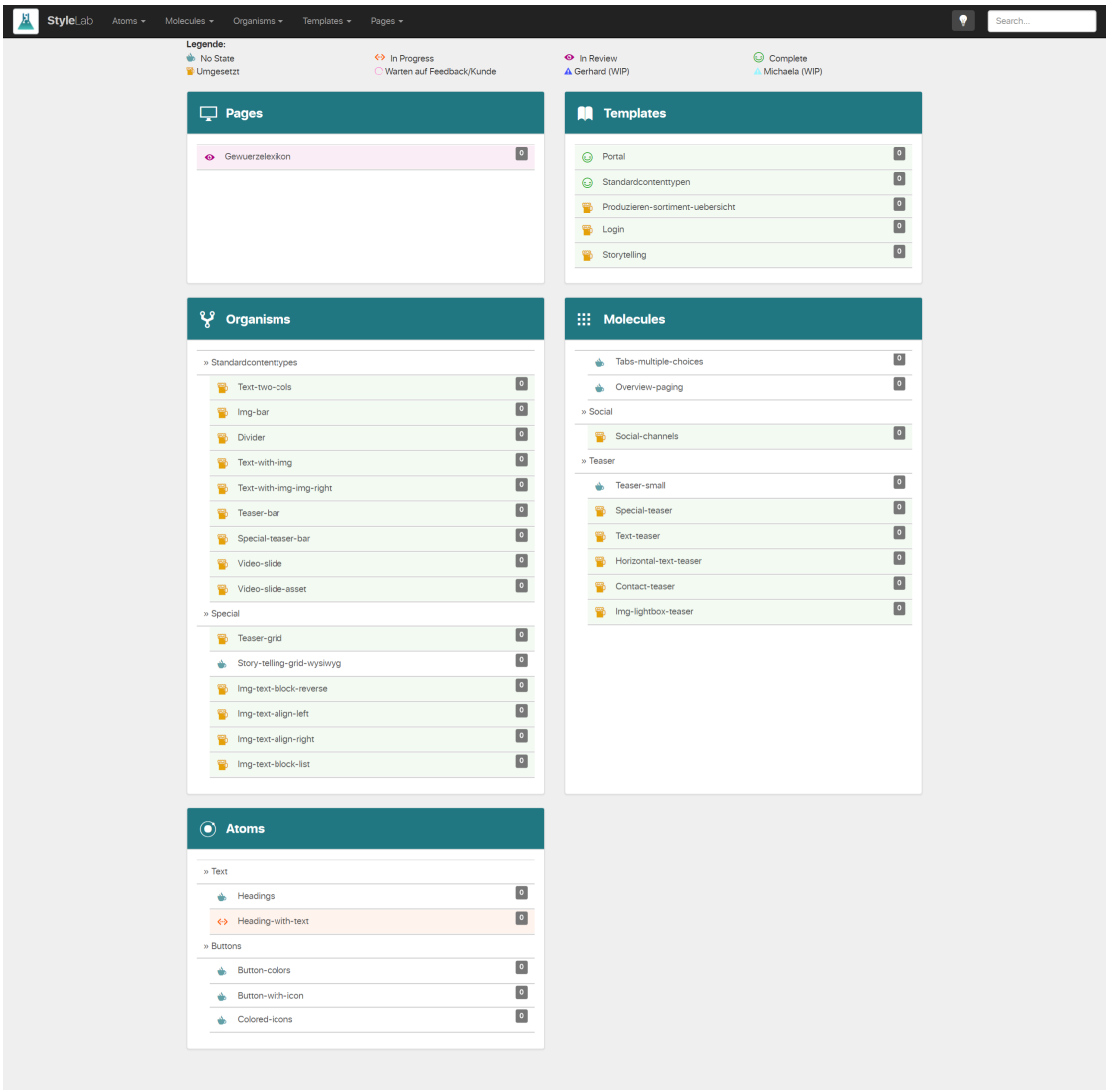


Abbildung 5.1: StyleLab-Home-Screen. Diese Abbildung zeigt nur ein Beispielbild der Anwendung. Im Normalfall sind weit mehr Elemente vorhanden.

leLab Plugin Zustände definiert werden. Die Legende zeigt die verschiedenen Zustände und ihre Bedeutung an.

Wenn auf ein Modul geklickt wird, wird dieses in einem eigenen View geladen (siehe Abbildung 5.2). Über die Navigationsleiste, welche an der Oberseite der Page zu sehen ist, kann schnell zu anderen Modulen gewechselt werden. In der View-Ansicht sind weitere Optionen in der Navigationsleiste zu finden. Die drei weißen Buttons neben der *Suche* sind dazu da, die Ansicht des Moduls in drei vordefinierten Größen in der Breite anzuzeigen. Über den <> -Button rechts oben im User-Interface lässt sich das unten sichtbare Codepannel anzeigen. Hier wird das Markup des Moduls angezeigt. Hier kann mit Buttons noch zwischen der `Source` und der `Rendered Source` gewechselt werden. Die `Source` zeigt den Markup Code an, welcher wirklich in der Datei steht, inklusive

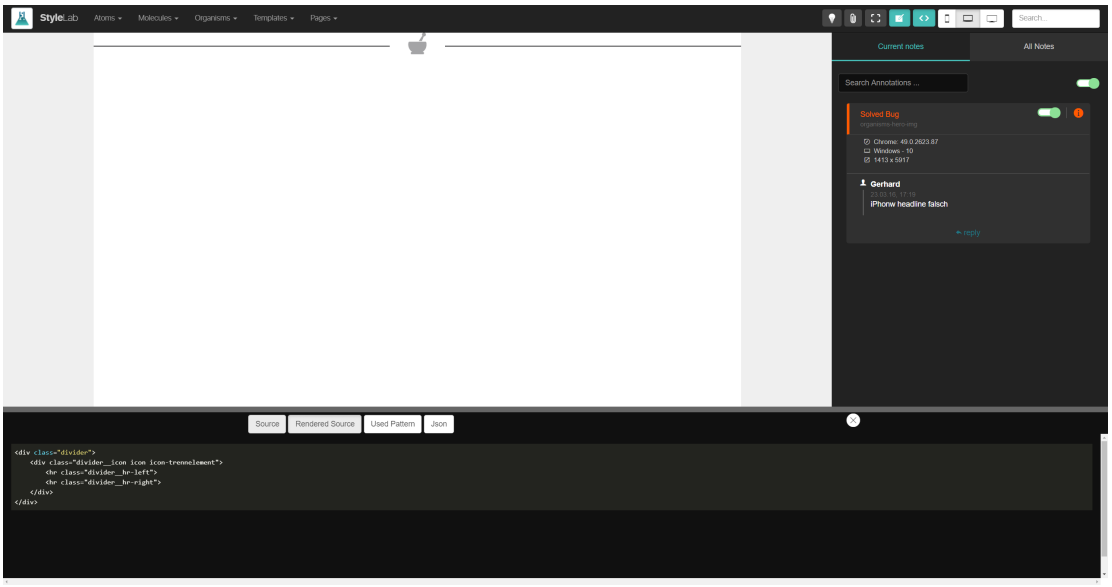


Abbildung 5.2: StyleLab Detailansicht einer Komponente.

dem PHP-Code. Bei der `Rendered Source` wird das fertig gerenderte Markup angezeigt, welches alle PHP-Includes schon besitzt. Falls andere Module in diesem Modul eingebunden wurden, werden diese angezeigt, wenn man auf `Used Patterns` klickt. Zusätzlich kann auf der gleichen Ebene wie das Modul eine Json-Datei angelegt werden, welche etwaige Datensätze, die über die Moduleinbindung wie zuvor beschrieben überschrieben werden können, bereitstellt. Mit dem -Button kann das Sidepannel geöffnet werden. Hier können Annotations, welche über eine andere Applikation auf eine Koordinate im Modul gesetzt werden können, angezeigt werden. Diese Annotations beinhalten einen Beschreibungstext, um ein Fehlverhalten des Views aufzuzeigen. Außerdem werden die Position der Annotation sowie die Browserversion und die Größe des Fensters mitgespeichert. Diese Annotationen können mit Hilfe eines Buttons auf *erledigt* gesetzt werden. Über den -Button lässt sich ein QR-Code aufrufen, um über ein Smartphone leicht auf die aktuelle Seite zu gelangen. Der -Button lässt den momentan geöffneten View außerhalb des StyleLab anzeigen. Über den -Button lassen sich alle vom Gulp-Task generierten Icons anzeigen (siehe Abschnitt 5.1.2 für Details zum `elements` Gulpfile).

Um zu einer Übersicht über alle Komponenten eines Ordners zu gelangen, kann in der Übersicht oder über die Navigation auf die Ordnernamen geklickt werden. Diese Ansicht beinhaltet einen Überblick über die Komponenten in dem jeweiligen Ordner, um mehrere Komponenten gleichzeitig anzuzeigen (siehe Abbildung 5.3).

5.1.2 Gulp

Die elments Frontend-Pipeline arbeitet auch mit Gulp und NPM. Dazu wird in jedem Pimcore ein vorgefertigtes Gulpfile erstellt. Dieses Gulpfile arbeitet mit NPM-Paketen. Wie in Abschnitt 4.3.2 beschrieben ist, gibt es einen `script`- und einen `css`-Task. Außerdem muss, wie beschrieben, ein `default`-Task vorhanden sein, welcher alle Tasks

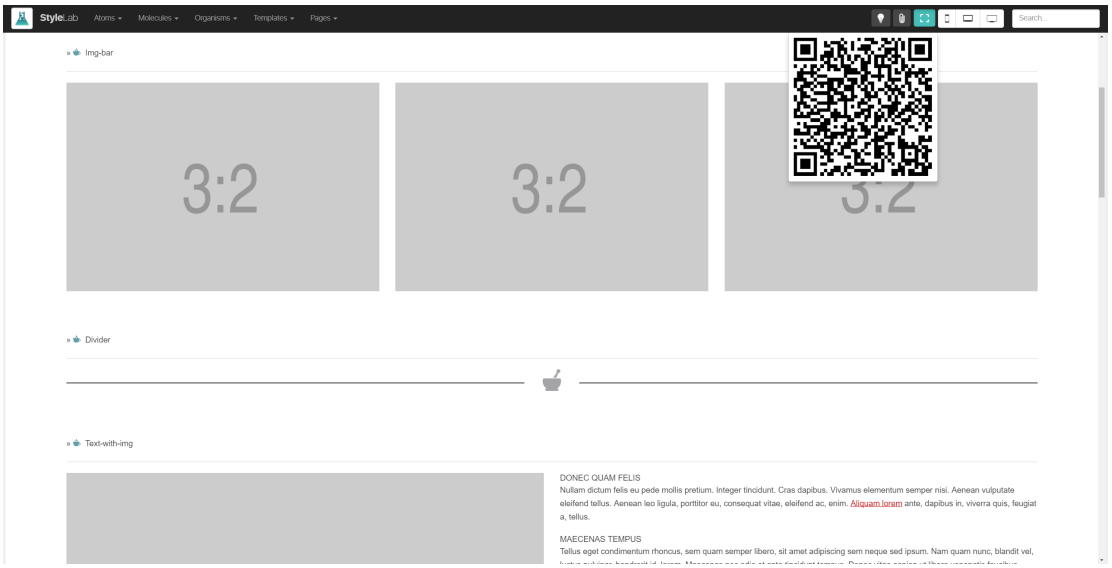


Abbildung 5.3: StyleLab Ansicht einer Untergruppe mit mehreren Detailansichten.

startet. Darüber hinaus gibt es auch einen `build`-Task. Hier wird der `build`-Task jedoch etwas anders verwendet, denn er führt im Voraus keinen `clean`-Task aus. Außerdem werden nur die Tasks ausgeführt, welche, wie bei dem `default`-Task, Dateien erstellen. Der einzige Unterschied besteht darin, dass diese Dateien anschließend in einen *build*-Ordner verschoben werden, um anzuzeigen, dass diese Dateien nicht mehr manuell verändert werden sollen. Zusätzlich gibt es noch zwei weitere Tasks. Der `images`-Task ist dazu da, statische Bilder, welche nicht über das CMS gewartet werden, mit dem NPM-Paket `gulp-image-optimization` zu optimieren und in einen Bilder-Ordner zu verschieben. Mit dem `iconfont-Task` können die SVG-Dateien, welche sich im Ordner `static/icons/svg/*.svg` des Pimcore Systems befinden, in eine Iconschriftart verwandelt werden. Diese eigens erstellte Schriftart beinhaltet dann die SVG-Dateien als Glyphen und kann innerhalb der Webapplikation benutzt werden. Hierzu wird eine CSS-Datei benötigt, welche die icons über CSS definiert. Sie können dann über die CSS-Klassen `icon` und eine weitere Klasse, welche das Icon spezifiziert (z.B. `icon-arrow`), aus der Schriftdatei, welche auch über den Task erstellt wird, geladen werden. Des Weiteren wird eine HTML-Datei erstellt, welche alle erstellten Icons ausliest und deren Verwendung dazu anzeigt. Diese HTML-Datei wird über das StyleLab beim Klicken auf den -Button geladen.

5.1.3 Gemeinsames Arbeiten

Um gemeinsames Arbeiten bei elements zu ermöglichen, wird ein Projekt `Remote` eingerichtet (siehe Abschnitt 3.3.1). Dies bedeutet, dass ein Entwicklungsserver mit Pimcore und dem StyleLab Plugin sowie der Gulp-Pipeline für jedes Projekt angelegt wird. Auf diesem Server wird der Gulp-Task ausgeführt, der auf Dateiänderungen reagiert. Jeder Entwickler und jede Entwicklerin hat über die IDE PhpStorm den Server als Remote-Host eingerichtet. Dies stellt die Konsistenz der Dateien zwischen den EntwicklerInnen

sicher. Wenn nun eine Datei verändert wird und ein Entwickler oder eine Entwicklerin dieses speichert, wird diese über die IDE automatisch auf den `Remote`-Server hochgeladen. Die Gulp-Pipeline reagiert dann auf diese Änderung und erstellt alle notwendigen Dateien für das Frontend. Die EntwicklerInnen können sich den Zustand der Dateien jederzeit im StyleLab über den Server ansehen und sehen so die erstellten Module aller beteiligten EntwicklerInnen.

5.2 Ansatz zur Verbesserung der Pipeline

Die Aufteilung und Funktionalität der Gulp-Tasks, welche im Gulpfile.js von elements definiert sind, erfüllen die Grundfunktionalitäten einer Frontend-Pipeline. Einzig der `build`-Task wäre zu verbessern und um einen `clean`-Task zu erweitern, damit eine bessere Trennung des ursprünglich entwickelten Codes und des zur Veröffentlichung bearbeiteten Codes gegeben ist. Dieser Eingriff auf die Pipeline wäre jedoch zu massiv, denn die Ordnerstruktur müsste dazu verändert werden. Auch das automatische Hochladen der veränderten Dateien ist über die IDE PhpStorm gegeben. Daher wird auf die Weiterentwicklung und Erweiterung der Funktionalitäten des StyleLab und dessen Design der Fokus gelegt, um die Arbeitsabläufe in Hinsicht auf die Zusammenarbeit der EntwicklerInnen zu verbessern.

5.3 Weiterentwicklung und Erweiterung der Funktionalitäten

Wie schon in Abschnitt 4.3.3 beschrieben wurde, ist das Testen der Views ein großer Bestandteil des Entwickelns. Zum Testen des Views muss der Entwickler bzw. die Entwicklerin immer manuell die Seite aktualisieren, um den neuesten Stand zu laden und somit die Veränderungen analysieren und überprüfen zu können. Beim gemeinsamen Entwickeln gestaltet sich die Aufgabe des Analysierens und Überprüfens schwieriger, insbesondere, wenn die einzelnen Komponenten, die entwickelt werden, auf einander aufbauen. Beim Aktualisieren der Seite können ungewollte Veränderungen auftreten, welche durch einen anderen Entwickler bzw. eine andere Entwicklerin verursacht wurden. Hier kann oft nicht auf den ersten Blick festgestellt werden, ob diese Veränderungen durch den eigenen Code entstanden sind, oder ob dieser aufgetretene unerwünschte Zustand durch eine Änderung eines anderen Entwicklers bzw. einer anderen Entwicklerin, auf der dieser Code aufgebaut ist, entstanden ist. Die Ursache dieser Veränderung kann nur durch genauere Analyse des gemeinsam erstellten Codes ermittelt werden. Um diese Unsicherheiten zu vermeiden, müsste nach jeder Änderung des Codes durch einen Entwickler oder eine Entwicklerin die Seite bei allen EntwicklerInnen, die von diesem Code abhängen, manuell aktualisiert werden. So könnte stets der aktuelle Zustand angezeigt werden. Da nicht einfach ersichtlich ist, ob ein Code verändert wurde oder man von diesem Codeabschnitt abhängig ist, ist dies manuell nicht möglich.

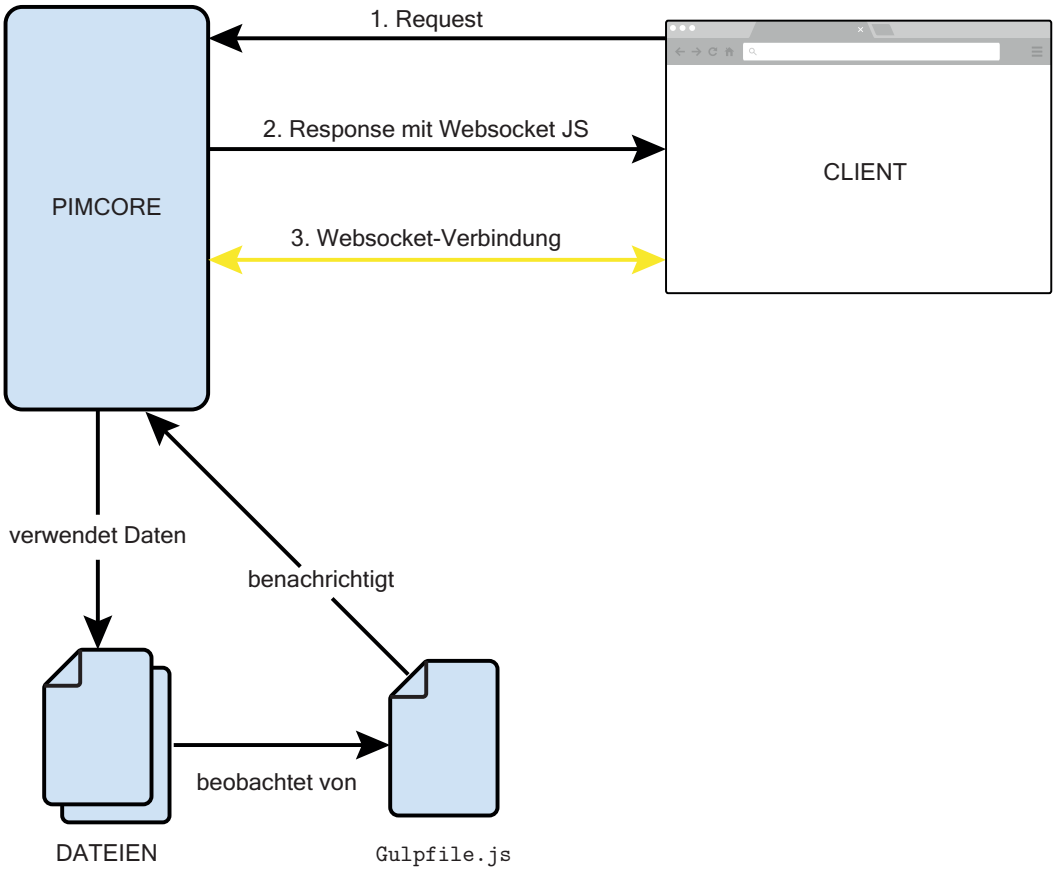


Abbildung 5.4: Websocket Verbindung mit Client über Pimcore [1].

5.3.1 Automatisches Aktualisieren

Das Aktualisieren des Views müsste also automatisch passieren, um die Änderungen aller EntwicklerInnen anzuzeigen, sobald diese passieren, falls der View von diesen Änderungen abhängt. Um ein Aktualisieren des geöffneten Views im Browser des Clients zu ermöglichen, muss der Client wissen, wann sich etwas am Server verändert. Um eine Änderung an den Client zu senden, muss eine konstante Verbindung aufgebaut werden. Eine Möglichkeit ist es, Polling zu verwenden, wo der Client kontinuierlich Anfragen an den Server stellt und überprüft, ob sich etwas geändert hat. Eine einfachere Methode ist es jedoch, diese Verbindung durch den Einsatz von Websockets zu realisieren (siehe Abschnitt 4.4). Dafür muss beim Laden der Webseite eine Websocket-Verbindung aufgebaut werden. Hier wird eine JavaScript-Datei bei der Server-Response mitgeschickt, welche die Websocket-Verbindung herstellt. Bei einer Änderung kann nun ein Signal über diese Websocket-Verbindung gesendet werden, welches dem JavaScript am Client einen Befehl zum Neuladen der Webseite sendet. Diese Änderungen von Dateien können zum Beispiel über *Watcher*-Tasks im Gulpfile verfolgt werden (siehe Abschnitt 4.3.2). Abbildung 5.4 zeigt einen solchen Ablauf.

CSS-Injektion

Um nicht bei jeder Änderung ein Neuladen der Seite zu verursachen, kann CSS auch injiziert werden. Das bedeutet: Wenn eine Änderung einer CSS-Datei beobachtet wurde, kann über den Websocket eine Benachrichtigung an den Client geschickt werden. Hier wird über JavaScript das DOM-Element für das Laden des CSS ausgetauscht. Beim Austauschen des Pfades für das Laden des CSS wird ein Zeitstempel an diesen angehängt, um zu verhindern, dass die Datei nicht aus dem Cache geladen wird, sondern neu über den Server heruntergeladen wird. Auch Bild-Dateien können über den Austausch des Pfades und das Hinzufügen eines Zeitstempels injiziert werden.

5.4 Veränderung des Designs

Der zweite Fokus beim Verbessern der Pipeline liegt auf der Gestaltung des StyleLab-Tools. Nach einer Analyse des bestehenden StyleLab-Interfaces wurde klar, dass es überladen wirkt. Es wirken viele verschiedene Farben und Elemente gleichzeitig auf die BenutzerInnen ein. Da dieses Tool beim Entwickeln von verschiedensten Komponenten zum Einsatz kommt, sollen diese auch visuell im Mittelpunkt stehen und sich klar von der Umgebung des StyleLab-Tools abheben. Auf der Hauptseite (siehe Abbildung 5.1) ist eine dominante schwarze Navigationsleiste zu finden, welche auf Detailseiten die Aufmerksamkeit auf sich zieht und von der eigentlichen Modulansicht ablenkt. Außerdem ist ein farbiges Logo vorzufinden. Der Kontrast der einzelnen Menüpunkte in der Navigationsleiste wirkt unübersichtlich und gerät dabei in den Hintergrund. Bei den Überschriften für die verschiedenen Module ist ein türkiser Hintergrund gesetzt, welcher auch hier die Aufmerksamkeit auf sich zieht. Die Detailansicht wirkt mit Funktionen überladen (siehe Abbildung 5.2). Diese können über die Buttons ein- und ausgeblendet werden. Dadurch wird jedoch die Anzahl der Buttons in der Navigationsleiste erhöht, die dadurch unübersichtlich wirkt, und sie lenken daher von der Ansicht des Moduls ab. Auch die Übersicht über mehrere Module wirkt unklar (siehe Abbildung 5.3). Es gibt keine klare Abtrennung der einzelnen Module. Es ist nicht klar ersichtlich, wo ein Modul anfängt und wo es aufhört.

5.5 Node-Server als Proxy

Im Abschnitt 5.3.1 wurde die Herangehensweise an eine Automatisierung des Neuladens der Webseite über eine Websocket-Verbindung theoretisch erläutert. Um diese Funktion in die bestehende Umgebung der Firma elements zu integrieren, müssen weitere Schritte implementiert werden. Ein Problem bei dieser Vorgangsweise ist die Herstellung der Verbindung zwischen dem Gulpfile und dem Pimcore-System, um Benachrichtigungen über Detailänderungen zu übermitteln (siehe Abbildung 5.4). Um eine Benachrichtigung zwischen dem auf Node basierten Gulp und dem auf PHP laufenden Pimcore-System zu ermöglichen, muss zum Beispiel eine Transmission Control Protocol(TCP)-Verbindung zwischen diesen zwei Instanzen hergestellt werden[1]. Außerdem muss eine Implementierung für Websockets, wie zum Beispiel Ratchet, verwendet werden, um Websockets für

[1] Mehr zu TCP unter: https://tools.ietf.org/html/rfc793

PHP zu ermöglichen[2].

Eine viel einfachere Vorgangsweise ist das Verwenden des schon bestehenden Node-Systems. Wie Abbildung 5.5 zeigt, kann ein Node-Server als Proxy verwendet werden. Durch diese Lösung muss kein neues kompliziertes Plugin geschrieben werden, sondern es kann das bestehende Node-System verwendet werden, um den Aufbau der Websocket-Verbindung zu verwalten. Der Node-Proxy-Server kann hierbei die Websocket-Verbindung aufbauen und gleichzeitig die Dateien hinsichtlich Veränderungen beobachten.

5.5.1 Modulabhängige Aktualisierung

Da der Node-Proxy-Server nur das fertig gerenderte HTML des Pimcore-Servers als Antwort auf eine Anfrage bekommt, kann dieser nicht wissen, welche PHP-Module in diesem View inkludiert sind. Um nicht bei jeder Modul-Änderung an alle aktiven Websocket-Verbindungen einen Befehl zum Neuladen der Seite zu schicken, muss am Node-Proxy-Server gespeichert werden, welche Socket-Verbindungen von welchen Dateien abhängen. So kann sichergestellt werden, dass der Browser nur neu geladen werden muss, wenn sich ein Modul ändert, das für das Anzeigen des Views notwendig ist. Um dies zu erreichen, muss der Node-Proxy-Server wissen, welche Module im View verwendet werden. Da der Node-Proxy-Server nur für das Weiterreichen von Anfragen zuständig ist und keine PHP-Dateien rendern kann, muss der Pimcore-Server dem Node-Proxy-Server eine Liste von verwendeten Modulen mit übergeben. Damit der Pimcore-Server weiß, dass er diese Liste mitsenden muss, wird beim Durchreichen des Requests ein Proxy-Parameter angehängt (siehe Abbildung 5.5). Danach wird ein JSON mit dem Inhalt einer Liste für die verwendeten Module und die fertig gerenderte HTML-Antwort geschickt. Die HTML-Antwort wird an den Client zurückgegeben. Über eine beim Response mitgeschickte ID, welche bei der Verbindungsherstellung des Websockets mit übertragen wird, merkt sich der Node-Proxy-Server, welche Websocket-Verbindung von welchen Modulen abhängig ist. Ändert sich nun ein Modul, kann über alle Websockets, die dieses Modul verwenden, eine Anfrage zum Neuladen der Seite gesendet werden.

[2]Siehe http://socketo.me/

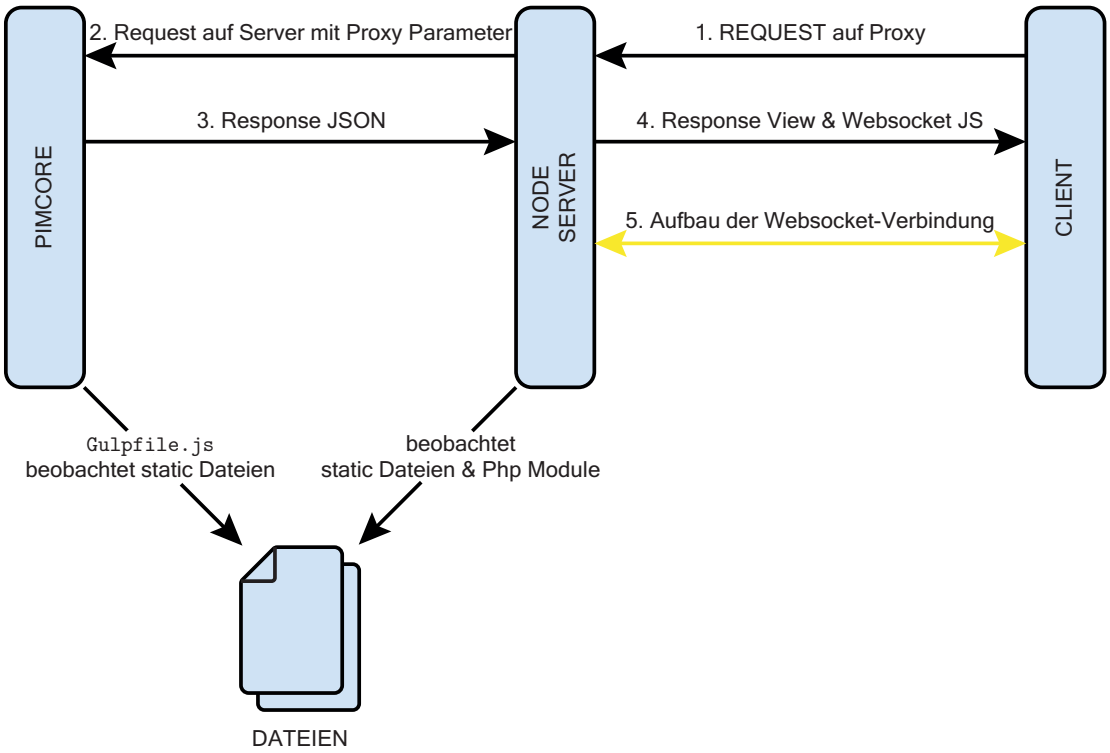


Abbildung 5.5: Websocket Verbindung mit Client über Pimcore und Node-Proxy-Server.

Kapitel 6

Implementierung

Wie in Kapitel 5 beschrieben wurde, ist die Aufgabe das Weiterentwickeln des bestehenden StyleLab-Tools der Firma elements, welches als Plugin im CMS Pimcore als Style Guide-Tool verwendet wird. Die Lösung, die im vorherigen Kapitel beschrieben wurde, ist die Umsetzung einer automatischen Aktualisierung der Webseite bei Ändern einer Komponente durch einen Entwickler bzw. eine Entwicklerin. Dies soll zum effektiveren Zusammenarbeiten der EntwicklerInnen beitragen. Dieses Real-Time-Feature wird durch den Einsatz von Websockets ermöglicht, da das Prinzip des Pollings, also das zyklische Abfragen des momentanen Zustands, zu ineffizient ist und nicht dem Real-Time-Prinzip entspricht.

Diese Aufgabe der Umsetzung lässt sich in vier Teile aufteilen. Wie in Abschnitt 5.5 beschrieben wurde, wird ein Proxy-Server als vermittelnde Instanz zwischen dem Pimcore-Server, welcher auf dem Zend Framework aufbaut, und dem Webclient verwendet. Dieses Prinzip führt zu der Unterteilung der ersten drei Abschnitte in Modifizierung des StyleLab-Plugins am Pimcore Server, Erstellen des Node.js Proxy Servers und Hinzufügen von Client-Funktionen zur Herstellung der Websocket-Verbindung. Aufgrund der Analyse des Designs, siehe Abschnitt 5.4, wird im Abschnitt 6.3 auf die veränderten Funktionalitäten eingegangen. Die Design-Änderungen werden in Abschnitt 6.4, beschrieben.

6.1 Modifizierung des StyleLab-Plugins

Um Funktionen an das StyleLab-Plugin hinzuzufügen, muss eine Erweiterung im StyleLab-Plugin geschrieben werden. Diese Erweiterung wird als Zend-Plugin definiert, welches sich dann in den Zend Dispatch Prozess einhängt.

6.1.1 Zend Dispatch Prozess

Der Zend-Dispatch-Workflow ist eine MVC-Implementierung[1] des Zend-Frameworks. Dieses Zend-Modul wird von Pimcore verwendet, um den Request abzuarbeiten und den Response zu generieren. Abbildung 6.1 zeigt die Implementierung in die Pimcore-Umgebung und den Ablauf des Controller-Workflows. Die grün markierten Bereiche sind

[1] MVC – Model View Controller

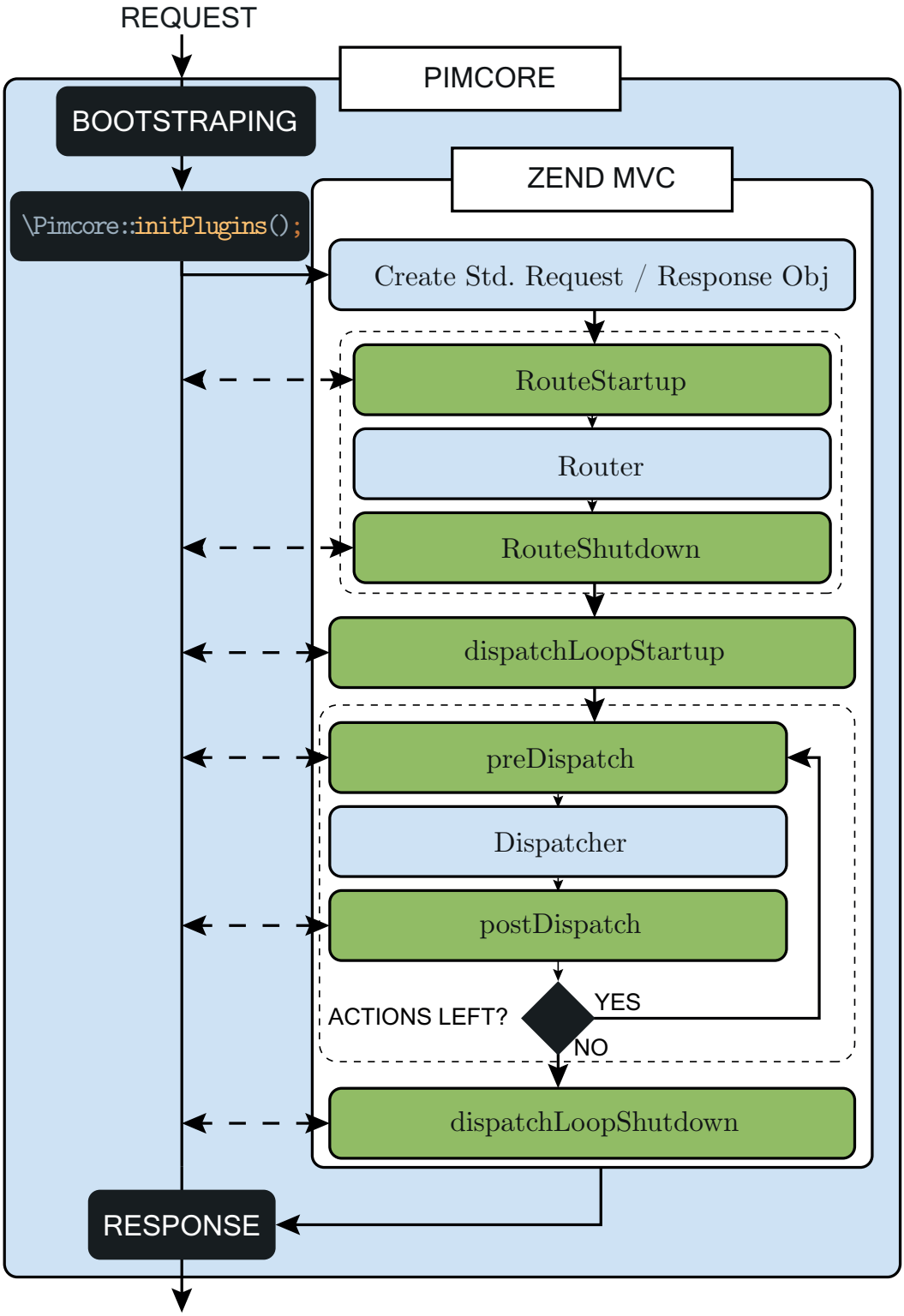


Abbildung 6.1: Die Abbildung beschreibt den MVC-Arbeitsablauf von ZEND innerhalb des Pimcore-Systems. Grün markierte Bereiche beschreiben Ereignismethoden.

Ereignismethoden, welche vom Pimcore System überschrieben werden können. Eine Ereignismethode, manchmal auch Einschubmethode genannt[2], ist eine Schnittstelle, mit Hilfe derer ein Programmierer bzw. eine Programmiererin einen fremden Programmcode in eine bestehende Anwendung integrieren kann, um diese zu erweitern. Ereignismethoden existieren bereits als leere Methoden im `Zend_Controller_Plugin_Abstract`, von dem das StyleLab-Plugin erbt.

Ein Request an das Pimcore System startet zuerst das Bootstrapping, welches verantwortlich für das Laden der notwendigen Systemkomponenten und das Initialisieren der Umgebung ist. Ein Teil dieses Bootstrapping-Prozesses ist das Laden der vorhandenen Pimcore-Plugins, welche mit Hilfe des Abstract-Zend-Plugin-Controllers erstellt werden. Als Nächstes wird der Zend-Dispatch-Workflow geladen, welcher für das Erstellen des Request- und Response-Objektes zuständig ist. Danach durchläuft das System das Routing und den Dispatcher, welche für das richtige Laden der Controller zuständig sind. Hier sind einige Ereignismethoden vorgesehen, um das System zu erweitern. Nachdem alle für den Request zuständigen Controller und Methoden ausgeführt wurden, wird die dispatchLoopShutdown-Ereignismethode aufgerufen. Hier ist das Response-Objekt vollständig generiert und kann nun vor dem Senden verändert werden.

6.1.2 Hinzufügen der Ereignismethode

Wie in Abschnitt 5.5 beschrieben, muss für das Zwischenspeichern der im von Pimcore generierten Response-Bodys verwendeten Views am Node-Proxy-Server der Response von Pimcore verändert werden. Um dies zu erreichen, muss eine Ereignismethode, welche nach dem fertigen Erstellen der Response aufgerufen wird, überschrieben werden. Die `dispatchLoopShutdown`-Ereignismethode eignet sich dafür perfekt. Programm 6.1 zeigt das Überschreiben der Methode. In dieser Methode wird der Response in ein JSON-Objekt umgewandelt. Dieses Objekt beinhaltet nun alle geöffneten Dateien, den ursprünglichen Response-Body und einen Kontrollwert über den sichergestellt wird, dass dieses JSON-Objekt für das Speichern der Views im Node-Proxy bestimmt ist. Um sicherzustellen, dass das StyleLab-Plugin auch ohne dem Node-Proxy-Server funktioniert, wird der Inhalt der Ereignismethode nur ausgeführt, wenn bei dem ursprünglichen Request der Übergabeparameter `browserSynced=1` gesetzt ist. Um zu erkennen, welche Views nun verwendet wurden, wird die Methode `get_included_files()` verwendet, welche ein Array mit den Namen der inkludierten Dateien zurückliefert.

[2] Im Englischen „hook“ genannt.

Programm 6.1: Dieses Programm zeigt das Überschreiben der `dispatchLoopShutdown`-Ereignismethode. Hier wird das fertige Response-Objekt für Requests über den StyleLab-Proxy überschrieben.

```
1 class BrowserSyncedPlugin extends \Zend_Controller_Plugin_Abstract
2 {
3     private static $isBrowserSynced = false;
4
5     public static function isBrowserSynced($isBrowserSynced = null)
6     {
7         if ($isBrowserSynced === null) {
8             return self::$isBrowserSynced;
9         }
10        self::$isBrowserSynced = $isBrowserSynced;
11    }
12     public function dispatchLoopShutdown()
13      {
14        if (!Tool::isHtmlResponse($this->getResponse())) {
15          return;
16        }
17
18        //Erkennen ob Body über den Browsersync−Proxy geladen wurde
19        if ($this->enabled && self::$isBrowserSynced) {
20
21          //Holen des Responsebodys
22          $body = $this->getResponse()->getBody();
23
24          //Finden aller html views die für diesen Request verwendet wurden
25          $startsWith = PIMCORE_DOCUMENT_ROOT . "/html/";
26          $length = strlen($startsWith);
27          $openFiles = [];
28
29          foreach (get_included_files() as $file) {
30            if (substr($file, 0, $length) === $startsWith) {
31                      $openFiles[] = str_replace(PIMCORE_DOCUMENT_ROOT . "/", "",
      $file);
32            }
33          }
34
35          //Response in Json umwandeln und Benötigte Dateien sowie Body zurückschicken
36          $this->getResponse()->setBody(json_encode([
37            "distinctReload" => true,
38            "body" => $body,
39            "openFiles" => $openFiles
40          ]));
41         }
42      }
43 }
```

6.1.3 Registrierung der StyleLab-Erweiterung

Um das Plugin nun zu verwenden, muss es beim Zend-Plugin-Brooker registriert werden. Diese Registrierung wird in der Datei `Plugin.php` des ursprünglichen StyleLab-Plugins vorgenommen:

```
\Zend_Controller_Front::getInstance()->registerPlugin(new \StyleLab\
BrowserSyncedPlugin());
```

Nun muss nur noch der BrowserSynced-Übergabe-Parameter im Controller des ursprünglichen StyleLab-Plugins ausgelesen werden und intern gespeichert werden:

```
1 if($this->getParam("browserSynced") && !$this->getRequest()->isXmlHttpRequest()) {
2     \StyleLab\BrowserSyncedPlugin::isBrowserSynced(true);
3 }
```

6.2 Erstellen des Node.js Proxy-Servers

Als nächster Schritt wird der Proxy-Server erstellt, welcher über Node.js läuft. Der Proxy Server hat, wie in Abschnitt 5.5 beschrieben, mehrere Aufgaben. Einerseits besteht die Aufgabe darin, Requests des Clients an den Pimcore Server weiterzuleiten und mit einem Parameter zu versehen, sodass dieser die StyleLab-Erweiterung ausführt und eine veränderte Response mit den Viewparametern zurückschickt. Andererseits muss der Proxy-Server mit jedem Client eine Websocket Verbindung aufbauen, diese managen und das Datei-System auf Änderungen überprüfen. Sobald eine Änderung passiert, muss entschieden werden, welchen Websocket Verbindungen ein Update geschickt wird. Um das Updaten und Herstellen der Websocket-Verbindungen zu erleichtern, wurde das NPM-Packet `BrowserSync` verwendet.

6.2.1 BrowserSync

In einem anfänglichen Prototyp wurde versucht, wie in Abschnitt 5.3.1 theoretisch erklärt, über eine Websocket Verbindung und eine dazugehörige JavaScript-Datei am Client das automatische Neuladen des Browsers bei einer Änderung einer Datei zu ermöglichen. Um nun nicht Sicherungssysteme für alle Eventualitäten selbst schreiben zu müssen und diesen Prozess zu automatisieren, wurde das NPM-Paket BrowserSync verwendet. BrowserSync ist ein auf Node.js basierter Server, der beim Entwickeln von Frontends helfen soll, indem er Dateiveränderungen und Interaktionen über mehrere Geräte synchronisiert. Da es ein simpler Node.js-Server ist, kann er lediglich HTML-Dateien sowie statische Dateien versenden. Um mit Pimcore zu funktionieren, kann dieser Node.js-Server als Proxy geschaltet werden, wodurch der Request weitergeleitet wird. Beim Senden des ursprünglichen Responses greift dann BrowserSync ein und hängt eine JavaScript-Datei an, welche mit dem Node-Proxy-Server die Websocket-Verbindung aufbaut. Außerdem werden Pfade beim Erstellen des BrowserSync-Servers in der Konfiguration mit übergeben, welche BrowserSync auf Änderungen überwacht. Ändert sich eine Datei, wird der Client davon benachrichtigt.

BrowserSync-Plugins

Um die Funktionalität von BrowserSync zu verändern, ohne das NPM-Plugin umschreiben zu müssen, haben die EntwicklerInnen von BrowserSync eine Plugin-Funktionalität inkludiert. Beim Erstellen einer BrowserSync-Instanz können über die Parameter andere NPM-Plugins, welche mit BrowserSync funktionieren, instanziert werden und die Funktionalität von BrowserSync erweitert werden. Diese Plugins können auf das BrowserSync-Objekt zugreifen und somit BrowserSync-Funktionen überschreiben oder nutzen, um zum Beispiel Middlewares hinzuzufügen.

6.2.2 Authentifizierung

Zunächst wird eine Authentifizierungsfunktion implementiert, welche sicherstellt, dass zwischen den verschiedenen UserInnen unterschieden werden kann. Dafür wird eine Konfiguration, `CustomRoutes`, in der BrowserSync-Konfiguration angelegt, welche eine neue Route zu einer Loginseite festlegt. Im BrowserSync-Plugin wird dann eine Middelware angelegt, welche die Authentifikation durchführt. Zur Authentifikation wird das NPM-Plugin `workfront-api` verwendet. Mit Hilfe dieses Plugins können die für die Firma elements verwendeten Login-Zugänge verwendet werden.

6.2.3 Sync-Parameter

Um, wie in Abschnitt 5.5.1 beschrieben, den am Proxy-Server ankommenden Request um einen Parameter zu erweitern, wird im Proxy-Server eine Middleware-Funktion erstellt. Mit Hilfe des Parameters kann der Pimcore-Server den Proxy-Request von einem normalen Request unterscheiden.

```
1 module.exports.addSyncParameter = function (req, res, next) {
2     try {
3         let parsed = url.parse(req.url, true);
4 
5         let queryStarter = "?";
6 
7         //falls bereits ein Parameter in der url existiert
8         if (parsed.search.length) {
9             queryStarter = "&";
10        }
11 
12        //Sync-Parameter an url anhängen
13        req.url += queryStarter + "browserSynced=1";
14    } catch(exception) {
15        console.log(exception);
16    }
17    next();
18 };
```

Diese Middleware-Funktion, welche im `Middleware.js` angelegt wird, muss nun über das im BrowserSync-Plugin verfügbare BrowserSync-Objekt als Middleware eingefügt werden.

```
1 // Sync-Parameter bei jedem Request hinzufügen
2 bs.addMiddleware("", Middleware.addSyncParameter);
```

6.2.4 Mime-Type

Damit zwischen den verschiedenen Requests und deren Dateitypen unterschieden werden kann, wird der Mime-Type überprüft. Alle Dateien, welche nicht dem Mime-Type `'html'` oder `'application/octet-stream'` entsprechen, sollen gar nicht erst gebuffert werden sondern sofort an den Client weitergeleitet werden, um die Performance der Applikation zu steigern:

```
module.exports.bufferResponse()
    let parsed = url.parse(req.url, true);
    let mime = Mime.lookup(parsed.pathname);

    if (bufferedMimeTypes.includes(mime) && !req.xhr) {
        new BufferedResponse(req, res);
    }

    next();
};
```

6.2.5 Response abändern

Wenn nun der Mime-Type dem Typ `'html/text'` entspricht, wird ein Objekt der Klasse `BufferedResponse` erstellt, welches das Umwandeln und Buffern der Response regelt, indem die vom ursprünglichen Response-Objekt zur Verfügung stehenden Methoden wie `writeHead`, `write` und `end` überschrieben werden. Wenn nun das ganze Objekt fertig gebuffert, also übertragen, wurde, wird das vom Pimcore-Server zurückgegebene JSON geparst und der Json-Response-Parameter `distinctReload` überprüft. Falls dieser gesetzt ist und dieses JSON auch den HTML-Response-Body sowie die geöffneten Dateien enthält, wird als neuer Response der im JSON enthaltene HTML-BODY festgelegt, welcher der von Pimcore erstellten HTML entspricht. Danach werden die geöffneten Dateien sowie die zuvor aus der URL extrahierte `UID`, welche von Pimcore als URL-Parameter gesetzt wurde, im `SocketManager` in einer Hash-Map hinterlegt, um später die `UID` mit den geöffneten Dateien verknüpfen zu können. Im folgenden Codeabschnitt ist die Übergabe der Daten an den `SocketManager` zu sehen:

```
if (json.distinctReload && json.body && json.openFiles) {
    SocketManager.addPendingFiles(this.uid, json.openFiles);
    response = json.body;
}
```

6.2.6 SocketManager

Der `SocketManager` wurde geschaffen, um die verschiedenen Socketverbindungen sowie die geöffneten Dateien zu verwalten und miteinander zu verknüpfen. Beim Erstellen des `SocketManger`-Singleton wird ein `ClientStore` erstellt (mehr dazu in Abschnitt 6.2.7). Außerdem wird ein Objekt erstellt, welches die in der JSON-Response enthaltenen benötigten Dateien mit der `UID` als Hash-Map speichert. Um einen weiteren Eintrag in der Hash-Map zu erstellen, wurde die Funktion `addPendingFiles` erstellt.

```
addPendingFiles(id, openFiles) {
    this.pending[id] = openFiles;
}
```

Beim Laden des gesamten Plugins über BrowserSync wird der `Socketmanager` initialisiert. Dafür wird das BrowserSync-Objekt sowie das *Socket.io*-Objekt, welches ein Teil von BrowserSync ist, an den `SocketManager` übergeben. Dies ermöglicht nun, weitere Socket-Events hinzuzufügen, damit der `SocketManager` auf die vom Client geschickte `UID` reagieren kann. Sobald der Client die Verbindung mit dem Server über Websocket herstellt, wird das Event `socket:uid` geschickt, welches die `UID` beinhaltet. So kann die Websocket-Verbindung mit den geöffneten Dateien verknüpft und im `ClientStore` abgespeichert werden.

```
1 socket.on("socket:uid", (data) => {
2     this.clientStore.add(socket, this.pending[data.uid]);
3     delete this.pending[data.uid];
4 });
```

6.2.7 ClientStore

Der `ClientStore`-Singleton besteht aus einer Sammlung von `Client-Objekten` (siehe Abschnitt 6.3). Wenn der `SocketManager` die geöffneten Dateien mit dem Websocket verknüpft, wird dies dem `ClientStore` übergeben. Dieser überprüft anschließend die gesetzte Session-ID von der Auth-Middleware und erstellt ein neues `Client-Objekt`, falls noch keines mit dieser Session-ID vorhanden ist. Dies wird im Clients-Array des `ClientStore` abgespeichert. Anschließend wird dem `Client-Objekt` ein weiterer Workspace hinzugefügt und die Socket-ID sowie der Socket selbst und die dazu gehörigen geöffneten Dateien übergeben.

6.2.8 Client-Objekt

Ein `Client`-Objekt speichert nun die Socket-ID sowie den Socket selbst und die dazu gehörigen geöffneten Dateien in einem Workspace-Objekt ab. Das bedeutet, ein Client kann mehrere Workspaces geöffnet haben. So können verschiedene geöffnete Browserinstanzen sowie Geräte einem Client zugeordnet werden.

6.2.9 Überschreiben des BrowserSync-Reload-Mechanismus

Um nicht alle geöffneten Clients bei einer gefundenen Änderung der verwendeten Dateien neu laden zu müssen, muss die BrowserSync implementierte Funktion `doFileReload` überschrieben werden. Beim genaueren Betrachten der ursprünglichen Funktion wurde festgestellt, dass innerhalb dieser Funktion über eine Konfiguration, welche über eine Funktion der BrowserSync Bibliothek geladen wird, festgestellt wird, ob für die übergebene Datei ein Neuladen des Browsers mittels dem Befehl `"browser:reload"` oder nur ein Neuladen der Datei, also Injektion, über den Befehl `"file:reload"` über die Websocket-Verbindung erfolgen soll. So soll also die überschriebene Funktion auch diese zwei Socket-Befehle schicken. Der Unterschied jedoch ist, dass bei Neuladen des Browsers entschieden werden soll, welche Clients diesen Befehl gesendet bekommen. Dazu werden bei dieser Funktion alle Clients abgefragt, ob sie diese Datei in einem ihrer Workspaces als Abhängigkeit besitzen. Um dies zu ermöglichen, wird die Funktion `notifyIfAffected` verwendet. Sie durchsucht die Workspaces nach der Datei und sendet den Websocket-Befehl `"browser:reload"`, falls die Datei enthalten ist:

```
notifyIfAffected(file) {
    let fileParts = file.split(path.sep);

    //going over all workspaces
    _.each(this.workspaces, (workspace) => {
        //and all files in these workspaces
        _.each(workspace.files, (workspaceFile) => {
            let workspaceFileParts = workspaceFile.split(path.sep);

            let fi = fileParts.length - 1;
            let wfi = workspaceFileParts.length - 1;
            let matched = true;

            //see if the file is included
            while (fi >= 0 && wfi >= 0 && matched) {
                if (fileParts[fi--] != workspaceFileParts[wfi--]) {
                    matched = false;
                }
            }

            //if its included notifying client for file
            if (matched) {
                workspace.socket.emit("browser:reload");
            }
        });
    });
}});
```

6.3 Hinzufügen von Client Funktionen

Auch die am Client verwendeten Funktionen wurden verändert und erweitert. Wie in Abschnitt 6.4 beschrieben, sind einige der Änderungen der Funktionalität notwendig, um die visuelle Repräsentation und die Verwendbarkeit des Tools zu steigern.

6.3.1 Socket Identifizierung

Wie in Abschnitt 6.2.6 beschrieben wurde, muss der Client die in Pimcore generierte `UID` über den Socket zurück an den Node.js-Proxy-Server schicken, um den Socket mit den dazugehörigen Dateien zu identifizieren. Dazu wird die `UID`, welche von Pimcore zur URL hinzugefügt wird, ausgelesen. Diese wird dann mit dem Websocket-Befehl `ßocket:uid"` zurück an den Node.js-Proxy-Server gesendet, welcher mit dieser `UID` die Dateiliste mit dem Socket identifizieren kann:

```
;(function (bs) {

    var uid = window.location.href.match(/uid\=(\w+)/);

    if(uid) {
        bs.socket.emit("socket:uid", {uid: uid[1]});
    }

})(___browserSync___);
```

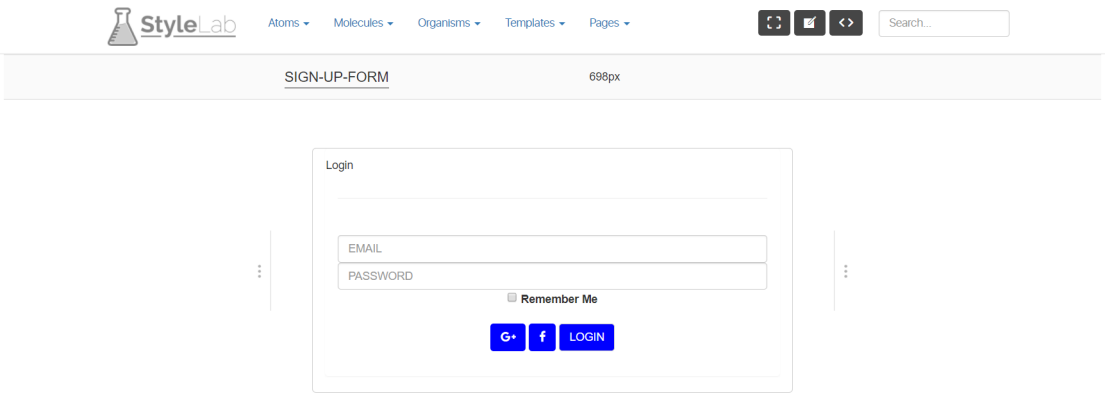


Abbildung 6.2: Verbesserte StyleLab Detailansicht.

6.3.2 Skalierungsfunktion

Um die Anzahl der Buttons, wie in Abschnitt 5.4 beschrieben, zu vermindern und die Funktionalität zu erweitern, wurden die Skalierungsbuttons durch eine interaktive Skalierung ersetzt (siehe Abbildung 6.2). Diese Skalierungsgriffe erscheinen beim Hovern oder Interagieren mit der Komponente auf den beiden horizontalen Rändern der Komponente. Diese Skalierungsgriffe können durch Klicken und Bewegen verschoben werden. Durch das Verschieben passt sich die Breite der Komponente an. Die momentane Breite wird immer durch einen Pixelwert über der Komponente gekennzeichnet. Als minimale Breite wurden 320px genommen. Dies entspricht der Standard-Minimalbreite eines Mobiltelefons.

6.4 Design Veränderungen

Um die User-Interaktionen zu erleichtern und den Fokus mehr auf die Module zu legen, sind auch Veränderungen am Design vorgenommen worden. Um das Design zu verändern, ist die Datei `custom.css` angelegt worden. In dieser Datei wurden bestehende DOM-Elemente verändert. Außerdem wurden einige PHP-Dateien, welche für das Generieren des Stylelab-Grundgerüsts zuständig sind, angepasst. Auch die JavaScript-Datei `frontend.js` wurde verändert, um Einfluss auf die visuelle Repräsentation zu nehmen.

6.4.1 Vereinfachung bestehender Funktionalitäten

Ziel der Funktionalitätsveränderung war es insbesondere, die Anzahl der Bedienelemente zu minimieren und deren Funktion auf interaktive Elemente zu legen, welche ausgeblendet werden können. Wie in Abschnitt 5.4 beschrieben, ist einer der Mängel im Design das überladene Interface. Um dieser visuellen Belastung des Interfaces entgegenzuwirken, wurden die Buttons in der Navigationsleiste analysiert. Die drei Buttons, die die Breite des Moduls in verschiedenen Weiten anzeigen, wurden durch eine interaktive Skalierung ersetzt, wie in Abschnitt 6.3.2 beschrieben wurde.

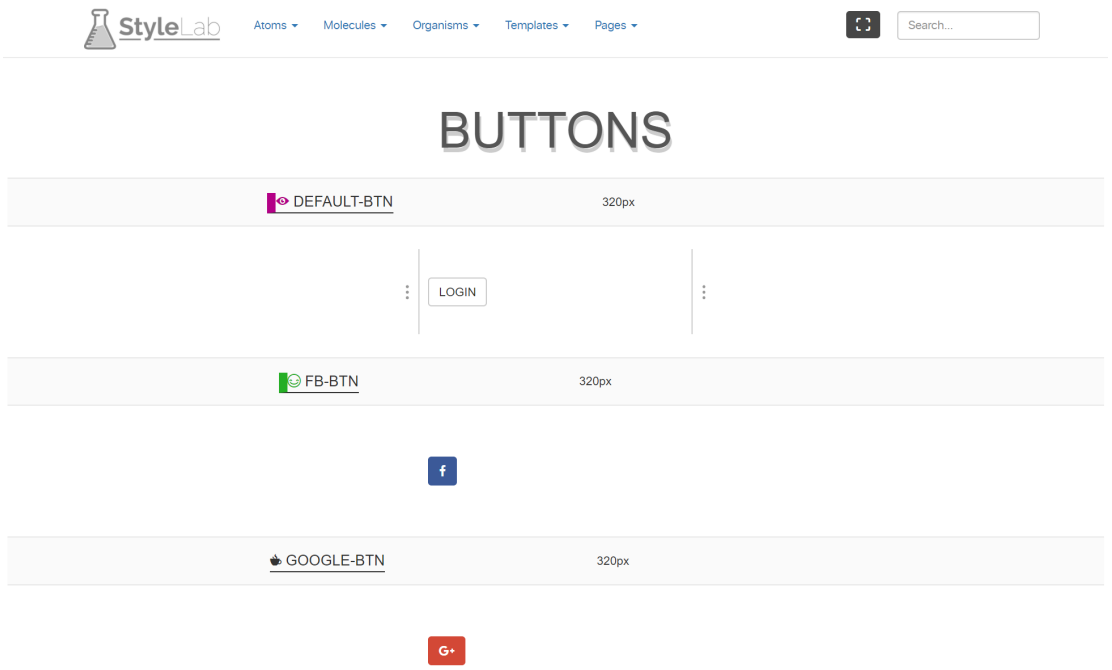


Abbildung 6.3: Verbesserte StyleLab Multikomponentenansicht.

6.4.2 Neutraleres Design

Wie in den Abbildungen 6.2–6.4 zu sehen ist, wurden im Vergleich zu den Abbildungen 5.1–5.3 einige visuelle Änderungen vorgenommen. Vor allem wurde versucht, Farbelemente aus dem ursprünglichen Design zu minimieren. Das Logo wurde vereinfacht und in verschiedene Graustufen abgeändert. Dies soll den Fokus auf das Logo verringern. Dunkle Elemente, wie die Navigationsleiste oder auszuklappende Paneele, wurden aufgehellt, um den Fokus davon abzuziehen.

6.4.3 Verbesserungen der einzelnen Seiten

Abbildung 6.2 zeigt die veränderte Detailseite. Im Vergleich zur ursprünglichen Detailseite (siehe Abbildung 5.2) wird nun der äußere Rand des Modules durch die Skalierungsgriffe abgetrennt. Zusätzlich wurde eine Überschriftsleiste hinzugefügt, welche den Namen sowie die momentane Pixel-Breite anzeigt. Durch Klicken auf den als Link gekennzeichneten Namen kann nun das Modul ohne das StyleLab-Grundgerüst in einem neuen Fenster geöffnet werden. Dies macht den -Button überflüssig und verringert die Buttonanzahl weiter.

Abbildung 6.3 zeigt die verbesserte Übersichtsseite über mehrere Module. Die ursprüngliche Übersichtsseite (siehe Abbildung 5.3) hatte, wie in Abschnitt 5.4 besprochen wurde, Probleme, die einzelnen Komponenten abzutrennen. Durch das zusätzliche Einfügen der Überschriftsleiste, welche schon in der Detailseite verwendet wurde, konnte eine klarere Abtrennung der verschiedenen Module geschaffen werden. Durch das Auf-

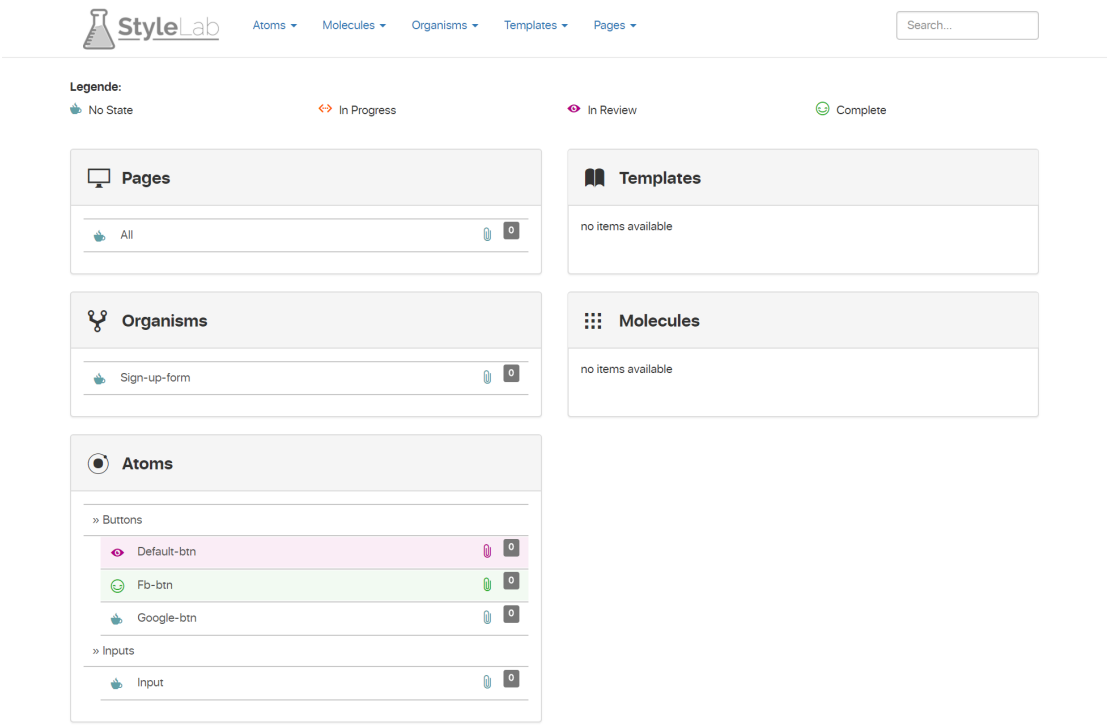


Abbildung 6.4: Verbesserter StyleLab-Home-Screen.

teilen der Module in eigene Bereiche und das Hinzufügen der Skalierungsgriffe kann nun auch für jede Komponente die Pixelbreite individuell konfiguriert werden. Auch hier kann mit Klick auf einen Modulnamen das Modul in einem neuen Fenster ohne das StyleLab-Grundgerüst geöffnet werden. Zusätzlich ist für die leichtere Übersicht eine Überschrift der Kategorie unter dem Header eingefügt worden.

Die Abbildung 6.4 zeigt die aktualisierte Version des StyleLab-Home-Screens. Im Vergleich zu Abbildung 5.1, welche die alte Version zeigt, ist auch hier die klare Verringerung von Farben zu erkennen. Alle Hintergrundfarben der Überschriften wurden durch einen leichten Grauton ersetzt. Auch der Hintergrund der Webseite ist nun weiß.

Kapitel 7

Evaluierung

In diesem Kapitel wird das zuvor umgesetzte Tool durch die Befragung von Experten evaluiert. Für die Erhebung der Daten wurde eine genaue Testumgebung definiert, in der sich immer zwei Frontend-EntwicklerInnen gleichzeitig einer zuvor definierten Aufgabe stellten. Nach dem Testen wurde den TeilnehmerInnen ein Online-Fragebogen zur Evaluierung bereitgestellt.

7.1 Datenerhebung

Um zu testen, ob die erstellte Software die Firma elements und auch andere EntwicklerInnen in ihrem Entwicklungsprozess in einem ähnlichen Umfeld unterstützt und die Produktivität der Arbeit steigert, wurde ein `objektives` Verfahren ausgesucht, welches harte Daten hervorbringt. Diese harte Daten sind einfach zu vergleichen. Bei einer `subjektiven` Testmethode steht die Bewertung der Benutzungsschnittstelle durch den Benutzer bzw. die Benutzerin im Vordergrund. Hier werden sogenannte weiche Daten ermittelt, wie beispielsweise, ob die Benutzung des Tools für den User bzw. die Userin freundlich und hilfreich gestaltet war [5, S. 18]. Um auch weiche Daten für die Auswertung zu erzeugen, kann die ausgewählte Methode angepasst werden.

7.1.1 Auswahl der Methode

Um diese Daten zu erheben, bietet sich die quantitative Befragung an. Einer der Vorteile der quantitative Befragung mit Hilfe eines Online-Fragebogens sind die niedrigen Erhebungs- und Auswertungskosten des Verfahrens [6, S. 116]. Ein weiterer Vorteil ergibt sich aus der Anonymität der Befragten, wodurch mit ehrlicheren Antworten der Befragten zu rechnen ist [8, S. 100]. Ebenso eignet sich die quantitative Methode dazu, Hypothesen präzise als Fragen zu formulieren und die Ergebnisse exakt miteinander vergleichen zu können [9, S. 90]. Dieser Vorteil zieht auch einen Nachteil mit sich, denn es ist ein stark vorgegebener Bewertungsprozess, der eine Richtung vorgibt. Dies kann durch Einfügen von offenen Fragestellungen im Fragebogen aufgeweicht werden, um auch weiche Daten zu erzeugen [6, S. 116].

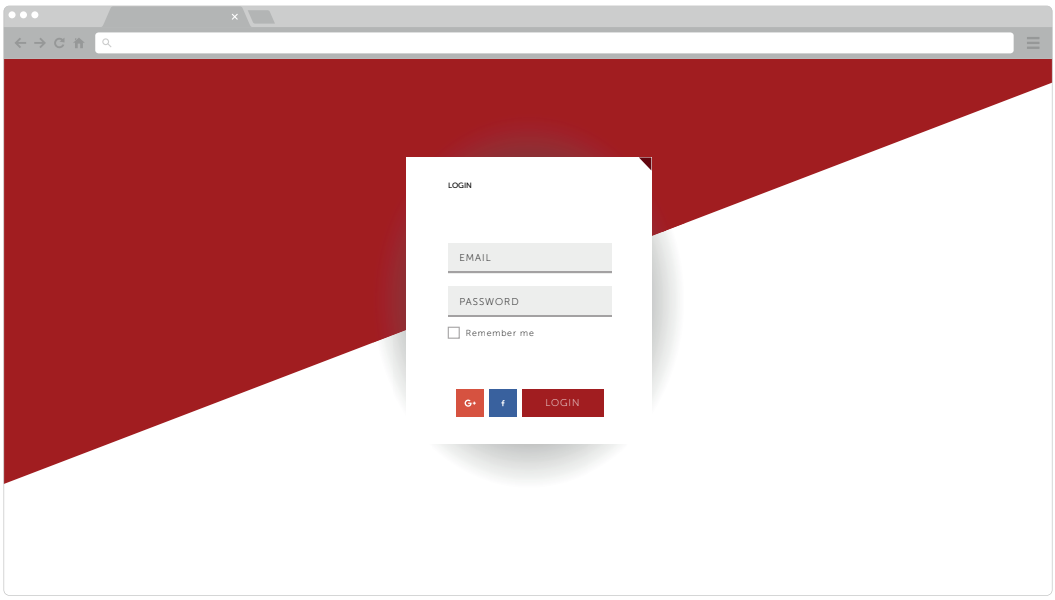


Abbildung 7.1: Screendesign der Login Form für die Testaufabe.

7.1.2 Zielgruppe der Untersuchung

Die Zielgruppe der Untersuchung sind vorwiegend erfahrene Frontend-EntwicklerInnen. Minimalvoraussetzungen sind CSS- und HTML-Kenntnisse. Kenntnisse in JavaScript sind für den Test nicht von Nöten. Da dieses Plugin überwiegend für die Firma elements erstellt wurde, um deren Frontend-Pipeline zu optimieren, sind natürlich deren EntwicklerInnen ein Teil der Zielgruppe. Um jedoch Objektivität zu bewahren, wurden auch externe EntwicklerInnen herangezogen, welche zuvor nicht mit dem Tool gearbeitet haben.

7.2 Methodisches Vorgehen

In diesem Abschnitt wird die Vorbereitung für die Testaufgabe beschrieben, ebenso wie der Aufbau und der Ablauf derselben. Außerdem wird auf das Verfahren der Auswertung eingegangen.

7.2.1 Erstellung der Testaufgabe

Bevor der Test beginnen konnte, musste eine Testaufgabe erstellt werden und es mussten Rahmenbedingungen definiert werden. Da es sich bei der Zielgruppe der Untersuchung vorwiegend um erfahrene Frontend-EntwicklerInnen handelt, musste eine Aufgabe geschaffen werden, welche nicht allzu einfach, jedoch auch für AnfängerInnen bewältigbar ist. Dies ist eine große Herausforderung. Außerdem soll die Testaufgabe den Zweck des Tools untersuchen und prüfen, ob die Echtzeit-Aktualisierungen auch tatsächlich einen Vorteil beim Entwickeln bringen. Ebenfalls soll durch die Aufgabe untersucht werden, ob dies einen Vorteil beim gemeinsamen Arbeiten bringt und somit das kollaborative

Arbeiten bei einem Frontend-Task optimiert. Um den kollaborativen Aspekt zu testen, müssen immer zwei EntwicklerInnen gleichzeitig den Test absolvieren. In der Testaufgabe wurde daher auf `PersonA` und `PersonB` referenziert, um die zwei KandidatInnen zu unterscheiden. Außerdem müssen die Aufgaben der zwei EntwicklerInnen miteinander zu tun haben, beziehungsweise aufeinander aufbauen, damit diese EntwicklerInnen gemeinsam arbeiten müssen. Durch die Beschaffenheit des Tools ist dies recht einfach. Da es auf dem `Atomic Design` aufbaut, sind die einzelnen Komponenten von einander getrennt, jedoch können sie auf einander aufbauen. So muss sich `PersonA` um das Erstellen von `Atomen` aus dem `Atomic Design` kümmern und `PersonB` baut darauf auf. Um die Aufgabe zu erleichtern, wurde Bootstrap als Standard Library eingebaut, damit es von den Testpersonen verwendet werden konnte. Außerdem wurden die Dateien für das Erstellen der `Atome` sowie das Einbinden der Dateien in einen Organismus für `PersonB` zuvor angelegt. Für ein realistisches Testbeispiel wurde für die Testaufgabe das Umsetzen einer Login Form verwendet (siehe Abbildung 7.1). Da `PersonA` sich um die `Atome` kümmert, ist sie für das Umsetzen der zwei Input-Felder und der drei verschiedenen Buttons zuständig. `PersonB` kümmert sich daher um das Rundherum. Sie ist dafür zuständig, den Verlauf im Hintergrund zu erstellen und die Login-Box im Ganzen sowie die kleine Ecke auf der rechten Seite der Box zu realisieren. Die Abstände der Komponenten werden von den zwei Personen gemeinsam erstellt. Dabei müssen sie sich absprechen. Da das Umsetzen einer maßgeschneiderten Checkbox ein etwas höherer Aufwand ist, wurde diese Komponente ausgeschlossen und der Testperson, die zuerst mit allem fertig ist, überlassen, sofern noch Zeit war. Für die Aufgabe wurde ein Zeitlimit von 30 Minuten vorgegeben.

Um den Testablauf immer gleich zu gestalten, wurde eine Testangabe verfasst (siehe Anhang A). Die Anleitung besteht aus drei Seiten. Die erste Seite beinhaltet den Zugang zum Testserver, der auf einem Rootserver gehostet ist. Die zweite Seite der Angabe beinhaltet die Beschreibung der Tasks für die Personen. Auf der dritten Seite der Testanleitung ist das Screendesign von Abbildung 7.1 zu finden, welches umzusetzen ist.

7.2.2 Aufbau der Testumgebung

Bei den Tests war es wichtig, für alle KandidatInnen die gleiche Testumgebung zu schaffen. Da einige der Probanden örtlich von einander getrennt waren, war es notwendig, eine klare Anleitung (siehe Abschnitt 7.2.1) und vorgegebene Voraussetzungen für die Testumgebung zu schaffen. Um das automatische Hochladen für jeden User bzw. jede Userin gleich zu gestalten, wurde PhpStorm ab der Version 2016 als IDE vorausgesetzt. Als Browser wurde Chrome ab der Version 56 vorausgesetzt. Zum Kommunizieren während des Arbeitens wurde in den meisten Fällen Skype genutzt. Jeder Proband hatte zwei Bildschirme zur Verfügung und hatte die IDE und das Browserfenster gleichzeitig offen. Außerdem waren sie instruiert, immer den Komponenten, welchen sie gerade bearbeiteten, im Browser anzusehen, um von der Echtzeit-Aktualisierung zu profitieren. Während des Tests konnten sowohl Fragen an den Testleiter gestellt werden als auch mit dem zweiten Probanden kommuniziert werden. Dies schaffte eine Atmosphäre des Pair-Programming. Der Aufbau der Testumgebung wurde bei jedem Testablauf gemeinsam aufgesetzt und vor dem Testen kontrolliert. Nach jedem Test wurde die Serverumgebung zurückgesetzt.

7.2.3 Testablauf

Der Testablauf verlief in allen Fällen gleich. Zuerst bekamen die KandidatInnen die Testangabe (siehe Abschnitt 7.2.1). Danach erfolgte eine kleine Einführung in das StyleLab-Tool. Wenn die KandidatInnen extern gearbeitet haben, wurde meist über Screensharing das Einrichten des Testservers vorgezeigt. Anschließend wurde gemeinsam die Testumgebung getestet, damit alle KandidatInnen die gleichen Voraussetzungen vorfanden. Danach wurde sichergestellt, dass die KandidatInnen bereit waren und die Aufgabe verstanden hatten. Nun hatten sie 30 Minuten Zeit, die Testaufgabe zu erfüllen und das StyleLab-Tool zu testen. Nach Erfüllen der Aufgabe oder Ablauf der Zeit wurde den KandidatInnen der Testfragebogen (siehe Anhang B) zur Verfügung gestellt. Das Ausfüllen des Fragebogens hat den Test beendet.

7.2.4 Auswertungsverfahren

Um den Testfragebogen auszuwerten, wurde der Dienst von Umfrageonline[1] verwendet. Hier können die Daten in Prozentanteilen visuell dargestellt werden, um so Schlüsse ziehen zu können (siehe Anhang C). Die Ergebnisse der Umfrage sind anonym und können daher nicht zurückverfolgt werden. Jede Antwort wird jedoch einer abstrakten Person zugewiesen. Dies lässt alle Antworten einer Testperson zurückverfolgen, um somit genauere Schlüsse zu ziehen. Außerdem kann über die gegebenen Antworten nach gewissen Rastern gefiltert werden. Dies bedeutet, dass zum Beispiel über eine Antwort der KandidatInnen gefiltert werden kann, sodass alle weiteren Antworten nach einem gewissen Schema eingeteilt werden können und zum Beispiel nur die weiteren Antworten aller Personen, welche die gleiche Antwort bei dieser Frage gegeben haben, erscheinen.

7.3 Annahme

Die Annahme für die Ergebnisse der Umfrage ist, dass die Testumgebung für das gemeinschaftliche Entwickeln einer Frontend-Umgebung von der Mehrheit der TeilnehmerInnen als hilfreich empfunden wird. Außerdem soll festgestellt werden, ob die Herangehensweise an eine Entwicklung des Frontends über das Atomic Design Konzept bekannt ist und nach Einschätzung der EntwicklerInnen als hilfreich erachtet wird. Mittels Fragen zu Gewohnheiten der EntwicklerInnen und Erkundigungen nach Erfahrungen mit dem Erstellen von Websites soll darüber hinaus festgestellt werden, ob die umgesetzten Maßnahmen den Entwicklungsstil der KandidatInnen unterstützen.

7.4 Ergebnisse

Die Auswertung der Fragen wurde mit Hilfe von Umfrageonline, wie in Abschnitt 7.2.4 beschrieben, erstellt. Zwölf der 22 TeilnehmerInnen, also 54.54% der KandidatInnen, waren Mitarbeiter der Firma elements und waren mit dem StyleLab-Tool schon im Vorfeld vertraut. Deshalb ist bei den Ergebnissen zu beachten, dass diese TeilnehmerInnen vermutlich die Aufgabe etwas leichter lösen konnten als TeilnehmerInnen, die mit dem StyleLab-Tool noch nie zuvor gearbeitet haben.

[1] www.umfrageonline.com

7.4.1 Auswertung der Fragen

Im diesem Abschnitt werden die Antworten der einzelnen Fragen aufgelistet. Zuerst werden die Absichten der Fragen besprochen. In weiterer Folge werden die dazugehörigen Antworten evaluiert.

Frage 1: Welcher Testgruppe haben Sie angehört?
Diese Frage wurde gestellt, um sicherzustellen, dass die Aufteilung der TestkandidatInnen in zwei Gruppen funktioniert hat. Hier wurde, wie vor dem Test angenommen und eigentlich vorausgesetzt, eine gleichmäßige Aufteilung erzielt. Durch diese Frage lässt sich nachvollziehen, welcher Kandidat bzw. welche Kandidatin vom Code des Partners abhängig war und welche oder welcher nicht.

Frage 2: Wie gut schätzen Sie Ihre Frontend-Entwicklungskenntnisse ein?
Über 85% der TeilnehmerInnen haben ihre Entwicklungskenntnisse als mindestens leicht fortgeschritten eingestuft. Außerdem gab es drei AnfängerInnen in der Testgruppe, was 13,6% der TeilnehmerInnen entspricht. Dies könnte bedeuten, dass sich die Gesamtgruppe der TeilnehmerInnen eher leicht mit der Testaufgabe zurechtgefunden hat. Außerdem dürfte dies auch die Verwendung des Tools für die TeilnehmerInnen erleichtert haben, da die meisten mehr Entwicklungs-Grundkenntnisse besaßen. Der Grund für diese Verteilung ist wohl die Einschränkung der TeilnehmerInnen auf das bereits erwähnte spezifische TeilnehmerInnenfeld (siehe Abschnitt 7.1.2).

Frage 3: Haben Sie vor diesem Tool schon von Atomic Design gehört?
Um Klarheit darüber zu erhalten, wie viele TeilnehmerInnen schon mit dem Prinzip des Atomic Designs vertraut waren, wurde diese Frage mit in die Untersuchung aufgenommen. 72,7% der Befragten wussten im Vorhinein schon über Atomic Design Bescheid. Dies könnte natürlich mit der hohen Anzahl an TeilnehmerInnen, die bei der Firma elements arbeiten, zu tun haben. Durch die hohe Prozentzahl ist wohl auch sichergestellt, dass sich die meisten Testpersonen in dem Tool leicht zurechtfinden konnten.

Frage 4: Haben Sie vor Verwenden dieses Tools schon nach der Atomic Design Methodik eine Website erstellt?
Diese Frage wurde den TeilnehmnerInnen gestellt, um zu erheben, inwieweit die Probanden schon mit der Atomic Design Methodik in ihrer Entwicklungsumgebung gearbeitet haben. Diese Frage ergab, dass 54,5% noch nie mit der Atomic Design Methodik eine Website erstellt haben. Das bedeutet, dass dieser Begriff mit 72.7% Verbreitung, wie in Frage drei beschrieben wird, bereits bekannt ist, jedoch die Anwendung dieser Methodik noch nicht als Standard zum Entwickeln angesehen und verwendet wird.

Frage 5: Sind Sie geübt im gemeinsamen Entwickeln eines Frontends mit einer oder mehreren ProgrammierpartnerInnen?
Die Probanden wurden auch gefragt, ob sie mit dem gemeinsamen Entwickeln im Frontend-Bereich mit einer oder mehreren Personen als PartnerInnen beim Programmierern vertraut sind. 63,6% der TeilnehmerInnen beantworteten diese Frage damit, dass sie

	sehr zufrieden (1)		zufrieden (2)		eher zufrieden (3)		eher unzufrieden (4)		unzufrieden (5)		gar nicht zufrieden (6)			
	Σ	%	Σ	%	Σ	%	Σ	%	Σ	%	Σ	%	Ø	±
Benutzerfreundlichkeit	5x	22,73	15x	68,18	2x	9,09	-	-	-	-	-	-	1,86	0,56
Übersicht der Komponenten	9x	40,91	11x	50,00	2x	9,09	-	-	-	-	-	-	1,68	0,65
Gestaltung des Tools	8x	36,36	7x	31,82	7x	31,82	-	-	-	-	-	-	1,95	0,84
Reaktionszeit des Tools bei Änderung in IDE	12x	54,55	6x	27,27	4x	18,18	-	-	-	-	-	-	1,64	0,79

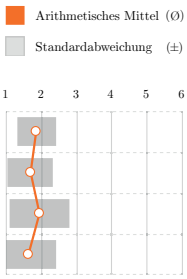


Abbildung 7.2: Auswertung zu Frage 9.

nicht darin geübt sind, zusammen mit anderen EntwicklerInnen im Frontend zu arbeiten. Mit dieser Testgruppe lässt sich in weiteren Schritten evaluieren, ob diesem TeilnehmerInnenbereich die Applikation beim gemeinsamen Entwickeln geholfen hat.

Frage 6: Wie oft testen Sie eine Komponente beim Entwickeln durch einen Reload der Seite?
In dieser Frage ging es um die Angewohnheiten der Testpersonen beim Testen während des Entwickelns. Zunächst wurde näher darauf eingegangen, wie oft die Befragten Komponenten durch einen Reload der Seite testen. 81,8% der TeilnehmerInnen sagten aus, dass sie nach jeder Änderung einer Komponente die Seite aktualisieren. Dies könnte die anfängliche Annahme unterstützen, dass es eine Hilfestellung ist, wenn die Komponenten automatisch bei einer Änderung neu geladen werden. Damit könnte den EntwicklerInnen dieser Schritt abgenommen werden. Diese Annahme wird in Frage 10 noch genauer untersucht.

Frage 7: Wie sieht Ihre momentane Entwicklungsumgebung aus? (Tools, IDE, Pipeline)
Hier wurde eine offene Frage gestellt, damit jeder Entwickler und jede Entwicklerin seine bzw. ihre Tools, IDE und Pipeline nach Belieben beschreiben konnte. Leider wurde diese Frage von den Probanden nicht sehr detailliert beantwortet. Was sich aber bei den Antworten klar herausstellt, ist, dass die meisten der Befragten PhpStorm verwenden. Außerdem konnte über diese Antworten auch die Umfrage von Ashley Nolan (siehe Tabelle 4.1) untermauert werden. Die meisten Probanden haben angegeben, eine Form von Task-Runner oder Tooling für ihre Frontend-Entwicklung zu verwenden.

Frage 8: Sind Sie mit Ihrer momentanen Entwicklungsumgebung zufrieden? Wenn nicht, warum?
Diese Frage wurde von 90,9% der TeilnehmerInnen damit beantwortet, dass sie mit der Entwicklungsumgebung, mit der sie momentan arbeiten, zufrieden sind. Dies könnte aber auch damit zu tun haben, dass viele der EntwicklerInnen nicht evaluieren, was bei ihrer Umgebung verbessert werden könnte.

	sehr hilfreich (1)		hilfreich (2)		eher hilfreich (3)		eher nicht hilfreich (4)		nicht hilfreich (5)		gar nicht hilfreich (6)			
	Σ	%	Σ	%	Σ	%	Σ	%	Σ	%	Σ	%	Ø	±
Für Ihren eigenen Code	12x	54,55	7x	31,82	1x	4,55	2x	9,09	-	-	-	-	1,68	0,95
Für Code Ihres Partners	8x	36,36	5x	22,73	5x	22,73	4x	18,18	-	-	-	-	2,23	1,15
Zum Testen der Website generell	8x	36,36	9x	40,91	3x	13,64	2x	9,09	-	-	-	-	1,95	0,95

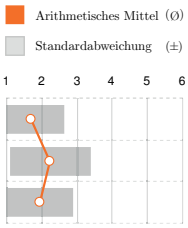


Abbildung 7.3: Auswertung zu Frage 10.

Frage 9: Wie würden Sie Ihre Erfahrung mit dem StyleLab-Tool als Ganzes benoten?
Bei dieser Frage wurde näher auf die Erfahrung der TeilnehmerInnen mit dem StyleLab-Tool als Ganzes, besonders auf Funktionalitäten und Präsentation des Tools, eingegangen. Die Beurteilung der Fragen konnte mit einer Notenskala von 1 (sehr zufrieden) bis 6 (gar nicht zufrieden) bewertet werden.

Wie man der Abbildung 7.2 entnehmen kann, wurden die TeilnehmerInnen zuerst nach ihrer Zufriedenheit mit der `Benutzerfreundlichkeit` des Tools gefragt. Hier zeigt sich, dass alle TeilnehmerInnen mindestens `eher zufrieden` bis `sehr zufrieden` mit der Benutzerfreundlichkeit sind, wobei der Durchschnitt, also der arithmetische Mittelwert, bei der Note 1,86 liegt.

Im nächsten Abschnitt wurde die `Übersicht der Komponenten`, welche das StyleLab-Tool bietet, bewertet. Auch hier wurde mit mindestens `eher zufrieden` bis `sehr zufrieden` bewertet. Hier liegt der arithmetische Mittelwert etwas niedriger, und zwar bei 1,68, da mehr KandidatInnen mit `sehr zufrieden` gestimmt haben.

Auch bei `Gestaltung des Tools` wurde von den KandidatInnen zwischen `eher zufrieden` bis `sehr zufrieden` bewertet. Dieser Abschnitt hat den schlechtesten arithmetischen Mittelwert in dieser Frage, mit einer Note von immerhin noch 1,95. Um diese Antwort weiter untersuchen zu können, können Frage 12 und Frage 16 herangezogen werden.

Zuletzt wurde bei dieser Frage die `Reaktionszeit des Tools bei Änderungen in der IDE` bewertet. Wie auch bei allen zuvor bewerteten Abschnitten liegt die Spanne der Antworten zwischen `eher zufrieden` bis `sehr zufrieden`. Hier wurde der beste arithmetische Mittelwert mit 1,64 erreicht.

Frage 10: Wie hilfreich fanden Sie das automatische Aktualisieren der zu bearbeitenden Komponente?
Diese Frage beschäftigt sich damit, wie hilfreich das automatische Aktualisieren für die Probanden war. Um verschiedene Aspekte des automatischen Aktualisierens zu analysieren, ist die Frage wie in Abbildung 7.3 in drei Unterpunkte unterteilt. Die Beurteilung der Fragen konnte auch hier mit einer Notenskala von 1 (sehr hilfreich) bis 6 (gar nicht hilfreich) bewertet werden.

Als erstes wurden die TeilnehmerInnen gefragt, ob das automatische Aktualisieren der Website für das Testen des eigenen Codes hilfreich war. Hier verlief die Spanne der Antworten von 1 (sehr hilfreich) bis 4 (eher nicht hilfreich). Trotz dieser größeren Spannweite im Gegensatz zu Frage neun wurde ein arithmetischer Mittelwert von 1.68 ermittelt. Dies bedeutet, dass das automatische Aktualisieren für den eigenen Code

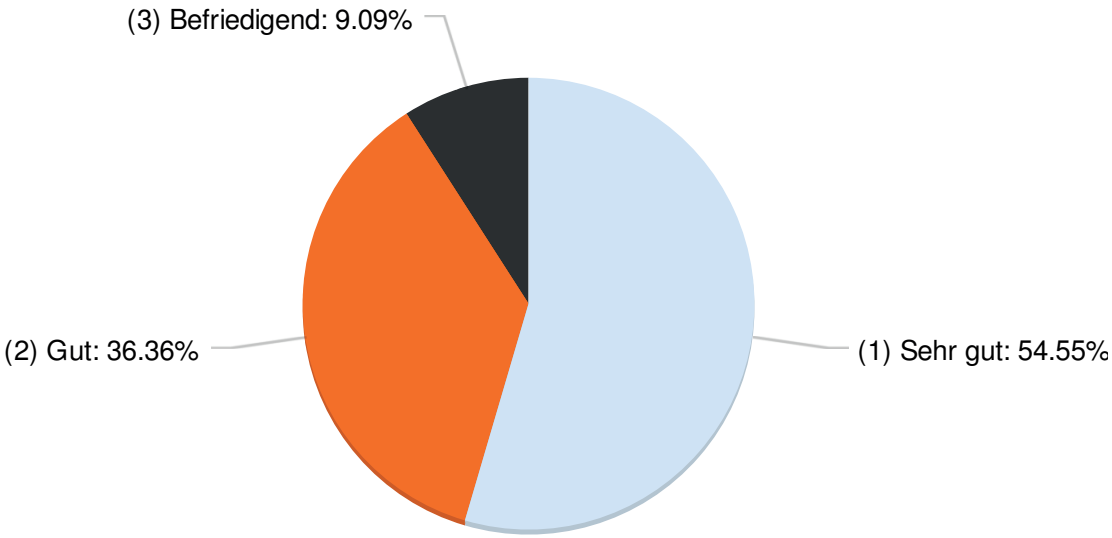


Abbildung 7.4: Auswertung der Frage 11.

der EntwicklerInnen großteils als sehr hilfreich empfunden wurde, es jedoch durch die größere Standardabweichung auch Ausnahmefälle gibt. Dieses Ergebnis beweist, dass das automatische Aktualisieren des eigenen Codes einen eindeutigen unterstützenden Vorteil für den Entwickler bzw. die Entwicklerin mit sich bringt.

Die Frage *für den Code ihres Partners* untersucht die Auswirkungen des Aktualisierens des Codes auf die Zusammenarbeit zwischen den EntwicklerInnen. Das Ergebnis zeigt einen schlechteren arithmetischen Mittelwert von 2,23. Auch hier wurde zwischen 1 (sehr hilfreich) und 4 (eher nicht hilfreich) bewertet. Die höhere Standardabweichung von 1.15 und der Umstand, dass über ein Drittel der Befragten schon sehr zufrieden mit den Features waren, zeigt die große Divergenz zwischen den EntwicklerInnen. Um diese Meinungsverschiedenheit genauer zu untersuchen, wurden die TeilnehmerInnen, die diesen Abschnitt mit eher nicht hilfreich bewertet haben, näher untersucht. Überraschenderweise haben 75% dieser EntwicklerInnen der Testgruppe A angehört, welche nicht vom Code des Partners abhängig waren, und auch nicht vom automatischen Aktualisieren des Partners betroffen waren. Beim genaueren Betrachten der Personen, die das automatische Aktualisieren als eher hilfreich (Note 3) ansahen, gehörten immer noch 60% der Gruppe A an. Die TeilnehmerInnen, die diesen Abschnitt jedoch mit sehr hilfreich (Note 1) bewerteten, gehörten hauptsächlich der Gruppe B an, welche durch das automatische Aktualisieren der Seite durch den Code des Partners betroffen war. Daraus resultiert, dass jene TeilnehmerInnen, die diesen Abschnitt weniger positiv bewertet haben, nicht vom automatischen Aktualisieren abhängig waren und auch nicht von dieser Funktionalität des Aktualisierens profitieren hätten können. Das bedeutet, dass wenn ein Entwickler bzw. eine Entwicklerin von der Funktion des automatischen Aktualisierens des Codes des Partners bzw. der Partnerin profitieren konnte, er oder sie dies als hilfreich empfand. Hingegen konnte ein Entwickler bzw. eine Entwicklerin, dessen bzw. deren Code nicht auf dem Code des Partners bzw. der Partnerein aufbaute und somit keine Aktualisierungen vorfand, diesen Abschnitt nicht mit hilfreich benoten,

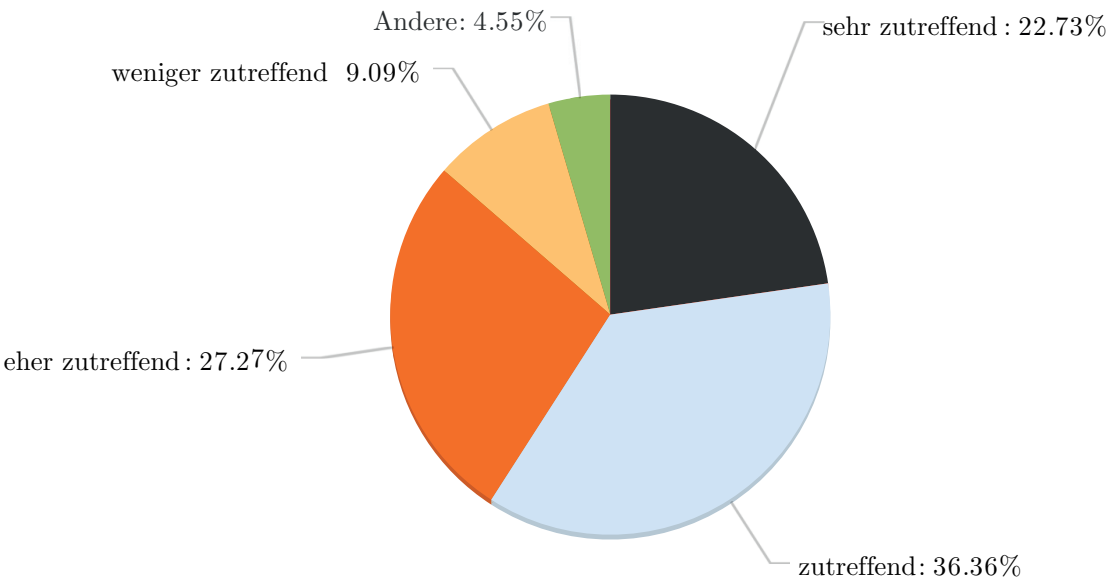


Abbildung 7.5: Auswertung der Frage 12.

da es für ihn oder sie nicht präsent war.

Beim dritten Abschnitt ging es um das Testen der Website generell mit Hilfe des automatischen Aktualisierens. Hier wurde, wie bei den vorherigen Fragen, zwischen 1 (sehr hilfreich) und 4 (eher nicht hilfreich) bewertet. Wie in Abbildung 7.3 zu sehen ist, ergibt sich ein arithmetisches Mittel von 1,95. Das bedeutet, dass das Aktualisieren auch zum Testen der Website geeignet ist. Dies könnte damit zu tun haben, dass der Zustand durch das automatische Injizieren von Code wie in Abschnitt 5.3.1 beschrieben, nicht verändert wird und somit das Testen erleichtert.

Frage 11: Wie half Ihnen die Unterteilung der Komponenten nach der Atomic Design Methodik während des Entwickelns?
Wie in Abbildung 7.4 zu sehen ist, bewerteten über 50% der TeilnehmerInnen die Unterteilung der Komponenten nach der Methode des Atomic Designs mit sehr gut (Note 1). Der Notendurchschnitt dieser Frage bei einer Skala von 1 bis 6 ist 1,55. Dies bedeutet, dass die Unterteilung des StyleLabs nach der Atomic Design Methodik hilfreich ist.

Frage 12: Heben sich die erstellten Komponenten ausreichend vom Grunddesign des StyleLabs ab?
Wie in Abbildung 7.5 zu sehen ist, ist die Abweichung der Antworten größer als bei allen bisherigen Antworten. Die Frage, die offen beantwortet wurde, könnte Aufschluss über den Antagonismus der Antworten geben. Die offene Antwort besagt: „Auf den Detailseiten sehr gut, in der Übersicht nicht so“. Wie in Abbildung 6.3 zu sehen ist, wurde im Gegensatz zur Abbildung 5.3 die Unterteilung der Module auf der Übersichtsseite klarer getrennt, was aber offensichtlich noch nicht genug war. Hier könnten wohl noch weitere Optimierungen vorgenommen werden, um Module klarer von einander zu trennen. Generell wurde diese Frage jedoch positiv beurteilt.

Frage 13: Würden Sie etwas bei der Echtzeit-Aktualisierungsfunktion verändern? Wenn ja, was?
Die TeilnehmerInnen wurden bei dieser Frage dazu aufgefordert, Veränderungsmöglichkeiten bei der automatischen Aktualisierung vorzuschlagen, falls solche als notwendig und hilfreich empfunden wurden. Eine wichtige Anmerkung war, dass es praktisch wäre, zu sehen, wann und von wem die letzte getätigte Aktualisierung vorgenommen wurde. Manche Probanden wollten sogar einen detaillierten Log für Änderungen. Auch würden sich manche wünschen, dass es möglich wäre, dass nur ihr Code aktualisiert wird.

Frage 14: Was hat Ihnen an diesem Tool am besten gefallen?
Um zu eruieren, welche Funktionen am besten angekommen sind, wurden die TeilnehmerInnen gefragt, was ihnen am Tool gefallen hat. Den meisten Probanden hat das automatische Aktualisieren des Browsers ohne manuelles Drücken einer Taste am besten gefallen. Außerdem war es für viele KandidatInnen hilfreich, dass es ein Popup-Fenster gibt, welches zeitnahe Information über die Aktualisierung liefert. Dass die einzelnen Komponenten nach der Atomic Design Methode aufgeteilt dargestellt werden und dass dies zu einer verbesserten Übersichtlichkeit der Elemente führt, wurde ebenfalls als eine sehr hilfreiche Funktion bewertet.

Frage 15: Haben Sie Funktionen vermisst? Wenn ja, welche?
Diese Frage beschäftigte sich damit, um welche Funktionen die Befragten das Tool noch erweitern würden oder welche Funktionen sie sich noch für das Tool wünschen. Mehrmals wurde angegeben, dass es eventuell hilfreich wäre, die zuletzt veränderten Komponenten absteigend in einer Liste anzugeben, wie schon bei Frage 13 beschrieben wurde (siehe Abbildung 7.4.1). Auch das momentane Stoppen des automatischen Aktualisierens durch Veränderungen des Partners, damit sein Zustand damit nicht betroffen ist, wäre hilfreich. Ein weiterer Vorschlag war es, das ganze System um eine zusätzliche Komponente zu erweitern, die die Updates, welche im StyleLab sichtbar sind, auch in der IDE anzeigt und somit auch der Code am Client synchronisiert wird. Dies wäre zum Beispiel mit einem PhpStorm-Plugin möglich.

Frage 16: Was hat Sie beim Benutzen des Tools am meisten gestört?
Eine weitere offene Frage beschäftigte sich damit, was den TeilnehmerInnen beim Benutzen des Tools nicht gefallen hat, beziehungsweise, was sie gestört hat. Aus dem Feedback der TeilnehmerInnen lässt sich schließen, dass auch nur eine kleine Notification anstatt eines Popup Elements zur Information der Aktualisierung ausreichen würde. Auch wurde von manchen Probanden das Wackeln des gerenderten Views als störend empfunden, welches von der CSS-Injection oder dem Neuladen der Webseite kommen könnte. Weiters wurde noch bemerkt, dass es verwirrend war, dass durch Fehler im Code des Partners die Änderung der Website als eigener Fehler angenommen wurde. Die anfängliche Darstellung der Website und die Konfiguration der Website waren für manche TeilnehmerInnen auch verwirrend.

Frage 17: Kennen Sie vergleichbare Tools?
Auf die Frage, ob die TeilnehmerInnen ähnliche Tools benennen können, antworteten 84,4%, dass sie auch ein vergleichbares Tool kennen. Diese Tools waren vor allem das StyleLab von elements und patternlab.

Frage 18: Bitte geben Sie Ihr Alter an
Diese Frage sollte Aufschluss darüber geben, wie groß der Anteil der TeilnehmerInnen war, von denen angenommen werden kann, dass sie altersbedingt schon über längere Programmiererfahrung verfügen. Das Durchschnittsalter der Probanden betrug 25,77 Jahre. Dies zeigt, dass wohl die meisten der EntwicklerInnen schon Erfahrung haben.

Frage 19: In welchem Tätigkeitsbereich sind Sie tätig?
18,2% der Befragten gaben an, dass sie im Frontend tätig sind. 50,0% sind dagegen im Backend tätig. Die restlichen 32,8% der TeilnehmerInnen arbeiten in anderen Bereichen, welche meist übergreifende Bereiche zwischen Frontend und Backend waren. Keiner der Befragten ist im Design beschäftigt.

7.4.2 Zusammenfassung und Interpretation der Antworten:

Nach genauerer Analyse der Fragen und Reflexion der Ursprungshypothese ist das Ergebnis der Evaluierung als sehr positiv einzuschätzen. Die Erweiterung des StyleLab um die Real-Time-Funktionen hatte im Großen und Ganzen einen positiven Effekt auf die EntwicklerInnen. Auch private Gespräche nach dem Testen haben ergeben, dass die Mehrheit der TestteilnehmerInnen für das Integrieren der Funktionalität in die elements Frontend-Build-Pipeline ist. Durch das Testen der Umgebung und das Evaluieren der Fragen ist aufgefallen, dass vor allem durch das Injizieren von Code der Zustand der Applikation erhalten bleibt. Dies hilft beim Testen komplexerer Komponenten. Es ist zu vergleichen mit dem Verändern des Codes direkt in den Chrome-Entwicklertools. Der klare Vorteil ist jedoch, dass dies bei allen geöffneten Browsern gleichzeitig funktioniert und somit das Testen einfacher und globaler ist. Außerdem ist aufgefallen, dass die Kommunikation und der Austausch zwischen den EntwicklerInnen während des Testens intensiver war als sonst. Der Vorschlag für das Hinzufügen eines *STOP*-Buttons, welcher das Neuladen, beziehungsweise Injizieren des Codes des Partners verhindert, könnte möglicherweise ein sehr hilfreiches Feature sein, welches die Produktivität weiter fördern könnte.

Kapitel 8

Zusammenfassung und Ausblick

Die Aufgabenstellung dieser Arbeit war das Erstellen eines Prototypen, der eine Frontend-Build-Pipeline um Real-Time-Funktionen erweitern soll. Außerdem sollte bewiesen werden, dass das Hinzufügen einer solchen Real-Time-Funktion zu der schon existierenden Frontend-Build-Pipeline einen Vorteil beim Entwickeln eines Frontends bietet. Retrospektiv und zusammenfassend kann durch das Auswerten der Testergebnisse festgestellt werden, dass das Neuladen insgesamt von den testenden und befragten EntwicklerInnen als positive und hilfreiche funktionale Erweiterung wahrgenommen wurde. Dieses Ergebnis beantwortet die anfangs gestellte Fragestellung.

In der Zukunft wird der Prototyp sicherlich weiter entwickelt werden, da das Resultat vielversprechend ist. Einige der von den Testern vorgeschlagenen Änderungen, wie zum Beispiel das Blockieren von Aktualisierungen anderer EntwicklerInnen, könnten eingebaut werden. Auch eine Möglichkeit der Anzeige eines Bearbeitungsprotokolls für jede einzelne Komponente ist denkbar. Um die Abhängigkeit von anderen Modulen zu minimieren, sollte auf BrowserSync verzichtet werden, da die Grundfunktionen leicht nachimplementiert werden können und dadurch einfacher wartbar und erweiterbar werden. Bei der Anzeige der Module im StyleLab ohne dessen Grundfunktionen und äußeren Grundgerüsts ist ein Fehler im Prototyp festzustellen, welcher erst nach dem Testen bemerkt wurde. Hier wird von Pimcore in der URL keine `UID` mitgeschickt. Somit können der Websocket-Verbindung keine Dateien zugeordnet werden und ein Neuladen der Seite ist nicht möglich. Um diesen Fehler zu beheben, genügt es, die `UID` erst im Node-Proxy-Server zu erstellen und dem Client mitzugeben.

Anhang A

Testangabe

Für die Evaluierung wurde eine Testangabe erstellt. Mit Hilfe dieser Testangabe sollte sichergestellt werden, dass alle KandidatInnen die gleiche Ausgangsposition haben. Auf den folgenden drei Seiten ist die Testangabe angefügt. Sie wurde den Probanden im Querformat vorgelegt. Auf der ersten Seite ist der Zugang zum Testserver beschrieben. Außerdem ist eine Abbildung auf dieser Seite vorzufinden, welche als Anleitung für die Einrichtung des externen Testservers in der PhpStorm IDE dient. Die zweite Seite beschreibt die Testaufgabe der Probanden. Im Anschluss ist das umzusetzende Design für die Probanden abgebildet. Die Probanden sollen sich während des Umsetzens an die Vorgaben halten.

Evaluierung – Collaborative Frontend and Auto-refresh

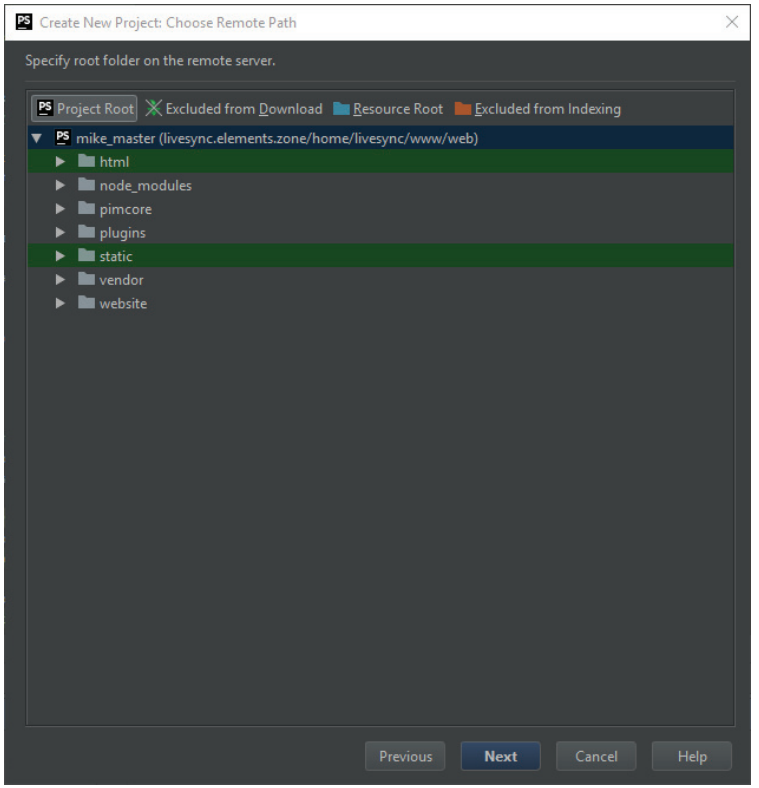
SFTP:

Bitte NUR die Ordner **HTML** und **STATIC** herunterladen.

Host: kuss.eu

User: rudisch

Password: m7VUrkShyqmepEyuW9rUuH4R



Stylelab Web:

http://thesis.kuss.eu:3000/stylelab

Bitte NUR über den PORT 3000 auf das Stylelab zugreifen.

Test Setup:

2 Personen arbeiten **GLEICHZEITIG** am gleichen Projekt im Stylelab. Es ist immer die **IDE** und gleichzeitig das **Browser-Fenster** mit der Repräsentation der momentan bearbeiteten Komponente zu öffnen. Nach jeder kleinen Änderung ist STRG+S zu drücken um das File auf den Server zu laden. Der Web-view sollte automatisch neu geladen werden.

Aufgabe:

Person A:

Sie arbeiten ausschließlich im Ordner *html/00-atoms* und sind für die Erstellung der Atome zuständig. Das bedeutet, Sie setzen das CSS für die Input-Felder und die 3 Buttons um, welche in Ihrem Ordner schon angelegt wurden. Das Zusammenfügen der Atome und Generieren des Rests wird von PersonB erledigt. Das CSS-File für die Bearbeitung befindet sich unter *static/css/personA.css*

Person B:

Ihre Aufgabe ist es, mit den von PersonA erstellten Atomen zu arbeiten, diese NICHT zu verändern, sondern nur zu verwenden, um die Login-Form zu erstellen. Ihre Aufgabe besteht darin, diese Komponenten, Inputs und Buttons, in der richtigen Weise zu verwenden und das Drumherum zu erstellen, also den Hintergrund und die Form in der Mitte. Das CSS-File für die Bearbeitung befindet sich unter *static/css/personB.css*

Dauer: ~30 Minuten

Personen: 2

Nach dem Test gibt es eine kurze Umfrage: https://www.umfrageonline.com/s/69831a4


LOGIN
EMAIL
PASSWORD
Remember me
LOGIN

Anhang B

Fragebogen

Um die Erfahrung der TestkandidatInnen mit dem StyleLab-Tool zu evaluieren, wurde ein Fragebogen erstellt. Dieser Fragebogen war online über die Webseite Umfrageonline zu erreichen. Alle Probanden haben diesen Fragebogen über das Online-Portal ausgefüllt. Auf den folgenden Seiten ist der Fragebogen angefügt.

Stylelab Usability Test

Generelle Information

Vielen Dank für das Testen des Stylelabs.
Die kommenden Fragen bewerten Ihre Erfahrung mit dem Tool.

Welcher Testgruppe haben Sie angehört?

- ◯ A
- ◯ B

Wie gut schätzen Sie Ihre Frontend Entwicklungskentnisse ein?

- ☐ Experte
- ☐ Fortgeschritten
- ☐ Leicht Fortgeschritten
- ☐ Anfänger
- ☐ _____

Haben Sie vor diesem Tool schon von Atomic Design gehört?

- ◯ ja
- ◯ nein

Haben Sie vor Verwenden dieses Tools schon nach der Atomic Design Methodik eine Website erstellt?

- ◯ ja
- ◯ nein

Sind Sie geübt im gemeinsamen Entwickeln eines Frontends mit einer oder mehreren ProgrammierpartnerInnen

- ◯ ja
- ◯ nein

Wie oft testen Sie eine Komponente beim Entwickeln durch einen Reload der Seite?

- ◯ nach jeder Änderung
- ◯ Nur nach einer größeren Änderung (manuell)
- ◯ In regelmäßigen Abständen (Automatisches Tool)
- ◯ nach Fertigstellen einer abgeschlossenen Komponente
- ◯ Anderes: [ ]

Wie sieht Ihre momentane Entwicklungsumgebung aus? (Tools, IDE, Pipeline)

[ ]

Sind Sie mit Ihrer momentanen Entwicklungsumgebung zufrieden? Wenn nicht, warum?

- ◯ Ja
- ◯ [ ]

Wie würden Sie Ihre Erfahrung mit dem Stylelab-Tool als Ganzes benoten?

	sehr zufrieden	zufrieden	eher zufrieden	eher unzufrieden	unzufrieden	gar nicht zufrieden
Benutzerfreundlichkeit	◯	◯	◯	◯	◯	◯
Übersicht der Komponenten	◯	◯	◯	◯	◯	◯
Gestaltung des Tools	◯	◯	◯	◯	◯	◯
Reaktionszeit des Tools bei Änderung in IDE	◯	◯	◯	◯	◯	◯

Wie hilfreich fanden Sie das automatische Aktualisieren der zu bearbeitenden Komponente?

	sehr hilfreich	hilfreich	eher hilfreich	eher nicht hilfreich	nicht hilfreich	gar nicht hilfreich
für IHREN EIGENEN Code	◯	◯	◯	◯	◯	◯
für Code Ihres Partners	◯	◯	◯	◯	◯	◯
zum Testen der Website generell	◯	◯	◯	◯	◯	◯

Wie half Ihnen die Unterteilung der Komponenten nach der Atomic Design Methodik während des Entwickelns?

◯ (1) Sehr gut

◯ (2) Gut

◯ (3) Befriedigend

◯ (4) Ausreichend

◯ (5) Mangelhaft

◯ (6) Ungenügend

Heben sich die erstellten Komponenten ausreichend von dem Grunddesign des Stylelabs ab?

- ◯ sehr zutreffend
- ◯ zutreffend
- ◯ eher zutreffend
- ◯ weniger zutreffend
- ◯ eher nicht zutreffend
- ◯ nicht zutreffend
- ◯ _____

Offenes Feedback

Würden Sie etwas bei der Echtzeit-Aktualisierungsfunktion verändern? Wenn ja, was?

- ◯ Nein
- ◯ Ja, _____

Was hat Ihnen an diesem Tool am besten gefallen?

_____

Haben Sie Funktionen vermisst? Wenn ja, welche?

Was hat Sie beim Benutzen des Tools am meisten gestört?

Kennen Sie vergleichbare Tools?

- ◯ Nein
- ◯ Ja,

Bitte geben Sie Ihr Alter an

In welchem Tätigkeitsbereich sind Sie tätig?

- ◯ Frontend
- ◯ Backend
- ◯ Design
- ◯ Anderes:

Anhang C

Auswertung

Zur Auswertung des Testfragebogens (siehe Anhang B) wurde der Dienst der Webseite Umfrageonline verwendet. Dieser Auswertungsmechanismus der Webseite Umfrageonline generiert automatisch Statistiken für die beantworteten Fragen der TestkandidatInnen. Hier können vorab schon einige Schlüsse gezogen werden und der Ausgang der Evaluierung begutachtet werden. Außerdem bietet dieses Tool das Verbinden von verschiedensten Aussagen der Kandidaten. So kann nach Aussagen gefiltert werden. Das im Folgenden angefügte PDF zeigt die Auswertung der Fragen und die dazugehörigen Statistiken.

Stylelab Usability Test

1. Welcher Testgruppe haben Sie angehört? *

   Anzahl Teilnehmer: 22

   11 (50.0%): A

   11 (50.0%): B

2. Wie gut schätzen Sie Ihre Frontend Entwicklungskentnisse ein? *

   Anzahl Teilnehmer: 22

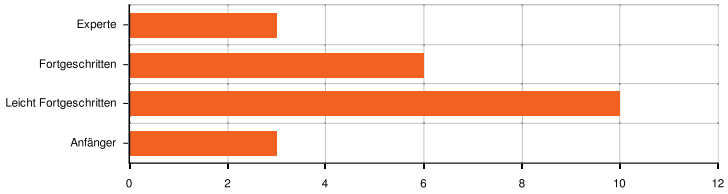
   3 (13.6%): Experte

   6 (27.3%): Fortgeschritten

   10 (45.5%): Leicht Fortgeschritten

   3 (13.6%): Anfänger

   - (0.0%): Andere

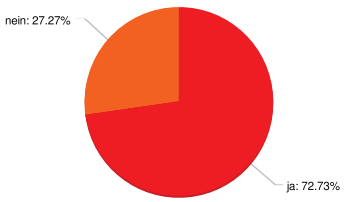


3. Haben Sie vor diesem Tool schon von Atomic Design gehört? *

   Anzahl Teilnehmer: 22

   16 (72.7%): ja

   6 (27.3%): nein

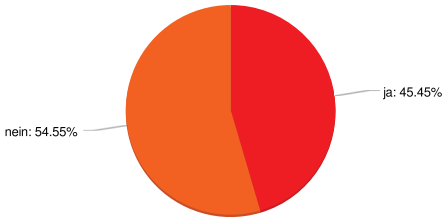


4. Haben Sie vor Verwenden dieses Tools schon nach der Atomic Design Methodik eine Website erstellt? *

   Anzahl Teilnehmer: 22

   10 (45.5%): ja

   12 (54.5%): nein

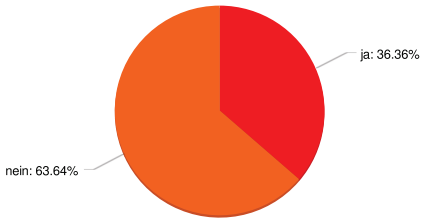


5. Sind Sie geübt im gemeinsamen Entwickeln eines Frontends mit einer oder mehreren ProgrammierpartnerInnen *

   Anzahl Teilnehmer: 22

   8 (36.4%): ja

   14 (63.6%): nein



6. Wie oft testen Sie eine Komponente beim Entwickeln durch einen Reload der Seite? *

Anzahl Teilnehmer: 22

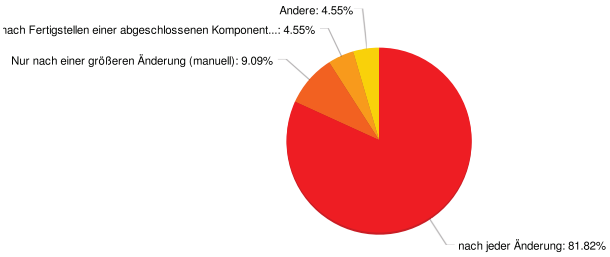
18 (81.8%): nach jeder Änderung

2 (9.1%): Nur nach einer größeren Änderung (manuell)

- (0.0%): In regelmäßigen Abständen (Automatisches Tool)

1 (4.5%): nach Fertigstellen einer abgeschlossenen Komponente

1 (4.5%): Andere



Antwort(en) aus dem Zusatzfeld:

- Meistens nach einem größeren HTML/CSS Block. Wenn es dann an den Feinschliff geht, dann auch mal nach jeder Änderung

7. Wie sieht Ihre momentane Entwicklungsumgebung aus? (Tools, IDE, Pipeline) *

Anzahl Teilnehmer: 22

- PHPStorm
- Composer, Gulp, Ember CLI, PHPStorm, Autorefresh bzw. F5
- Webstorm
- phpstorm, stylelab
- PHPStorm, Stylelab
- PHPStorm, Gulp, Angular-CLI, electron
- PhpStorm, Git, Gulp, Composer, Bootstrap
- PHPStorm, Stylelab
- PhpStorm, Notepad++, Chrome
- Visual Studio 2017
- JSF in Eclipse (Kepler) + JBoss AS
- PHPStorm, Gulp, Git, Composer, Putty
- phpstorm, putty, pimcore, composer, git,
- Phpstorm, Gulp, Postcss
- PhpStorm, Gulp, Pimpcore
- Vim, Chromium
- vi, qtcreator
- PHPStorm, Chrome
- PHP-Storm
- Php Storm, Web Storm
- PhpStorm
- Visual Studio Code, Github Atom, Jetbrains PyCharm, PHP Storm

Bootstrap, Ruby Middleman, Pyhton Django / Flask, Node with Nunchuck, Gulp

8. Sind Sie mit Ihrer momentanen Entwicklungsumgebung zufrieden? Wenn nicht, warum? *

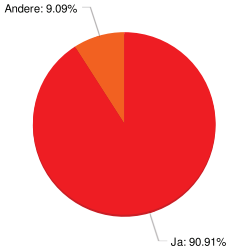
Anzahl Teilnehmer: 22

20 (90.9%): Ja

2 (9.1%): Andere

Antwort(en) aus dem Zusatzfeld:

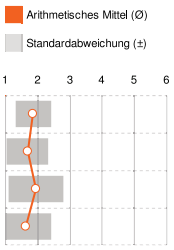
- es ist zu langsam, stürzt ab, integrierter Browser ist schlecht.
- kein dark theme, fehlende editor-features fuer front-end



9. Wie würden Sie Ihre Erfahrung mit dem Stylelab-Tool als Ganzes benoten? *

Anzahl Teilnehmer: 22

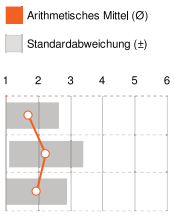
	sehr zufrieden (1)		zufrieden (2)		eher zufrieden (3)		eher unzufrieden (4)		unzufrieden (5)		gar nicht zufrieden (6)			
	Σ	%	Σ	%	Σ	%	Σ	%	Σ	%	Σ	%	Ø	±
Benutzerfreundlichkeit	5x	22,73	15x	68,18	2x	9,09	-	-	-	-	-	-	1,86	0,56
Übersicht der Komponen…	9x	40,91	11x	50,00	2x	9,09	-	-	-	-	-	-	1,68	0,65
Gestaltung des Tools	8x	36,36	7x	31,82	7x	31,82	-	-	-	-	-	-	1,95	0,84
Reaktionszeit des Tools…	12x	54,55	6x	27,27	4x	18,18	-	-	-	-	-	-	1,64	0,79



10. Wie hilfreich fanden Sie das automatische Aktualisieren der zu bearbeitenden Komponente? *

Anzahl Teilnehmer: 22

	sehr hilfreich (1)		hilfreich (2)		eher hilfreich (3)		eher nicht hilfreich (4)		nicht hilfreich (5)		gar nicht hilfreich (6)			
	Σ	%	Σ	%	Σ	%	Σ	%	Σ	%	Σ	%	Ø	±
für IHREN EIGENEN Code	12x	54,55	7x	31,82	1x	4,55	2x	9,09	-	-	-	-	1,68	0,95
für Code Ihres Partners	8x	36,36	5x	22,73	5x	22,73	4x	18,18	-	-	-	-	2,23	1,15
zum Testen der Website…	8x	36,36	9x	40,91	3x	13,64	2x	9,09	-	-	-	-	1,95	0,95



11. Wie half Ihnen die Unterteilung der Komponenten nach der Atomic Design Methodik während des Entwickelns? *

Anzahl Teilnehmer: 22

12 (54.5%): (1) Sehr gut

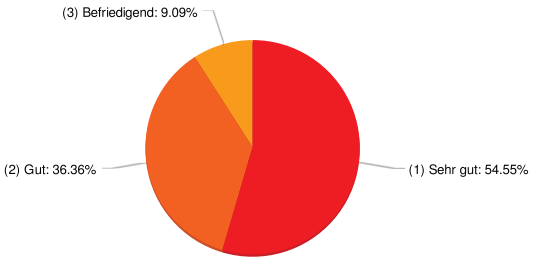
8 (36.4%): (2) Gut

2 (9.1%): (3) Befriedigend

- (0.0%): (4) Ausreichend

- (0.0%): (5) Mangelhaft

- (0.0%): (6) Ungenügend



12. Heben sich die erstellten Komponenten ausreichend von dem Grunddesign des Stylelabs ab? *

Anzahl Teilnehmer: 22

5 (22.7%): sehr zutreffend

8 (36.4%): zutreffend

6 (27.3%): eher zutreffend

2 (9.1%): weniger zutreffend

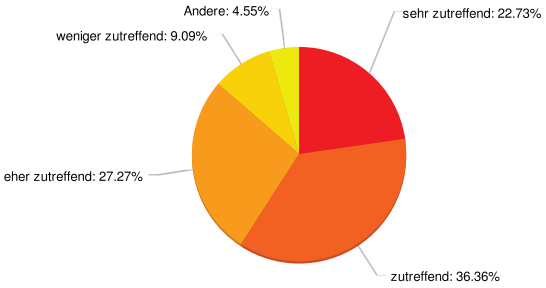
- (0.0%): eher nicht zutreffend

- (0.0%): nicht zutreffend

1 (4.5%): Andere

Antwort(en) aus dem Zusatzfeld:

- Auf den Detailseiten sehr gut, in der Übersicht nicht so



13. Würden Sie etwas bei der Echtzeit-Aktualisierungsfunktion verändern? Wenn ja, was? *

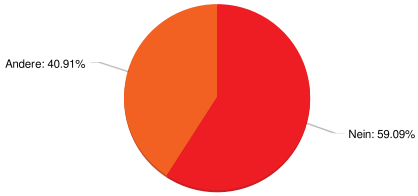
Anzahl Teilnehmer: 22

13 (59.1%): Nein

9 (40.9%): Andere

Antwort(en) aus dem Zusatzfeld:

- Evtl. etwas schneller.
- Sehen wer und wann die letzte Änderung vorgenommen hat.
- Wann wurde zuletzt aktualisiert und evtl. wer
- Letzte Aktualisierungszeit hinzufügen oder aktualisiert vor x Sekunden.
- Wenn möglich, die Aktualisierung unerbinden, wenn devtools aktiv sind. Gleichzeitiges Arbeiten an einer File ermöglichen.
- Aktualisierung nur für eigenen Code
- ON / OFF button
- Datein von Partner im Projekt als "Dirty" markieren
- controlling sync



14. Was hat Ihnen an diesem Tool am besten gefallen? *

Anzahl Teilnehmer: 22

- Kein manuelles Neu-Laden des Browser.
- CSS Injektion, erspart das kleine herumexperimentieren in den Dev Tools des Browsers, da die Änderung in der IDE genauso schnell geht, ohne auf F5 drücken zu müssen. Außerdem muss ich die IDE nicht mehr verlassen
- Pop Up Fenster mit Info der Aktualisierung
- automatischer reload
- auto refresh
- Die Aufteilung in die einzelnen kleinen Komponenten, während ich direkt an diesen Komponenten schrauben kann. Und dass ich meinem Partner live by der Arbeit zuschauen kann.
- Alles. Echtzeit war sowohl fürs Hot Code Reloading für mich alleine, als auch gemeinsam sehr hilfreich.
- Atomic Aufteilung
- Die Übersichtlichkeit der Funktionalitäten und Elemente.
- Immer der aktuelle Entwicklungsstand und die Möglichkeit die Breite an den Elementen zu testen.
- automatische aktualisierung der seite bei aenderungen, vorallem durch partner
- Die nahtlose Liveaktualisierung
- Echzeit-Aktualisierungsfunktion
- hot reloading
- Das automatische Aktualisieren der Files im Browser.
  Die Aufsplittung der einzelnen Komponenten nach dem Atomic Prinzip.
- .
- Benutzerfreundlichkeit
- Automatisches Aktualisieren der eigenen Änderungen
- nette spielerei mit automatischem reloaden
- Änderungen der anderen Beteiligten sofort ersichtlich
- simplicity and independent functionalities
- Übersichtlichkeit

15. Haben Sie Funktionen vermisst? Wenn ja, welche? *

Anzahl Teilnehmer: 22

- Keine.
- Für das Beispiel nicht. Musste nur auf dem Formular bleiben.
- genaue Beschreibung/History welche Komponente verändert worden ist
- nein
- phpstorm plugin erstellen damit man automatisch änderungen vom server bekommt
- Keine.
- Debugging Funktion: Evtl. eine Art Log zum ausklappen welche Files zuletzt verändert wurden.
- nein
- Ich habe keine Funktionen vermisst.
- Keine
- nein
- Nein
- verwendete patterns aufgelistet und verlinkt
- -
- -
- .
- Siehe oben Feedback
- Nein
- "Stop" button, damit ein gewisser stand eingefroren wird und nicht durch änderungen des programmierpartners überschrieben/zerstört wird
- nein
- some comments would be helpful
- Nein

16. Was hat Sie beim Benutzen des Tools am meisten gestört? *

Anzahl Teilnehmer: 22

- Zu Beginn wurde der Style nicht richtig angezeigt, außer in dem Formular selbst.
- Die Popups von Browsersync, ein kleines dezentes grünes/rotes Pünktchen würde reichen.
- teilweise wackeln der Komponenten
- nichts
- server war kurz langsam
- Nichts.
- Option zum Ausblenden der "injected into..." wäre toll
- Man ist evtl.mal verwirrt wer grad welche Anpassung gemacht hat und ob man jetzt selber für einen Fehler verantwortlich ist oder der andere.
- Nichts
- Nichts
- auftreten von merge konflikten -> laesst sich aber durch klare abgrenzung der arbeitsbereiche gut vermeiden
- Keine wirklich großen Störfaktoren gefunden
- iframe initial size
- -
- -
- .
- Eig. nichts
- Dass ich mich mit CSS3 nicht ausreichend beschäftigt habe ;-)
- wenn durch reload ein gewisser stand, bei dem man etwas fixen muss, wieder geändert wird
- einstellbare Fenstergröße mit den Reglern zu Beginn leicht irritierend, mehrfach waren Änderungen von Person A nicht gleich ersichtlich auf Grund von unterschiedlichen Monitorgrößen oder der eingestellten Fenstergröße
- initial configurations
- Fehlende Locks welche Überschreibungsfehler zugelassen hat.

17. Kennen Sie vergleichbare Tools? *

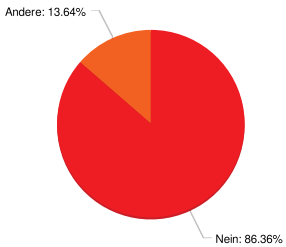
Anzahl Teilnehmer: 22

19 (86.4%): Nein

3 (13.6%): Andere

Antwort(en) aus dem Zusatzfeld:

- stylelab
- stylelab
- elements stylelab, patternlab



18. Bitte geben Sie Ihr Alter an *

Anzahl Teilnehmer: 22

- 25
- 25
- 21
- 22
- 34
- 26
- 24
- 24
- 31
- 26
- 25
- 25
- 27
- 26
- 21
- 25
- 25
- 34
- 31
- 19
- 26
- 25

19. In welchem Tätigkeitsbereich sind Sie tätig? *

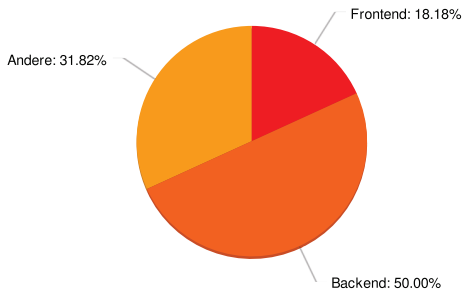
Anzahl Teilnehmer: 22

4 (18.2%): Frontend

11 (50.0%): Backend

- (0.0%): Design

7 (31.8%): Andere



Antwort(en) aus dem Zusatzfeld:

- E-learning
- Full-Stack Webdeveloper
- Frontend & Backend
- Frontend (JSF) + Backend (JavaEE)
- Compiler
- C++ Systementwicklung
- machine learning

Anhang D

Inhalt der CD-ROM

Format: CD-ROM, Single Layer, ISO9660-Format

D.1 PDF-Dateien

Pfad: /

Michael Kuss Thesis.pdf Masterarbeit mit Instruktionen (Gesamtdokument)

Pfad: /Anhang

Testangabe (Anhang A).pdf Testangabe für Probanden
Fragebogen (Anhang B).pdf Fragebogen für TestkandidatInnen
Auswertung (Anhang C).pdf Auswertung von Umfrageonline

D.2 Screen-Design

Pfad: /Test

testbild.eps . . . . . Erstelltes Screen-Design für Testablauf

D.3 Source

Pfad: /Source

Prototype.zip . . . . . Projektdateien und SQL-Dump

D.4 Online Quellen

Pfad: /Online Quellen

ashleynolan survey-2015-1.png Ashley Nolan Survey 2015 Teil 1
ashleynolan survey-2015-2.png Ashley Nolan Survey 2015 Teil 2
ashleynolan survey 2016-1.png Ashley Nolan Survey 2016 Teil 1
ashleynolan survey 2016-2.png Ashley Nolan Survey 2016 Teil 2
smashingmagazine style-guidelines.png Smashing Magazine Style Guides
stevefenton compiling-vs-transpiling.png Steve Fenton Compiling vs Transpiling

Pfad: /Online Quellen/ietf

ietf rfc6455-1.png . . . IETF rfc6455 Teil 1
ietf rfc6455-2.png . . . IETF rfc6455 Teil 2
ietf rfc6455-3.png . . . IETF rfc6455 Teil 3

D.5 Bilder

Pfad: /Bilder

atomic-design.eps . . . Atomic Design Aufbau
autofrontend.eps . . . . Automatische Frontend-Pipeline Ablauf
Frage9.eps . . . . . Auswertung Frage 9
Frage10.eps . . . . . Auswertung Frage 10
Frage11.eps . . . . . Auswertung Frage 11
Frage12.eps . . . . . Auswertung Frage 12
frontendpipeline.eps . . Frontend-Pipeline Ablauf
gulpfile.eps . . . . . Gulp Ablauf
gulpserver.eps . . . . . Gulp Server Aufbau
NodeJsAufbau.eps . . . Node.js Aufbau
nodeserver.eps . . . . . Node Server Aufbau
zendloop.eps . . . . . Zend Loop Ablauf

Pfad: /Bilder/icons

btn.psd . . . . . Button PSD Vorlage
commentbtn.png . . . . Comment Button
enhancebtn.png . . . . Enhance Button
fontbtn.png . . . . . Font Button
qrbtn.png . . . . . QR-Code Button

Pfad: /Bilder/Screenshots

stylelabdetail.png	Prototyp StyleLab Detailansicht
stylelabfull.png	Prototyp StyleLab Gesamtansicht
stylelabov.png	Prototyp StyleLab Overview
wibergdetail.png	Wiberg StyleLab Detailansicht
wiberfull.png	Wiberg StyleLab Gesamtansicht
wibergov.png	Wiberg StyleLab Overview

Quellenverzeichnis

Literatur

[1] Stefan Baumgartner. *Front-End Tooling with Gulp, Bower, and Yeoman*. Manning Publications, 2016 (siehe S. 10, 18, 24).

[2] Wolfgang Ecker, Wolfgang Müller und Rainer Dömer. *Hardware-Dependent Software - Principles and Practice*. Berlin Heidelberg: Springer Science Business Media, 2009 (siehe S. 5).

[3] Brad Frost. *Atomic Design*. ISBN 978-0-9982966-0-9. Massachusetts: Brad Frost, 2016. URL: http://atomicdesign.bradfrost.com/ (siehe S. 3).

[4] Davies Gavin. *Using Build Tools*. Massachusetts: Five Simple Steps, 2015 (siehe S. 12).

[5] Markus Hegner. *Methoden zur Evaluation von Software*. IZ-Arbeitsbericht. Bonn, Germany: IZ - Informations Zentrum Sozialwissenschaften, Mai 2003 (siehe S. 40).

[6] Claudia Hienerth. *Wissenschaftliches Arbeiten kompakt*. Wien: Linde, 2009 (siehe S. 40).

[7] Robert Martin. *Agile Software Development: Principles, Patterns, and Practices*. Upper Saddle River, N.J: Pearson Education, 2002 (siehe S. 4).

[8] Richard E. Mayer. *Multimedia Learning*. Cambridge: Cambridge University Press, 2009 (siehe S. 40).

[9] Huber Stigler und Hanelore Reicher. *Empirische Sozialforschung*. Innsbruck: Studienverlag Ges.m.b.H. Innsbruck, 2012 (siehe S. 40).

[10] Heikki Topi und Allen Tucker. *Computing Handbook*. Boca Raton, FL: CRC Press - Taylor & Francis Group, 2013 (siehe S. 1).

Online-Quellen

[11] Steve Fenton. *COMPILING VS TRANSPILING*. Nov. 2012. URL: https://www.stevefenton.co.uk/2012/11/compiling-vs-transpiling/ (besucht am 03. 08. 2017) (siehe S. 13).

[12] I. Fette u. a. *The WebSocket Protocol*. Dez. 2011. URL: https://tools.ietf.org/html/rfc6455 (besucht am 19. 09. 2017) (siehe S. 18).

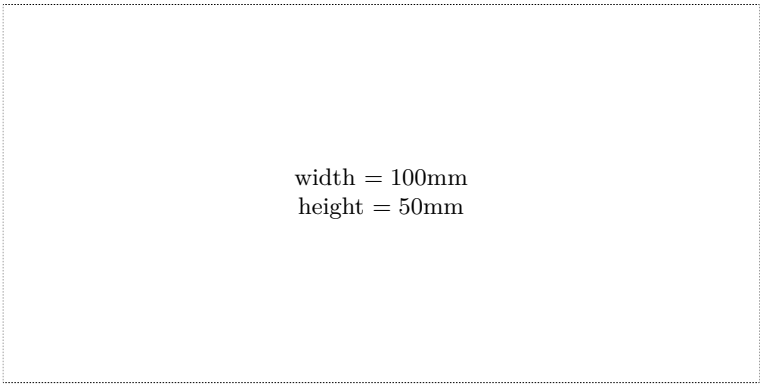
[13] Kat Neville. *How To Design Style Guides For Brands And Websites*. Juni 2010. URL: https://www.smashingmagazine.com/2010/07/designing-style-guidelines-for-brands-and-websites/ (besucht am 18. 05. 2017) (siehe S. 6).

[14] Ashley Nolan. *The State of Front-End Tooling – 2015*. Sep. 2015. URL: https://ashleynolan.co.uk/blog/frontend-tooling-survey-2015-results (besucht am 24. 09. 2017) (siehe S. 13).

[15] Ashley Nolan. *The State of Front-End Tooling – 2016*. Nov. 2016. URL: https://ashleynolan.co.uk/blog/frontend-tooling-survey-2016-results (besucht am 03. 08. 2017) (siehe S. 13).

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —