HOCHSCHULE COBURG

Studie zur Integration von dynamischen Inhalten in CMS-Systeme mit Hilfe von Webframeworks

Studiengang Informationstechnologie für Unternehmensanwendungen

Masterarbeit

vorgelegt von

Julian Kandler

am –.–.201-

Betreut durch

Stefan Forner (EXXETA AG) und

Prof. Dr. Dieter Wißmann (Hochschule Coburg)

Ehrenwörtliche Erklärung

Ich versichere hiermit, dass ich diese Arbeit selbstständig verfasst und noch nicht anderweitig für Prüfungszwecke vorgelegt habe. Es wurden keine anderen als die angegebenen Quellen oder Hilfsmittel verwendet. Wörtliche oder sinngemäße Zitate sind als solche gekennzeichnet.

....

Ort, Datum Unterschrift

Vorwort

Fehlt komplett...

Kurzzusammenfassung

Fehlt komplett...

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Codebeispielverzeichnis

Abkürzungsverzeichnis

AEM Adobe Experience Manager

Ajax Asynchronous JavaScript and XML (Asynchrones JavaScript und XML)

AMC Adobe Marketing Cloud

AMD Asynchronous Module Definiton

AoT Ahead-of-Time (Vorzeitig)

API Application Programming Interface (Programmierschnittstelle)

CMS Content Management System (Inhaltsverwaltungssysteme)

CORS Cross-Origin Resource Sharing

CRX Content Repository Extreme

CSS Cascading Style Sheets (gestufte Gestaltungsbögen)

DI Dependency Injection

DOM Document Object Model (Dokument Objekt Model)

HTL HTML Template Language

HTML HyperText Markup Language (Hypertext-Auszeichnungssprache)

HTTP Hypertext Transfer Protocol (Hypertext-Übertragungsprotokoll)

IoC Inversion of Control

JAR Java Archive

JCR Content Respository for Java Technologie

JiT Just-in-Time (Rechtzeitig)

JRE Java Runtime Environment

JSON JavaScript Object Notation

JSP JavaServer Pages

JSX JavaScript XML

MPA Multi-Page-Application

MVC Model View Controller (Modell-Präsentation-Steuerung)

MVVM Model View ViewModel (Modell-Präsentation Präsentationsmodell)

npm Node Package Manager

REST Representational State Transfer

SCE Strong Contextual Escaping

SOP Same-Origin-Policy

SPA Single Page Application (Einzelseiten-Webanwendungen)

URL Uniform Resource Locator

W3C World Wide Web Consortium

WebDAV Web-based Distributed Authoring and Versioning

XHR XMLHttpRequest

XML Extensible Markup Language (erweiterbare Auszeichnungssprache)

1 Einleitung

Das einführende Kapitel behandelt zuerst die Motivation dieser Arbeit, gefolgt von der Zielsetzung und die daraus entstehenden Aufgaben. Am Ende wird noch kurz ein Ausblick auf den restlichen Aufbau dieser Arbeit gegeben.

1.1 Motivation

Um moderne Webseiten einer Webpräsenz zu erstellen und zu pflegen, arbeiten meist mehrere Personen zusammen, um diese optisch, inhaltlich und funktionell zu gestalten. Dabei hat jeder unterschiedliche, technische Kenntnisse. Um auch Personen, die keinerlei Programmierkenntnisse besitzen, die Möglichkeit zu geben, eine Webpräsenz zu erstellen, wurden diverse Content Management System (Inhaltsverwaltungssysteme, CMS) entwickelt. Ein CMS besitzt zumeist eine grafische Bedienoberfläche, welche auch weniger technikaffinen Menschen das Erstellen von Inhalten auf einer Webpräsenz ermöglicht. Diese Inhalte sind meist statisch. Änderungen, die Aussehen und Inhalt betreffen, müssen manuell erbracht werden.
Viele Unternehmen sind derzeit dabei, ihre Webpräsenz zu reorganisieren. Hierbei fließt neben dem zuvor erwähnten statischen Inhalt oft auch von unterschiedlichen Portalen dynamisch generierter Inhalt mit ein. Dies können z. B. Konfigurationen von Autoherstellern, oder Antragsformulare von Banken sein. Diese dynamischen Inhalte werden häufig von sogenannten Microservices generiert. Microservices sind Architekturmuster, welche aus mehreren, voneinander unabhängigen Prozessen bestehen und kombiniert die benötigten Inhalte erzeugen.
Eine Lösung diese dynamischen, von Microservices generierten Inhalte in Webseiten einzubetten ist es die komplette Webseite neu zu laden und die erstellten Inhalte auf der Serverseite zu integrieren, wie in Unterunterabschnitt 2.3.1.1 beschrieben. Hierfür ließe sich das verwendete CMS nutzen, da diese zumeist auf serverseitigen Techniken und Programmiersprachen basieren. Jedoch müssten bei dieser Lösung auch die Inhalte, welche unverändert sind, vom Server erneut geladen werden, was zu einem unnötigen Mehraufwand führt, da die komplette Webseite, also auch die unveränderten Teile, vom Server

an den Client gesendet werden müssen. Um ein erneutes Laden der Webseite, und somit diesen Mehraufwand, zu vermeiden und das einbetten der dynamisch nachgeladenen Inhalte aus dem Portalumfeld in eine Webseite zu vereinfachen, sollen Lösungen gefunden werden, wie sich Webanwendungen, die mit einem Webframework erstellt wurden, zu integrieren.

1.2 Zielsetzung

Als Ziel dieser Arbeit sollen Erkenntnisse gewonnen werden, wie eine Webanwendung, die mit einem clientseitigen Webframework erstellt wurde, sich am Besten in den Adobe Experience Manager (AEM) (Unterunterabschnitt 2.3.4.1) integrieren lässt, um dynamische Inhalte aus dem Portalumfeld nachladen zu können. Der Fokus dieser Arbeit liegt hierbei auf den Webframeworks AngularJS (Unterabschnitt 3.3.1), AngularJS2 (Unterabschnitt 3.3.2), Aurelia (Unterabschnitt 3.3.3) und React (Unterabschnitt 3.3.4).
Dabei ist zu beachten, dass der AEM-Server unterschiedliche Konfigurationen aufweisen kann. Daher werden mehrere Lösungsansätze für die Integration, angepasst an die jeweilige Konfiguration, benötigt.
Weitere wichtige Aspekte für die Integration wären hier die Anpassung der Webanwendung und des AEM, damit diese von Suchmaschinen korrekt indiziert werden, und das Sicherstellen, dass Hyperlinks zum richtigen Pfad führen.

1.3 Aufgaben

Damit die zuvor genannten Ziele erreicht werden können, wird zunächst der Ist-Zustand untersucht, also wie eine Integration bis jetzt von statten ging. So werden etwaige erste Problematik erkannt.
Anschließend wird der Versuch unternommen jeweils eine Webanwendung der vier zuvor genannten Webframeworks zu integrieren. Dabei ist es zielführend Webanwendungen zu verwenden, die möglichst zahlreiche Funktionalitäten des jeweiligen Webframeworks nutzen. Der Integrationsversuchs wird abschließend mit verschiedenen Konfigurationen wiederholt. Somit lassen sich weitere Schwierigkeiten entdeckten.
Für die gefundenen Probleme gilt es nun Lösungen zu erarbeiten und diese zu dokumentieren. Sollte unter einer Konfiguration die Integration gelungen sein, so wird für diese eine alternative Herangehensweise benötigt.

Schlussendlich werden alle Formen der Integration und Problemlösungen gesammelt, dokumentiert, und weitestgehenden gegenübergestellt.

1.4 Aufbau der Arbeit

Im folgenden Kapitel 2 werden zunächst die Grundlagen dieser Arbeit erklärt. Dazu gehören unter anderem Definitionen, Grundlagen der Webtechnologie und in späteren Kapiteln verwendete Begriffe und Techniken.
Anschließend folgt im Kapitel 3 die Auflistung einiger Anforderungen an die Webframeworks, gefolgt von einer Zusammenfassung der relevanten Webframeworks mitsamt einer kurzen Beschreibung und Gegenüberstellung.
Im Kapitel 4 soll auf den Ist-Zustand eingegangen werden, und warum es nötig ist Lösungen im Zuge dieser Masterthesis zu erarbeiten.
Die erarbeiten Lösungen werden im Kapitel 5 genannt und erklärt. Abschließend wird im Kapitel 6 aus dem gesammelten Wissen dieser Arbeit ein Fazit gezogen und ein Ausblick auf weitere Schritte gegeben.

2 Grundlagen

In diesem Kapitel sollen einige Grundlagen erläutert werden, welche für das Verständnis der Arbeit unabdingbar sind.

2.1 Begriffsdefinitionen

Zunächst soll auf einige Begriffsdefinitionen eingegangen werden.

Begriff	**Definition**
Webbrowser	Ein Webbrowser ist ein Computerprogramm, das Ressourcen einer Webpräsenz, wie z. B. Webseiten und Bilder, aufrufen, darstellen und ggf. interpretieren kann.
Webpräsenz	Eine Webpräsenz ist der Zusammenschluss von Dateien und Ressourcen, zwischen denen sich durch Verwendung eines Webbrowsers navigieren lässt [JG16, S. 30]. Für die Pflege des Inhalts sind zumeist eine oder mehrere Personen verantwortlich.
Webseite	Unter einer Webseite versteht man eine einzige Seite einer Webpräsenz, die in einem Webbrowser dargestellt werden kann [JG16, S. 30]. Zwischen den einzelnen Webseiten kann navigiert werden.
Webanwendung	Eine Webanwendung ist ein Verbund von Webseiten, der, abhängig von gewissen Faktoren, anders dargestellt wird. Siehe hierzu Unterabschnitt 2.3.1. Eine Sonderform hiervon ist eine Single Page Application (Einzelseiten-Webanwendungen, SPA), welche im Unterunterabschnitt 2.3.1.3 erklärt wird.
HTML-Seite	Eine HTML-Seite ist eine Ressource, deren Quellcode aus HTML besteht und die von einem Webbrowser interpretiert und dargestellt wird.

(Web-)Server	Ein Server ist ein Programm auf einem Rechner bzw. ein Zusammenschluss von Rechnern, der verschiedene Dienste und Ressourcen zur Verfügung stellt. Im Folgenden wird ein Server, sofern nicht anders beschrieben, als Synonym für einen Webserver verstanden. Ein Webserver dient in erster Linie dazu, für seine Clients eine Webpräsenz zur Verfügung zu stellen. Siehe auch Unterabschnitt 2.2.5.
(Web-)Client	Ein Client kommuniziert mit einem Server und kann durch ein entsprechendes Protokoll dessen Dienste verwenden und seine Ressourcen abrufen. Ein Webclient ist ein Client, der auf die Ressourcen eines Webservers zugreift und diese in einem Webbrowser darstellt. Im Folgenden wird ein Client, sofern nicht anders beschrieben, als Synonym für einen Webclienten verstanden.
Besucher	Als Besucher versteht man im Folgenden eine Person, die eine Webpräsenz und dessen Webseiten besucht. Als Synonym kann auch der Begriff „Anwender“ Verwendung finden.
Entwickler	Hier ist aus dem Kontext heraus zu lassen, ob ein Entwickler eines Frameworks/einer Bibliothek gemeint ist oder ein Entwickler das besagte Framework/ die besagte Bibliothek nutzt. Hier sind mit Entwicklern meist eine oder mehrere Personen gemeint, die eine Webanwendung für einen Kunden entwickeln.
Kunde	Der Kunde ist der Auftraggeber für eine Webanwendung/eine SPA. Dessen IT-Landschaft unterscheidet sich zumeist von jener, in der ein Entwickler arbeitet.

Tabelle 1: Definitionen

2.2 Technische Grundlagen

Im Folgenden werden weitere Begriffe und grundlegende Techniken erläutert. Hierbei handelt es sich um Grundlagen der Webtechnologie.

2.2.1 HTML

HyperText Markup Language (Hypertext-Auszeichnungssprache, HTML) dient als Sprache, um HTML-Dokumente zu beschreiben. Ein HTML-Dokument ist ein Textdokument, bestehend aus mehreren Tags. Dies sind in spitzen Klammern eingeschlossene Namen.

Sie können durch Attribute weiter beschrieben werden. Ein Tag beschreibt Texte, Überschriften, Bilder, Tabellen etc. Zudem können Tags ineinander verschachtelt werden, um zusammenhängende Elemente logisch zu gruppieren. Ein Webbrowser kann diese Struktur interpretieren und entsprechend darstellen [Lub07]. Die aktuelle Version HTML5 wurde am 28.10.2014 veröffentlicht und bietet einige neue Funktionen [W3C14]. Der typische Aufbau eines HTML-Dokumentes ist hierbei wie folgt.

```
1  <!DOCTYPE HTML>
2  <html>
3    <head>
4      <title>Der Seitentitle</title>
5    </head>
6    <body>
7      <h1>Hello World</h1>
8      <div id="inhalt">Meine erste Webseite!</div>
9      <img src="flowerpot.png" alt="Eine Blume"/>
10   </body>
11 </html>
```

Codebeispiel 1: Beispiel HTML

Ein Webbrowser ist nun in der Lage den HTML-Code zu interpretieren und darzustellen, Codebeispiel 1 würde dabei wie Abbildung 1 im Webbrowser aussehen.

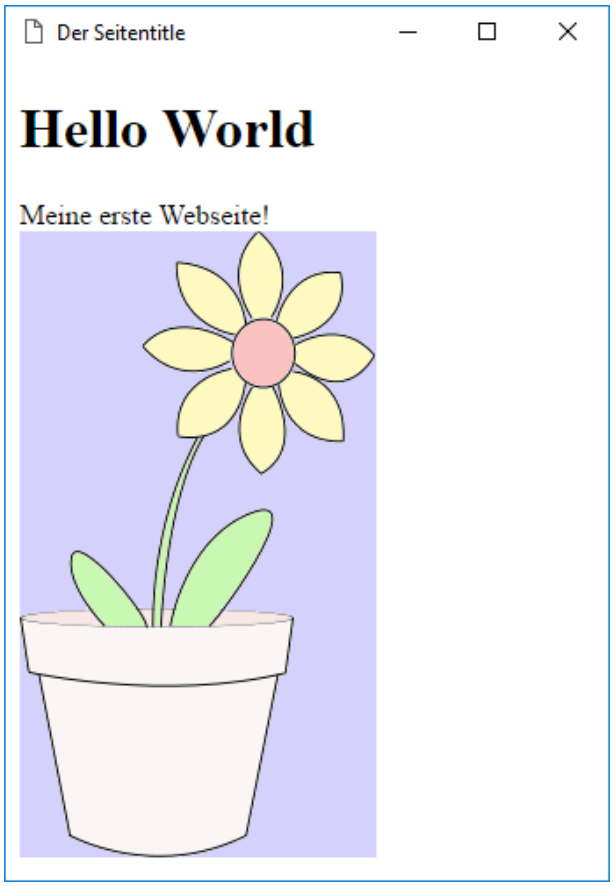

Abbildung 1: Beispiel HTML

2.2.2 CSS

Cascading Style Sheets (gestufte Gestaltungsbögen, CSS) bietet weitere Möglichkeiten zur Gestaltung von HTML-Dokumenten. Deren neueste Version, CSS3, befindet sich momentan in der Entwicklung, wird aber bereits jetzt von vielen Browsern verstanden. Die mit HTML beschriebenen Webseiten werden mit CSS in ihrer Erscheinung gestaltet und Dinge wie Farben und Breiten definiert. Man könnte z. B. das Codebeispiel 1 dahingehend erweitern, dass die Überschrift eine gelbe Hintergrundfarbe erhält, der darunter befindliche Text größer ist und das Bild kleiner dargestellt wird. Dies wird in Codebeispiel 2 gezeigt.

```
1 <!DOCTYPE HTML>
2 <html>
3   <head>
4     <title> Der Seitentitle </title>
5     <style>
6       h1 {background-color:yellow}
7       div {font-size:30pt}
8       img {width:75; height:130 px;}
```

```
9       </style>
10    </head>
11    <body>
12      <h1> Hello World </h1>
13      <div id="inhalt">Meine erste Webseite!</div>
14      <img src="flowerpot.png" alt="Eine Blume" />
15    </body>
16  </html>
```

Codebeispiel 2: Beispiel CSS

Im Webbrowser dargestellt sieht Codebeispiel 2 aus wie in Abbildung 2.

Abbildung 2: HTML erweitert mit CSS

Die Zeilen 6-8 in Codebeispiel 2 heißen CSS-Anweisungen. Der linke Teil ist der Selektor und gibt an, welche Elemente von der Regel betroffen sind. Der rechte Teil innerhalb der geschweiften Klammern ist die Deklaration. Diese besteht wiederum aus Paaren von Eigenschaften und Werten. Eigenschaft und Wert sind durch einen Doppelpunkt getrennt. Die Paare sind durch ein Semikolon getrennt. Mehrere CSS-Anweisungen ergeben zusammen ein Stylesheet.

2.2.3 JavaScript

JavaScript ist eine Programmiersprache, die beim Ausführen, also z. B. beim Aufruf einer Webseite, von einem entsprechenden Programm interpretiert wird, und sie zählt somit zu den Interpreter-Sprachen. Zudem ist es eine schwach typisierte Sprache, das heißt, einer Variablen wird nicht explizit ein Typ zugewiesen, sondern sie erhält diesen implizit durch seinen Wert. JavaScript wurde für den Einsatz innerhalb von Webseiten entworfen, wird heutzutage aber auch auf Servern verwendet. So werden Module für das Webframework Node.Js in JavaScript geschrieben.
Bei JavaScript handelt sich um eine Implementation von ECMAScript, welche von Ecma International unter der Bezeichnung ECMA-262 spezifiziert wird, und sie wurde im Juni 2016 in der mittlerweile siebten Version veröffentlicht. Neuere Versionen bieten Sprachkonstrukte wie Module, Klassen und vieles mehr an. Jedoch unterstützen die Browser oder besser gesagt deren Interpreter bis dato nicht alle neuen Funktionen, wie Tabelle 2 zu entnehmen ist.

Version / Browser	ECMAScript 5	ECMAScript 6	ECMAScript 7
Chrome 51	98 % (Seit CH23+)	98 %	25 %
Firefox 47	100 % (Seit FF21+)	90 %	29 %
Safari 9	96 % (Seit SF6+)	53 %	1 %
Edge 13	99 % (Seit IE10+)	79 %	7 %
Android Browser 5.1	98 % (Seit AN 4.4+)	29 %	4 %
iOS Safari 9	96 % (Seit SF6+)	53 %	1 %

Tabelle 2: Ausschnitt der Browserunterstützung von ECMAScript [ZAP16]

Zu erkennen ist, dass gängige Browser in ihrer aktuellen Version weder Version 6 noch Version 7 zuverlässig unterstützen, wohingegen ECMAScript 5 in den meisten Browsern seit geraumer Zeit großteils lauffähig ist. Soll auch älteren Browsern der Zugriff auf eine Webpräsenz gewährleistet werden, so sollte man auf den Standard ECMAScript 3 aus dem Jahr 1999 zurückgreifen, da alle in Tabelle 2 diese zu mindestens 96 % unterstützen. Zu beachten ist, dass in dieser Arbeit JavaScript als Synonym für ECMAScript steht. JavaScript ist lediglich eine Implementierung von ECMAScript von Mozilla. Jedoch werden aus geschichtlichen Gründen heute beide Begriffe oft als Synonym gehandhabt.

Mit JavaScript ist es möglich, auf Eingaben und Interaktionen des Anwenders zu reagieren und das Erscheinungsbild der Webseite dynamisch, ohne dass diese komplett neu geladen werden muss, zu verändern. Dies erfolgt durch Manipulation des Document Object Model (Dokument Objekt Model, DOM), wobei es sich um eine Schnittstelle für den Zugriff

auf ein HTML-Dokument handelt. Über die Schnittstelle lassen sich Knoten in einer Baumstruktur ablegen. Die durch das Codebeispiel 1 generierte HTML-Seite würde wie folgt aussehen.

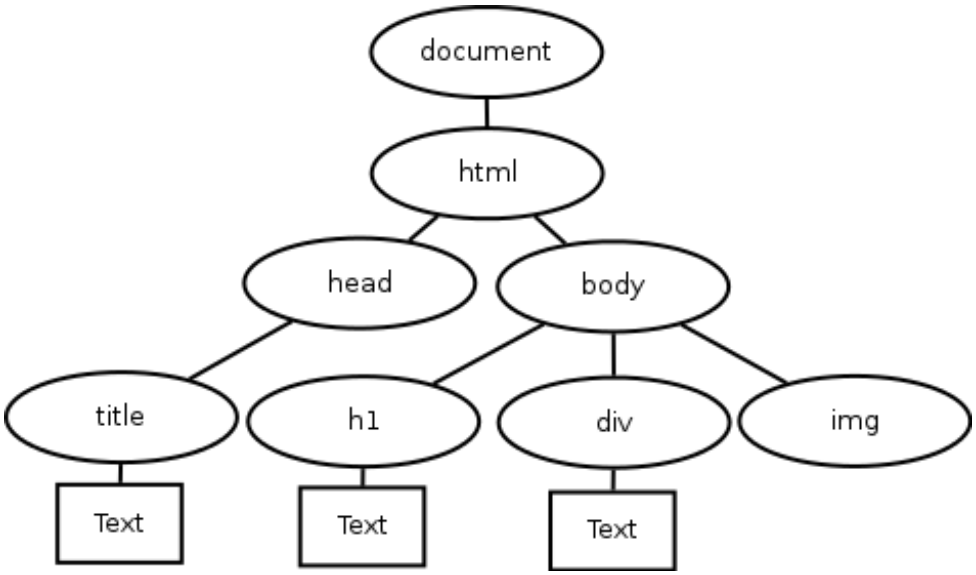

Abbildung 3: Vereinfachte Baumstruktur des DOM ohne Attribute

Der JavaScript-Code befindet sich innerhalb des HTML-Dokumentes in einem Script-Element, oder er wird als externe JavaScript-Datei in das HTML-Dokument eingebettet. Im folgenden Codebeispiel soll gezeigt werden, wie man ein Dokument mittels DOM-Manipulation beeinflussen kann.

```
1 <script>
2   document.getElementById("inhalt").innerHTML="Ein neuer, Text!";
3 </script>
```

Codebeispiel 3: Ein JavaScript Beispiel

Es wird im Dokument aus Abbildung 1 das Element mit dem id -Attribut und dem Wert „inhalt“ gesucht und dessen Inhalt ersetzt. Der Text „Meine erste Webseite!“ lautet nun: „Ein neuer Text!“. Zur DOM-Manipulation gehört auch die Möglichkeit, Elemente zu löschen, neue einzufügen und deren Attribute zu bearbeiten.

2.2.4 TypeScript

Bei TypeScript handelt es sich um eine von Microsoft entwickelte Programmiersprache. Diese lehnt sich an ECMAScript Version 6 an. TypeScript erweitert deren Funktionsumfang aber um neue Elemente.

Jedoch unterstützten die Browser aus Tabelle 2 kein TypeScript und können dieses nicht ausführen. Hierfür wird eine Anwendung benötigt, welche TypeScript in ECMAScript übersetzt. Solch eine Anwendung, genannt Transpiler, stellt unter anderem Microsoft zur Verfügung und übersetzt TypeScript wahlweise in ECMAScript 5 bzw. ECMAScript 3 [SS15, S. 439 f.]. Ein Transpiler ist ein Programm, das eine Programmiersprache in eine andere übersetzt. So ist einem Entwickler die Möglichkeit gegeben die Vorteile von ECMAScript 6, wie Klassen und Vererbung, zu nutzen. Zudem ist es in TypeScript, im Gegensatz zu JavaScript, möglich, Variablen einem Typ zuzuweisen, womit diese auch stark typisiert genutzt werden können.

2.2.5 Webserver

Die Entwicklung einer Webpräsenz ist prinzipiell auch ohne einen Webserver denkbar. Wird dieser lokal im Browser geöffnet, würde dies aber bei einem Ajax Aufruf, siehe hierzu Unterabschnitt 2.2.8, mit bei der Standardeinstellung gewisser Browser zu einem Same Origin Policy-Fehler führen [SS15, S. 45 f.].
Abhilfe schafft hier der Einsatz eines Webservers. Als Plattform stehen neben dem Apache HTTP Server, nginx oder dem Microsoft Webserver ein auf Node.js basierender Webserver zur Auswahl. Dieser wird in Unterabschnitt 3.1.1 näher erläutert.
Zudem wird ein Webserver für den Produktiveinsatz benötigt, um Besuchern den Zugriff mit einem Webbrowser zu erlauben.

2.2.6 Servlet

Bei einem Java-Servlet, kurz auch Servlet, handelt es sich um einen Java-Code, der auf einem Webserver ausgeführt wird [Wis12, S. 93]. Diese nehmen Anfragen von Clienten entgegen und beantworten diese mit dynamisch generierten Nachrichten. Diese Nachrichten können zum Beispiel HTML-Seiten sein, welche zum Erstellen von serverseitigen Webanwendungen, siehe Unterunterabschnitt 2.3.1.1, genutzt werden können.

2.2.7 HTTP

Beim Hypertext Transfer Protocol (Hypertext-Übertragungsprotokoll, HTTP) handelt es sich um ein zustandsloses Protokoll zur Datenübertragung, dessen Haupteinsatzgebiet im Internet ist. Über HTTP werden Ressourcen von einem Server zu einem Webbrowser übertragen. Hierbei kommunizieren der Webserver und der Webbrowser über das Protokoll.

Der Ablauf erfolgt in etwa wie in Abbildung 4.

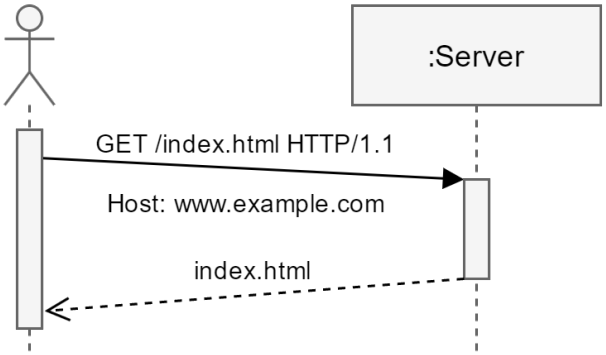

Abbildung 4: Kommunikationsbeispiel mit HTTP

Der Client fordert vom Server, dem der Hostname www.example.com zugewiesen ist, die gewünschte Ressource, hier index.html, an. Zusätzlich wird die gewünschte HTTP-Methode, hier GET, und die HTTP-Version, hier 1.1, definiert. Weiterhin ist das Versenden weiterer Parameter möglich. Diese Anfrage wird auch HTTP-Request genannt. Anschließend antwortet der Server im Erfolgsfall mit der angeforderten Ressource und weiteren Metainformation, im Fehlerfall mit einem entsprechenden Statuscode. Die Antwort des Servers nennt sich HTTP-Response.
Im Normalfall besteht eine Webseite aber nicht nur aus einer HTML-Seite, sondern noch aus weiteren Ressourcen. Diese werden, zumindest in der HTML-Version 1.1, in etwa wie in Abbildung 5 übertragen. Zu beachten ist, dass diese und folgenden Abbildungen im Gegensatz zur Abbildung 4 vereinfacht wurden und bewusst auf Informationen wie die HTTP-Methode und die HTTP-Version verzichtet wird.

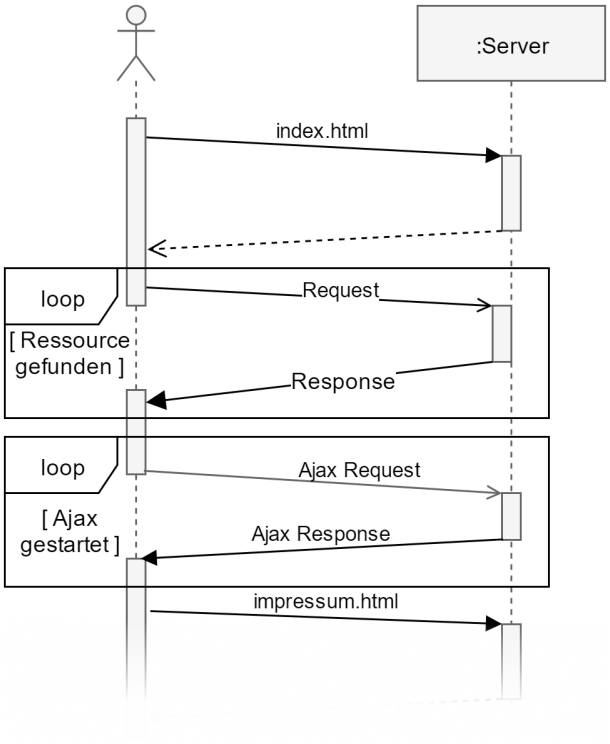

Abbildung 5: Ladevorgang einer Webseite

Zunächst wird die eigentliche Webseite, am Beispiel Abbildung 5 wäre dies index.html, angefordert. Nachdem diese vollständig geladen ist, durchsucht der Webbrowser den erhaltenen Quellcode nach weiteren Ressourcen, wie Grafiken, Skripte, Stylesheets etc. und fordert auch diese vom Webserver an. Dabei können mehrere Anfragen gleichzeitig ausgeführt werden. Der Webbrowser achtet aber darauf, dass unter anderem Skripte in der gleichen Reihenfolge ausgeführt werden, wie diese im Quellcode der Webseite auftauchen, auch wenn ein später auftauchendes kleines Skript früher vollständig geladen wurde als eine zuvor erscheiende Bibliothek, um Abhängigkeitsfehler zu vermeiden. Weitere Ressourcen kann der Client nun über Ajax anfordern, siehe hierzu Unterabschnitt 2.2.8. Möchte der Benutzer nun die Seite wechseln, zum Beispiel nach impressum.html, würde sich der Prozess entsprechend, gegebenenfalls mit anderen Ressourcen, wiederholen.

Einfach ausgedrückt, wird eine Webseite also wie folgt geladen.

HTML-Seite: Die darzustellende HTML-Seite wird vom Server angefordert.

Ressourcen: Der Quellcode der HTML-Seite wird nach weiteren Ressourcen, wie Grafiken, Skripte oder Stylesheets, durchsucht und diese werden geladen.

Ajax: Ajax Anfragen werden ausgeführt.

2.2.8 Ajax

Durch die Verwendung von JavaScript lässt sich das Konzept von Asynchronous JavaScript and XML (Asynchrones JavaScript und XML, Ajax) bewerkstelligen. Unter Zuhilfenahme dieses Konzeptes lassen sich neue Inhalte von einem Server nachladen, ohne die bestehende Seite neu zu laden. Dafür wird eine HTTP-Anfrage an den Webserver gestellt, welcher mit den entsprechenden Daten antwortet. Diese erhaltenen Daten können anschließend durch DOM-Manipulation in die Seite eingebettet werden. Dies senkt sowohl die Ladezeit als auch das zu ladende Volumen an Daten.
Besagte Daten werden häufig, wie man vom Namen schon ableiten kann, im XML-Format übertragen. Andere Formate, wie z. B. JSON, sind jedoch auch möglich.

2.2.9 XML und JSON

Bei der Extensible Markup Language (erweiterbare Auszeichnungssprache, XML) handelt es sich um eine Notation, um Daten hierarchisch und strukturiert in einer Textdatei und in einem für den Menschen lesbaren Format darzustellen [Seb10, S. 34]. Als Beispiel soll eine kurze Datenbank für Musikstücke im Codebeispiel 4 dienen. Diese besteht aus vier Musikstücken mit jeweils einer ID, einem Titel und einem Preis.

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <songs>
3    <song>
4      <id>1</id>
5      <title>Lets Go</title>
6      <price>0.99</price>
7    </song>
8    <song>
9      <id>2</id>
10     <title>Song about Pencils</title>
11     <price>2.3</price>
12   </song>
13   <song>
14     <id>3</id>
15     <title>Samba samba</title>
16     <price>0.5</price>
17   </song>
18   <song>
19     <id>4</id>
20     <title>Another Song</title>
21     <price>3.33</price>
22   </song>
```

```
23 </songs>
```

Codebeispiel 4: Ein XML-Beispiel

Ein anderes, ebenfalls verwendetes Übertragungsformat bei Ajax ist JavaScript Object Notation (JSON). Diese ist im Gegensatz zu XML kompakter und lässt sich direkt in JavaScript umwandeln [Rie09, S. 658]. So könnte die Musikdatenbank hier wie in Codebeispiel 5 aussehen.

```
1 [
2   { "id": 1, "title": "Lets Go", "price": 0.99 },
3   { "id": 2, "title": "Song about Pencils", "price": 2.3 },
4   { "id": 3, "title": "Samba samba", "price": 0.5 },
5   { "id": 4, "title": "Another Song", "price": 3.33 }
6 ]
```

Codebeispiel 5: JSON für eine einfache Musikdatenbank

Zu beachten ist, dass Codebeispiel 5 keine 1:1-Umsetzung von Codebeispiel 4 darstellt, sondern einen etwas anders hierarchischen Aufbau verwendet. Zudem geht die Information, dass es sich um Songs handelt, verloren. Die damit darstellbaren Daten sind jedoch identisch.

2.3 Fachliche Grundlagen

Die zuvor beschriebenen Begriffsdefinitionen und technischen Grundlagen dienten dem Verständnis des folgenden Kapitels.
Nun sollen auch einige Grundlagen erklärt werden, um das Thema dieser Arbeit besser verstehen zu können.

2.3.1 Webanwendung

Frühere Webpräsenzen waren meist recht simpel gestaltet. Traditionell waren dies oft einfache Plattformen, um statische Informationen über das Internet zu verteilen. Einzig bei der Navigation durch die einzelnen Webseiten erfolgte eine Interaktion mit dem Besucher. Der Client schickt an den Webserver eine Anfrage, welche Webseite er gerne hätte, und erhält als Antwort das HTML-Dokument, siehe Abbildung 6.

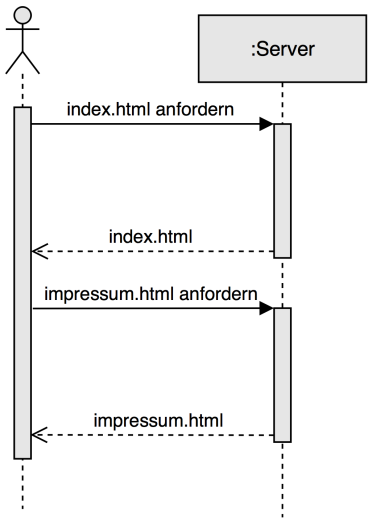

Abbildung 6: Systematischer Ablauf beim Abruf einer Webseite

Erst durch den Onlinehandel, webbasierte Buchungssysteme und Online-Auktionssysteme entstanden die ersten Webanwendungen als Weiterentwicklung von Webpräsenzen. Hierbei interagiert der Besucher neben der klassischen Navigation noch auf anderen Wegen mit der Webanwendung. So kann dieser zum Beispiel bei einem Onlinehändler für Musik durch die einzelnen Webseiten navigieren, durch entsprechende Anfragen das Musikangebot nach seinen Interessen filtern und Musikstücke erwerben [BGP00, S. 89].
Ein wesentlicher Unterschied zu Softwareanwendungen ist hier, dass eine Webanwendung mithilfe von Techniken und Standards, unter anderem vorgeschlagen vom World Wide Web Consortium (W3C), entwickelt wird und so Ressourcen über einen Webbrowser bereitstellt [KRR03, S. 2].

2.3.1.1 Serverseitige Webanwendungen

Für die Entwicklung von Webanwendungen wurden verschiedene Lösungsansätze entworfen. Diese lassen sich in erster Linie in serverseitige und clientseitige Lösungen einteilen. Bei serverseitigen Lösungen wird die Seite, abhängig von gewissen Parametern, vom Server zur Laufzeit generiert und an den Clienten gesendet. Als Beispiel soll ein Chatportal dienen. Dieses besteht aus einem Bereich für die Nachrichten, jeweils einem Feld für den Benutzernamen und die zu versendende Nachricht und einen Knopf zum Verschicken der Nachricht. Zur besseren Verdeutlichung dient Abbildung 7.

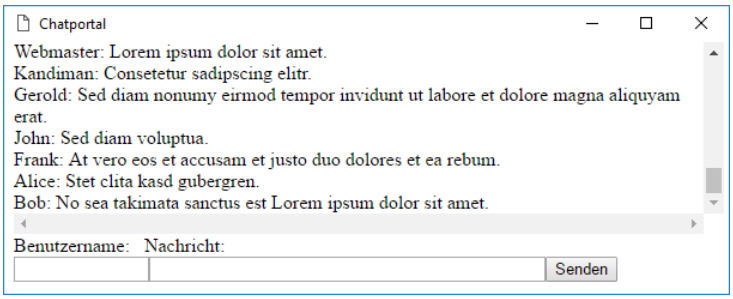

Abbildung 7: Ein einfaches Beispiel eines Onlinechats

Die serverseitige Lösung könnte vom Ablauf her in etwa wie in Abbildung 8 aussehen.

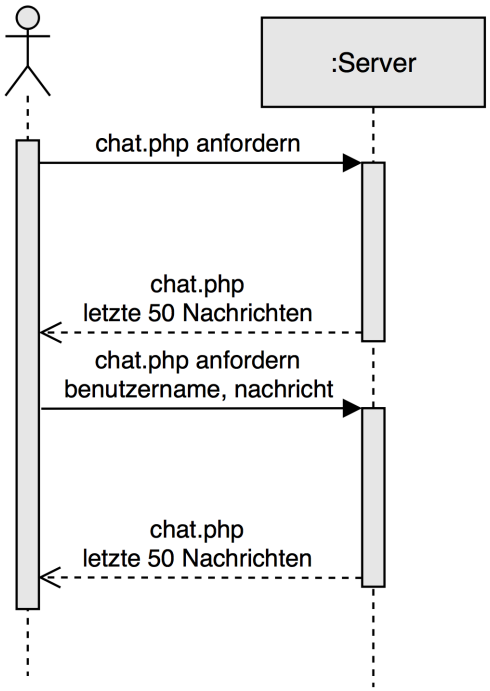

Abbildung 8: Kommunikation von Server und Client beim Laden neuer Informationen bei serverseitigen Webanwendungen

Bei der ersten Anfrage erhält der Benutzer die Webseite und die letzten 50 Nachrichten. Er kann die beiden Felder für den Benutzernamen und für die Nachricht befühlen und durch einen Klick auf die Schaltfläche „senden“ beide Inhalte an den Server schicken. Dieser speichert die neue Nachricht samt dazugehörigen Namen und antwortet dem Clienten mit einer HTML-Seite, die alle neuen Nachrichten, seine eigene eingeschlossen, umfasst. Jede neue Nachricht führt hierbei zu einem kompletten Neuaufbau der Seite und einem erneuten Versenden der alten Nachrichten vom Server an den Clienten. Da auch die alten Nachrichten erneut mit versendet werden, hat dies einen erhöhten Datentransfer zur Folge.

2.3.1.2 Clientseitige Webanwendungen

Um die Menge der zu versendenden Daten zu dezimieren, lassen sich clientseitige Lösungen zum Erstellen von Webanwendungen einsetzen. Diese bauen zumeist auf den Einsatz von JavaScript und Ajax. Die Nachrichten werden via Ajax an einen Service versandt und neue Nachrichten, ebenfalls mit Ajax, vom Service abgefragt. Neue und eigene Nachrichten werden durch DOM-Manipulation integriert, siehe Abbildung 9.

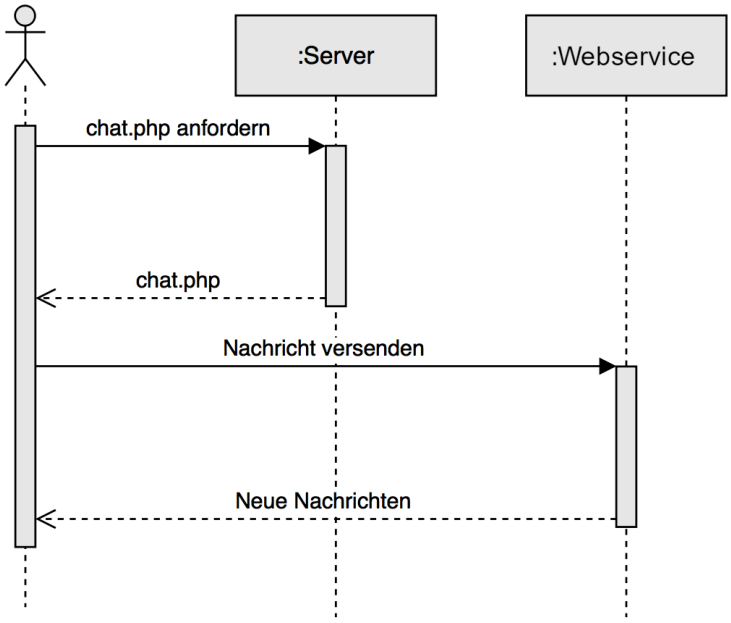

Abbildung 9: Kommunikation von Server und Client beim Laden neuer Informationen bei clientseitigen Webanwendungen

Ein Webservice ist hier ein Teil des Webservers, über den eine Maschine-zu-Maschine-Interaktion, typischerweise über HTTP, ermöglicht wird. Der Webservice besitzt eine vom Entwickler definierte Schnittstelle, über die der Austausch von Nachrichten ermöglicht wird.
Sowohl auf der Server- als auch auf der Clientseite wird der Datenverkehr verringert. Von Nachteil ist jedoch, dass im Browser auch JavaScript aktiviert sein muss. Ist dies nicht der Fall, so würde die Webanwendung nicht funktionieren.

Beide Ansätze lassen sich auch kombinieren. So könnte das Chatportal bei seiner ersten Anfrage die alten Nachrichten abrufen. Alle neuen Nachrichten werden anschließend über Ajax nachgeladen.

2.3.1.3 Single-Page-Webanwendung

Durch den Einsatz von Ajax und DOM lässt sich auch eine Single Page Application (Einzelseiten-Webanwendungen, SPA) entwickeln. Diese werden im Deutschen auch Single-Page-Webanwendung genannt. Hierbei besteht eine Webpräsenz lediglich aus einer HTML-Seite, man täuscht dem Besucher aber bei der Navigation mehrere Webseiten vor [SS15, S. 32]. Dies wird erreicht, indem der Inhalt der neu anzuzeigenden Webseite durch Ajax geladen und durch DOM-Manipulation in die Seite integriert wird. Der alte Inhalt wird ausgeblendet und ist später, ohne dass eine weitere Ajax-Anfrage nötig ist, wieder eingeblendet. Durch Einsatz dieser Technik müssen auf den einzelnen Webseiten wiederkehrende Elemente, wie zum Beispiel Navigation, Kopf- und Fußzeilen etc. nur einmal zu Beginn geladen werden. Die Inhalte der bereits besuchten Seiten liegen weiterhin ausgeblendet beim Clienten vor und können relativ zügig durch DOM-Manipulation wieder eingeblendet werden. Analog zum Begriff Single Page Application werden klassische Webanwendungen, bei denen jeder Webseite eine HTML-Seite zugewiesen wird, auch Multi-Page-Application (MPA) genannt. Zu beachten ist, dass der Begriff MPA in der Literatur nicht üblich ist und nur vereinzelt Verwendung findet.

2.3.2 Framework / Bibliothek

Eine Bibliothek ist eine Codesammlung, mir der sich wiederkehrende Codestrukturen abbilden lassen, um einen kürzeren und stabileren Quellcode zu verfassen. Ein Framework liefert zudem noch ein Programmiergerüst, welches vorgefertigte Funktionalitäten mitbringt. Dieses Gerüst liefert den Rahmen (Frame), innerhalb dessen sich Anwendungen erstellen lassen. Ein Framework legt im Gegensatz zu einer Bibliothek auch eine Steuerung von Verhaltensweisen bei der Verwendung fest [Ste11, S. 20]. Eine Art solcher Frameworks sind Webframeworks. Mit diesen lassen sich Webanwendungen entwickeln. In Rahmen dieser Arbeit sind z. B. die Webframeworks AngularJS, AngularJS2, React und Aurelia, siehe Abschnitt 3.3, relevant.

2.3.3 MV*-Architekturmuster

Häufig unterstützen Webframeworks, aber auch diverse Content Management-Systeme, verschiedene Architekturmuster, um die Entwicklung von Webanwendungen modularer zu gestalten. Der Gedanke dabei ist es, den Quellcode zum Beispiel in Logik und Präsentation aufzuteilen. So ist der Code für Entwickler leichter zu warten, die Testbarkeit verbessert sich und es ergeben sich wiederverwendbare Codestücke [SS15, S. 34].

Eines dieser Entwurfsmuster ist das Model View Controller (Modell-Präsentation-Steuerung, MVC). Dieses besteht aus drei Komponenten. Wie diese interagieren, beschreibt Abbildung 10.

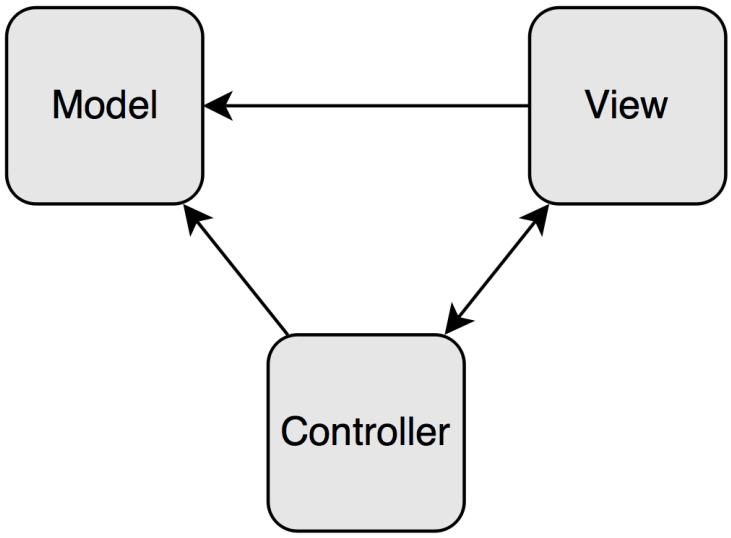

Abbildung 10: MVC-Muster

Die Model-Komponente beinhaltet die darzustellenden Daten. In der View-Komponente wird beschrieben, wie die Daten darzustellen sind. Die Controller-Komponente regiert auf Benutzeraktionen und beeinflusst die Darstellung der View und die Daten des Models. Zudem können beim MVC komplexere Views in mehrere einzelne Views aufgeteilt und ineinander verschaltet werden [GHJV09, S. 5 ff.].

Als weiteres Entwurfsmuster ist das Model View ViewModel (Modell-Präsentation Präsentationsmodell, MVVM) zu nennen. Model und View entsprechen dem des MVC-Musters. Anstelle eines Controllers wird hier aber ein View-Model eingesetzt. Hier wird die Logik der Bedienoberfläche (View) beschrieben und referenziert zu den Daten (Model) [Gar11, S. 40], vergleiche Abbildung 11. Im Gegensatz zum Controller im MVC-Muster besitzt das View-Model hier keinerlei Informationen über die View. Dies erleichtert den Austausch der View und erlaubt, dass sich mehrere Views auf ein View-Model beziehen.

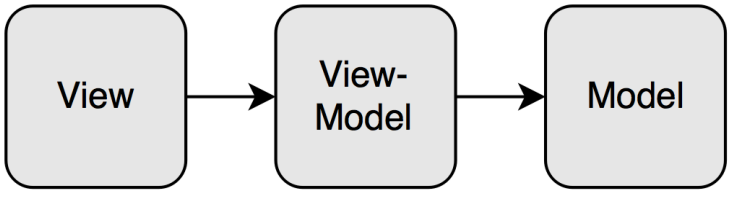

Abbildung 11: MVVM-Muster

Bei MV*-Architekturmustern wird die View häufig durch ein oder mehrere Templates (Vorlagen) beschrieben. Im Zusammenhang mit MV* ist ein Template eine Vorlage für

das Aussehen einer Webseite, in dem wiederkehrende Elemente definiert werden. Diese werden für gewöhnlich in separaten Dateien gespeichert und mit Platzhaltern und einer einfachen Logik versehen. Anhand der Daten des Models und ggf. weiterer Variablen werden die Platzhalter gefüllt und so ein HTML-Dokument erstellt, das dem Clienten zugeschickt wird. Die semantische Darstellung der Platzhalter und des Logikflusses ist von der verwendeten Templatesprache abhängig [New08, S. 37].

2.3.4 Content Management System

Unternehmen verwenden Webpräsenzen, um sich online zu präsentieren. Dabei liegt der Fokus des Managements darauf, Informationen schnell und aktuell dem Kunden zur Verfügung zu stellen. Um dies zu erreichen, werden Content Management System (Inhaltsverwaltungssysteme, CMS) verwendet. Diese erleichtern die Trennung von Inhalt (z.B. Texte, Bilder, Videos etc.), Layout und Struktur [Spo09, S.1 ff.]. Somit muss sich ein Redakteur nicht mit dem Aussehen der Webpräsenz beschäftigen und kann sich nur auf den Inhalt fokussieren.
Ein CMS bietet zumeist eine Vielzahl an Funktionalitäten. Dazu gehören das Erstellen und Strukturieren von Inhalten, eine Mehrbenutzerfähigkeit und administrative Funktionen zum Verwalten besagter Nutzer [Spo09, S. 55]. Die Verwendung erfolgt über eine grafische Bedienoberfläche, um den Umgang mit dem System auch technisch weniger affinen Menschen zu vereinfachen. Diese ist generell über einen Webbrowser aufzurufen und zu verwenden. Somit benötigt ein Autor keine Zusatzsoftware und kann ortsungebunden arbeiten [BS11, S. 19 f.].

2.3.4.1 Adobe Experience Manager

Adobe Experience Manager (AEM) ist ein ursprünglich von Day Software und dem Namen CQ entwickeltes CMS, das im Jahr 2010 von Adobe aufgekauft und nun unter neuem Namen von selbigem weiterentwickelt wird. Dieses ist in der Programmiersprache Java entwickelt, welche sich durch ihre Plattformunabhängigkeit auszeichnet. AEM gehört zu der Adobe Marketing Cloud (AMC) und ist mit den anderen darin enthaltenen Produkten kompatibel. Zu diesen gehören unter anderem Werkzeuge für die Analyse, das Erstellen von Zielgruppenprofilen und die Optimierung von Medieninhalten [Inc15, S. 1-2].
Adobe empfiehlt generell die Verwendung von drei laufenden Instanzen, jeweils auf voneinander unabhängigen Servern. Zunächst wäre hier die „Author“-Instanz zu nennen. In dieser können Autoren Inhalte erstellen, bearbeiten und administrieren. Sobald diese bereit für die Besucher sind, werden sie an die „Publisher“-Instanz weitergegeben und veröffent-

licht. Um die Performance zu verbessern, empfiehlt es sich, noch eine „Dispatcher“-Instanz zu verwenden. Diese läuft auf einem Webserver und speichert die vom Publisher generierten Webseiten zwischen. Vom Benutzer generierte Inhalte, wie zum Beispiel Kommentare, werden vom Dispatcher entgegengenommen und an den Publisher weitergeleitet [Inc15, S. 1-3 f.].

2.3.4.1.1 JCR

AEM verwendet das Application Programming Interface (Programmierschnittstelle, API) des Content Respository for Java Technologie (JCR). Diese API wird in der aktuellen Version 2.0 vom Java Community Process spezifiziert und unter der Bezeichnung JSR-283 veröffentlicht. Sie dient dazu, über standardisierte Schnittstellen auf Inhalte zuzugreifen. Der Austausch von Inhalten zwischen einzelnen Implementierungen von JCR ist somit möglich.
Bei JCR sind Inhalte in Form von Knoten und Eigenschaften abgelegt. An der Spitze gibt es einen Wurzelknoten und jeder Knoten kann beliebig viele Kindknoten besitzen, womit eine Baumstruktur entsteht. Zudem kann jeder Knoten eine beliebige Anzahl an Eigenschaften besitzen, welche aus Name-Wert-Paaren bestehen. Jede Eigenschaft erhält einen Typ, der angibt, was für den Wert eingetragen werden darf, beispielsweise eine Zeichenkette (String) oder eine Zahl (Integer). An Eigenschaften können keine Kinder angehängt werden. Als Beispiel soll Abbildung 3 dienen.

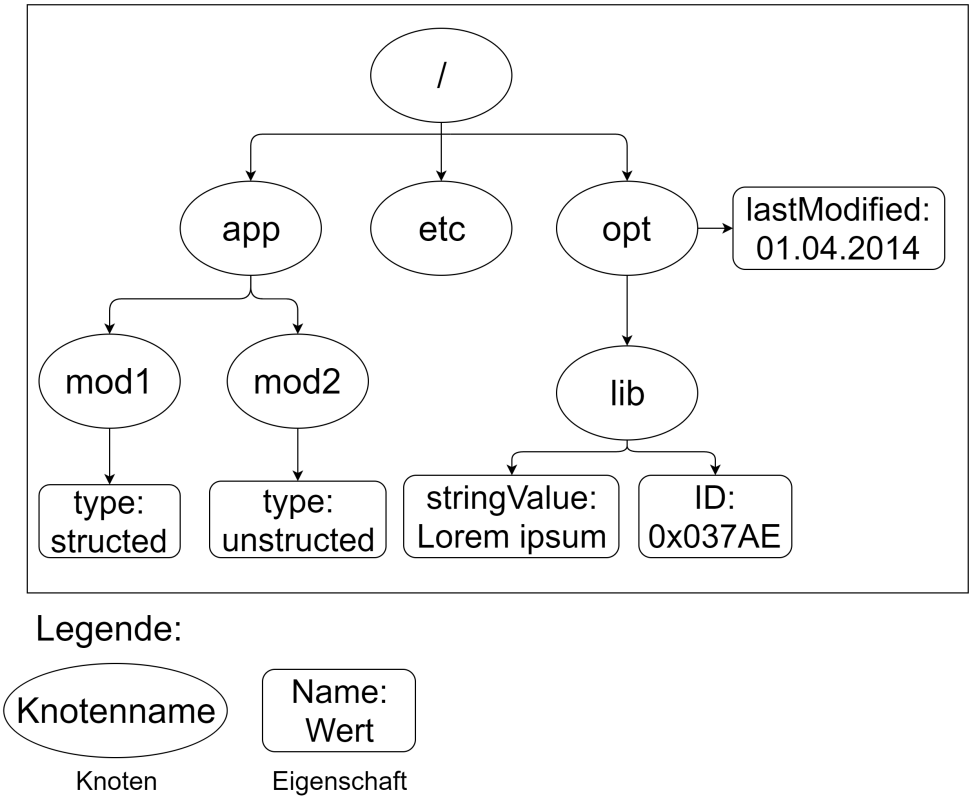

Abbildung 12: Baumstruktur des JCR

Für diese Arbeit gilt, dass sich der Baum von Abbildung 12 auch textuell wie folgt darstellen lässt.

```
1  /
2    + app/
3      + mod1/
4        - type (Name) = structed    # Ein Kommentar
5      + mod2
6      - type (Name) = unstructed
7    + etc/
8    + opt/
9      - lastModified (Date) = 01.04.2014
10     + lib/
11       - stringValue (String)= Lorem Ipsum
12       - ID = (String) 0x037AE
```

Codebeispiel 6: Textuelle Darstellung von JCR

Somit entspricht ein Plus (+) einem Knoten, ein Minus (-) einer Eigenschaft und eine Einrückung einer tieferen Ebene in der Baumstruktur. Der Inhalt der Klammern bei den Eigenschaften gibt dessen Typ an. Alles hinter einer Raute (#) dient als Kommentar für die textuelle Darstellung und ist im JCR nicht hinterlegt.

2.3.4.1.2 CRX

Alle Inhalte werden in das Content Repository Extreme (CRX) hinterlegt. Dieses basiert auf Apache Jackrabbit, welches als Referenzmodell für eine Implementierung von JCR dient [Ado16b].

2.3.4.1.3 OSGi und Apache Felix

AEM verwendet das OSGi-Framework Apache Felix. Ein solches Framework ist ein System, welches die Modularisierung von Anwendungen in Komponenten, so genannten Bundles, und deren Verwaltung ermöglicht. Bei einem Bundle handelt es sich um eine JAR-Datei, welche mit einer Versionsnummer, einem symbolischen Namen und weiteren Metainformationen versehen ist. In besagtem Java Archive (JAR) befinden sich der kompilierte Java-Code, Skripte und Ressourcen der Komponente [Inc15, S. 2-6 f.].

2.3.4.1.4 Apache Sling

Ein weiterer Bestandteil von AEM ist Apache Sling. Dieses Framework zum Erstellen von serverseitigen Webanwendungen benötigt eine JCR-Implementierung wie Apache Jackrabbit oder, wie im Fall von AEM, CRX. Anhand des angeforderten Pfades bei einem HTTP-Request wird entschieden, welches Skript bzw. Servlet ausgeführt werden soll und welche Daten aus dem JCR benötigt werden. Ziel von Apache Sling ist es, Inhalte aus dem JCR über ein HTTP-Request nach dem Representational State Transfer (REST)-Programmierparadigma bereitzustellen. Für REST müssen folgende Eigenschaften erfüllt werden.

- Jede Ressource ist über einen eindeutigen Uniform Resource Locator (URL) zu erreichen.
- Als Ressource können verschiedene Methoden angewandt werden. Bei einem HTTP-Request wären dies z. B. POST, GET DELETE oder PUT. Zu beachten ist, dass REST jedoch kein HTTP voraussetzt.
- Die Darstellung von Ressourcen kann in verschiedenen Formaten erfolgen.
- REST ist immer zustandslos [SS15, S. 82 f.].

Abbildung 13 beschreibt, wie Apache Sling einen Aufruf auflöst.

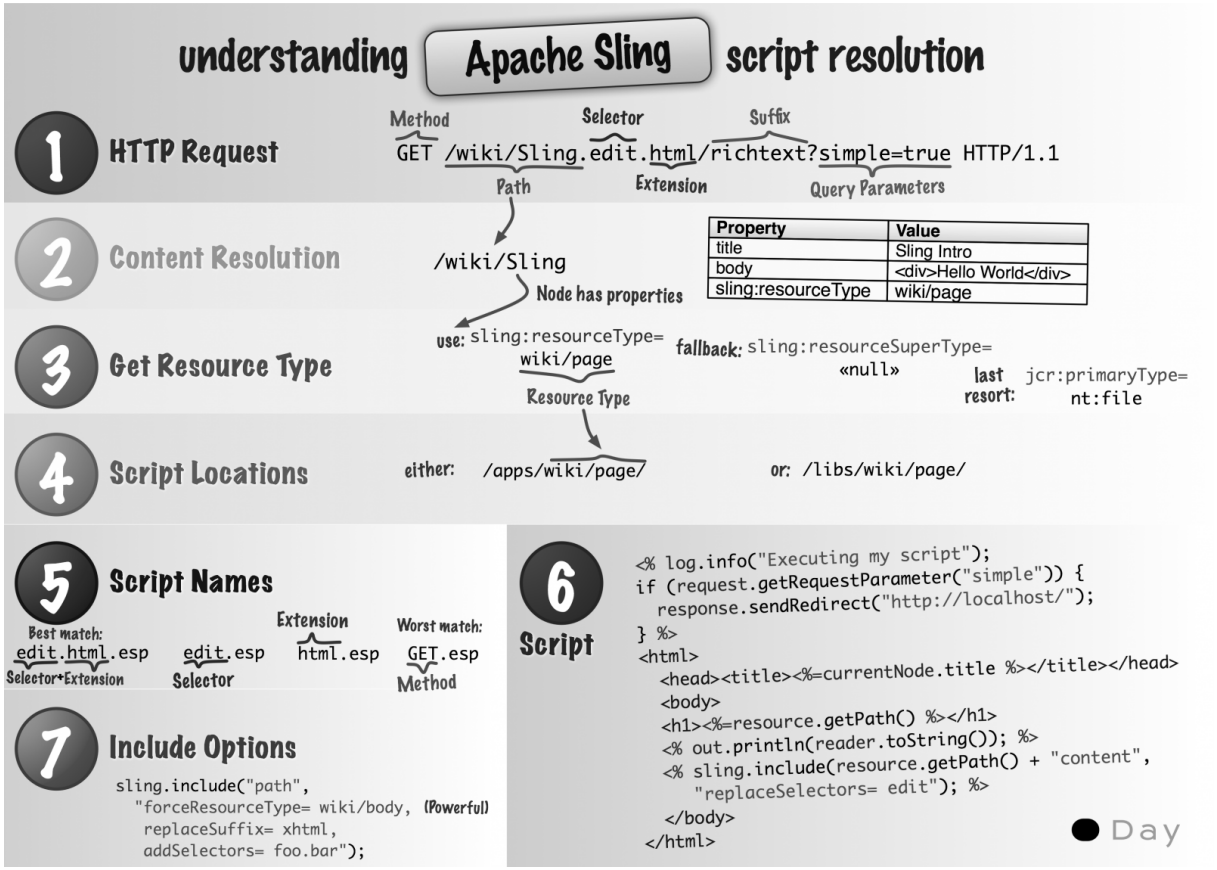

Abbildung 13: Auflösung einer Anfrage bei unter Sling, von [Ado16d]

Wird z. B. die Seite über einen Webbrowser *http://localhost:4502/wiki/Sling.edit.html/richtext?simple=true* aufgerufen, so wird die URL aufgeteilt. Alles ab „/wiki...“ wird von Sling aufgelöst, somit wird im Folgenden nur der Abschnitt ab „/wiki...“ betrachtet.

1. Alles bis zum ersten Punkt ist der Pfad, gefolgt von einem oder mehreren Selektoren und zuletzt der Erweiterung. Anschließend können noch Suffixe und Parameter folgen, die noch weitere Informationen mitgeben.

2. Die Eigenschaften des Knotens, die sich aus dem vorherigen Schritt erschlossen haben, werden abgefragt und nach einer mit dem Namen. „sling:resourceType“ gesucht. Diese hat in diesem Beispiel den Wert „wiki/page“.

3. Der Wert wird für den nächsten Schritt verwendet.

4. Nun wird zunächst unter /app/wiki/page nach einem geeigneten Skript gesucht. Wird dieses nicht gefunden, wird die Suche unter /libs/wiki/page fortgesetzt.

5. Der Name des gesuchten Skripts wird aus dem Selektor und der Erweiterung zusammengesetzt. Sollte kein entsprechendes Skript gefunden werden, wird zuletzt noch als Name die Methode ausprobiert.

6. Wird ein entsprechendes Skript gefunden, kann dieses gerendert werden, und es können z. B. die Parameter, die mit der URL mitgegeben wurden, einfügt werden.
7. ???

2.3.4.1.5 Autoren-Bedienoberfläche

AEM ermöglicht es Autoren, ihre Webinhalte über zwei Bedienoberflächen zu gestalten. Dies wäre zum einen die klassische Oberfläche, welche für Desktop-PCs ausgelegt ist, und zum anderen die Touch-optimierte für Tablets und Smartphones. Jeder Autor kann sich eine Oberfläche nach Belieben aussuchen und ggf. nachträglich zur jeweils anderen wechseln.
Da die Touch-optimierte Bedienoberfläche nach der Installation von AEM als Standard eingestellt ist und auch auf dem Desktop-PC lauffähig ist, werden sich die folgenden Kapitel, sofern nicht anders erwähnt, mit dieser Oberfläche befassen.
Die Autoren-Bedienoberfläche bietet zahlreiche Werkzeuge, um Webseiten nach Belieben zu gestalten. Dazu gehört der Bearbeitungsmodus für Seiten, zu sehen in Abbildung 14.

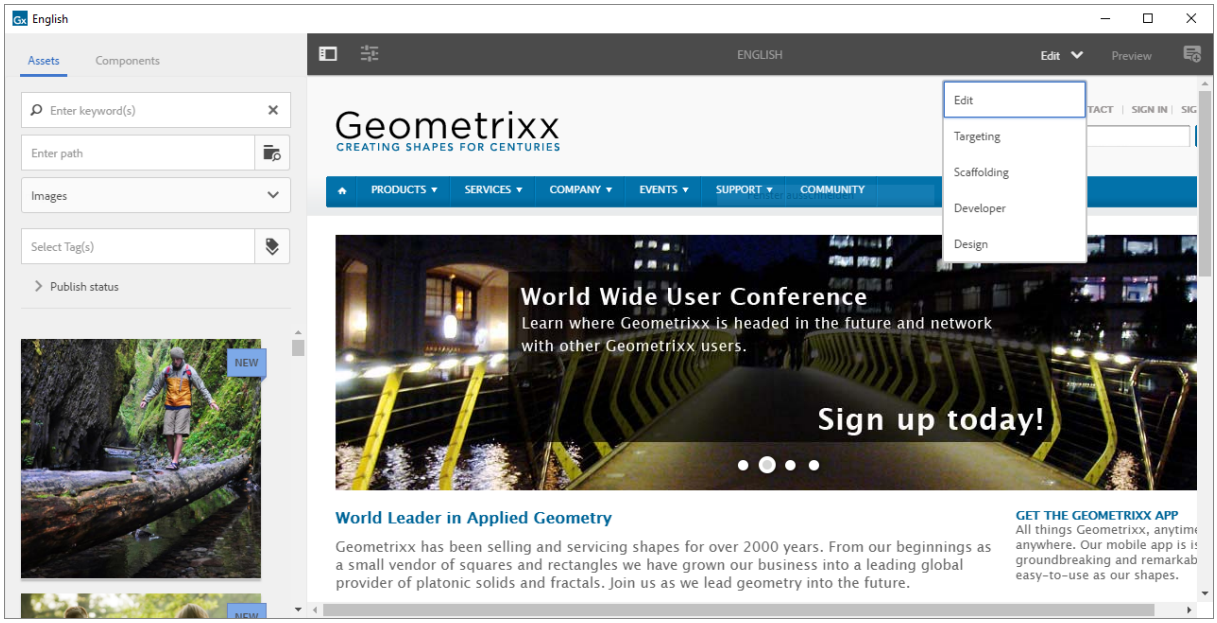

Abbildung 14: Autoren-Bedienoberfläche

Das linke Menü lässt sich nach Bedarf auf- und zuklappen. Unterschreitet das Browserfenster eine gewisse Breite, nimmt das Menü dessen gesamte Fläche ein. Hier können Inhalte (Bilder, Dokumente etc.) und Komponenten durch Ziehen und Loslassen der Seite hinzugefügt werden. In der oberen rechten Ecke wird zwischen verschiedenen Modi gewechselt. Diese sind:

Edit: Inhalte, Texte und Komponenten können eingefügt, entfernt und bearbeitet werden.

Targeting: Verwendung des Zusatzproduktes Adobe Target, Teil von AMC.

Scaffolding: Zum Erstellen von mehreren Seiten mit ähnlichem Aufbau.

Developer: Für Entwickler, um Komponenten zu debuggen.

Design: Hier wird das Design der Seite angepasst.

2.3.4.1.6 Administrator Bedienoberfläche

AEM bietet einige Werkzeuge an, mit deren Hilfe das System administriert und erweitert werden kann. Zum Beispiel wäre hier eine Bedienoberfläche für die OSGi-Verwaltung und für administrative Eingriffe in das CRX zu nennen.
Entwickler müssen häufig Eigenschaften und Knoten des CRX editieren. Eine Möglichkeit, dies zu bewerkstelligen, wäre mit dem Tool CRXDE Lite, welches ebenfalls über den Webbrowser erreichbar ist, und es wird bei AEM per Standard mit installiert.
Abbildung 15 zeigt einen Beispielknoten innerhalb einer lauffähigen AEM-Webanwendung, welche für Lernzwecke mit dem AEM installiert werden kann.

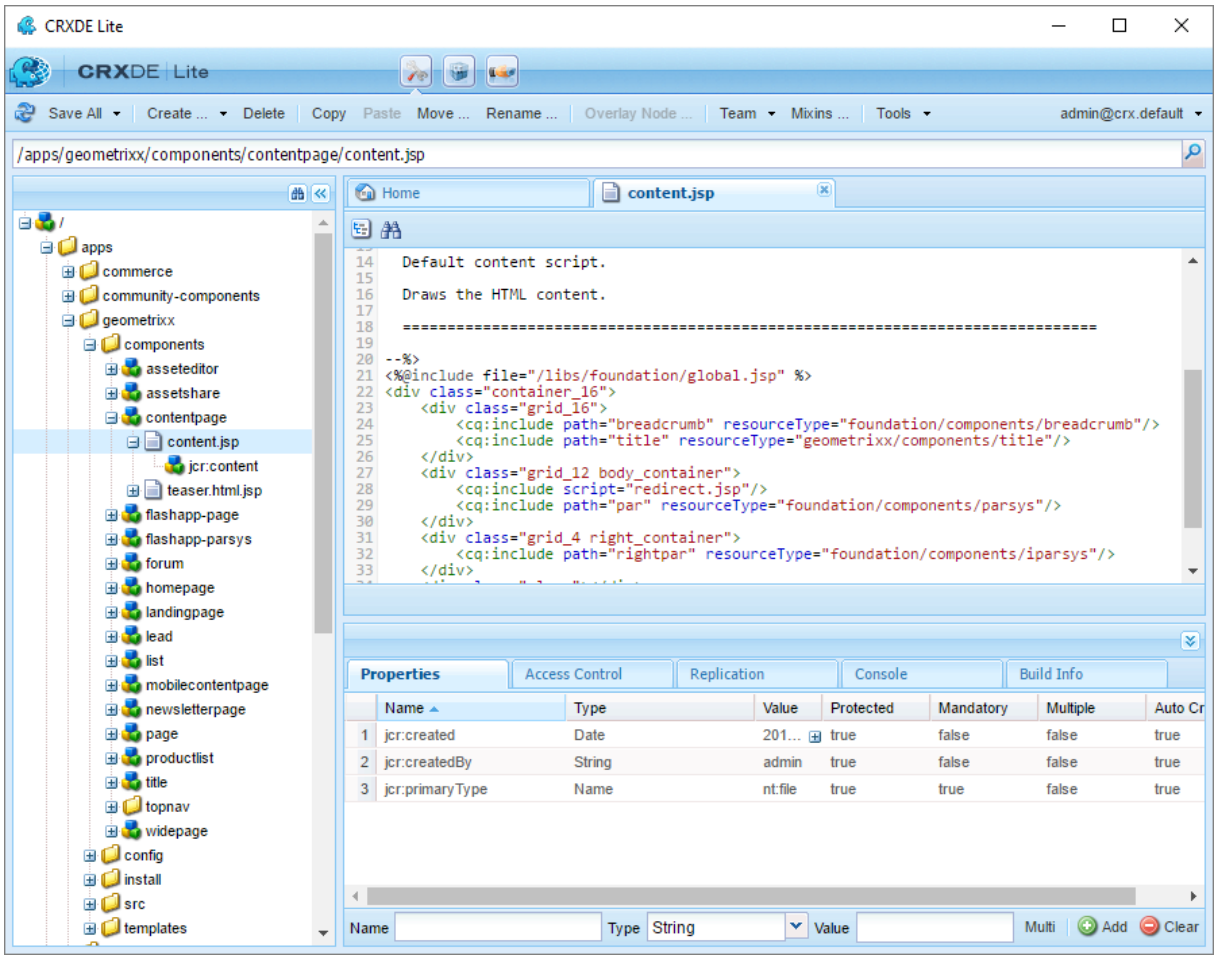

Abbildung 15: CRXDE Lite

Wie sich der Abbildung entnehmen lässt, wird die Baumstruktur auf der linken Seite angezeigt. Unten rechts sind die Eigenschaften des gewählten Knotens zu sehen und oben rechts lassen sich Templates und Java-Dateien bearbeiten.

2.3.4.1.7 AEM-Anwendung

In AEM basieren Webpräsenzen auf AEM-Anwendungen. Dies sind Strukturen im JCR und sie befinden sich in der Regel unter `/app` [Inc15, S. 6.4].
Die übliche Struktur einer Anwendung mit dem Namen helloworld ist in Codebeispiel 7 zu sehen.

```
1 /apps/helloworld
2   + components
3   + templates
```

Codebeispiel 7: Exemplarische Darstellung einer AEM Anwendung

Unter /apps/components werden für die Anwendung spezifische Komponenten, siehe Absatz 2.3.4.1.9, abgelegt. Unter Templates befinden sich JSP- und HTL- Vorlagen für die einzelnen Seiten.

2.3.4.1.8 HTL

Auch AEM verwendet Templates für seine Seiten. Als Model dienen Daten, welche über die JCR-API gewonnen werden. Als Template-Sprache kann man zwischen jener von Java-Server Pages (JSP) oder auch HTML Template Language (HTL) wählen. Letztere wurde von Adobe entwickelt und in AEM mit der Version 6.0 ausgeliefert und wird von Adobe als bevorzugte Templatesprache genannt [Ado16c].
HTL ist an HTML angelehnt, somit ist auch jedes in HTL verfasste Template zugleich gültiges HTML5 [Inc15, S. 5.11 f.]. Mit dem folgenden Codebeispiel 8 würden vom aktuellen JCR-Knoten alle Kinder in einer HTML-Liste dargestellt und jeder Listeneintrag abwechselnd mit der HTML-Klasse „odd“ bzw. „even“ versehen werden.

```
1 <ul data-sly-list.child="${currentPage.listChildren}">
2   <li class="${ childList.odd ? 'odd' : 'even'}">${child.title}</li>
3 </ul>
```

Codebeispiel 8: Ein HTL Beispiel

2.3.4.1.9 Komponenten

Komponenten in AEM sind wiederverwendbare modulare Zusammenschlüsse von spezifischen Funktionalitäten, die zur Darstellung von Inhalten auf einer Webpräsenz dienen [Ado16a].
Diese werden über die Autoren-Bedienoberfläche an ihren bevorzugten Ort innerhalb einer Webseite platziert, konfiguriert und ggf. wieder gelöscht.
AEM liefert bereits einige vorgefertigte Komponenten mit, wie eine Text-Komponente zum Bearbeiten von Text, eine Bild-Komponente, mit der Bilder platziert werden können, und einige mehr. Diese sind in JSP oder HTL geschrieben worden und wurden für die Touch-optimierte, klassische oder für beide Bedienoberflächen entworfen. Hier beschriebene Komponenten sind immer in HTL geschrieben und für die Touch-optimierte Bedienoberfläche ausgelegt. Die Logik der Komponenten kann in Java, JavaScript und mit Clientlibs erfolgen.

2.3.4.1.10 Clientlib

JavaScript und CSS können durch sogenannte Client Libraries oder kurz auch Clientlib ausgeliefert werden. Dies sind einfache Knoten mit einer bestimmten Struktur und Eigenschaften, welche sich innerhalb einer Komponente oder unter `/etc/cliebtlibs` befinden. Codebeispiel 9 zeigt exemplarisch die benötigte Struktur und ihre Eigenschaften.

```
1  /etc/clientlibs/angularapp
2    - jcr:primaryType (Name) = cq:ClientLibraryFolder  # Indiziert Knoten
       ↪ als Clientlib
3    - categories (String[]) = [angularApplication]     # Ein Array an
       ↪ Kategorien, an dem die Clientlib idendifiziert wird.
4    - dependencies (String[]) = [angularjs]            # Mögliche Abhä
       ↪ ngigkeiten zu anderen Clientlibs
5    - embedded (String[]) = [app1, app2]               # Andere Clientlibs
       ↪  können hier eingebettet werden
6    + js.txt                                           # Auflistung aller
       ↪ JavaScript Dateien
7      - jcr:primaryType = nt:file
8    + css.txt                                          # Auflistung aller
       ↪ CSS Dateien
9      - jcr:primaryType = nt:file
10   + scripts                                          # Ordner für
       ↪ Scripte
11     - jcr:primaryType (Name) = nt:folder
12   + styles                                           # Ordner für CSS
13     - jcr:primaryType (Name) = nt:folder
```

Codebeispiel 9: Exemplarische Darstellung einer Clientlib

Die Ordnernamen für CSS und JavaScript sind hierbei frei wählbar und können auch Unterordner beinhalten. Auch eine Platzierung direkt unterhalb des clientlib-Knotens ist möglich.
Innerhalb eines Templates wird nun die Clientlib mithilfe eines entsprechenden Tags verwendet. Dieser würde bei HTL wie in Codebeispiel 9 aussehen.

```
1  <sly data-sly-use.clientlib="/libs/granite/sightly/templates/
     ↪ clientlib.html" data-sly-unwrap/>
2  <sly data-sly-call="${clientlib.all @ categories='angularApplication'}"
     ↪ data-sly-unwrap/>
```

Codebeispiel 10: Verwendung einer Clientlib in AEM

Zeile 1 wird für den Einsatz von Clientlibs benötigt. Zeile 2 gibt an, welche Clientlib verwendet werden soll. Durch clientlib.all werden alle Ressourcen geladen, clientlib.js und

clientlib.css würde nur das Laden der JavaScript- bzw. CSS-Dateien bewerkstelligen. Der Wert hinter categories gibt ein Array an Kategorien an. Nun werden alle Clientlibs samt ihren Abhängigkeiten, welche diesen Kategorien entsprechen, geladen.

3 Webframeworks

Um die Komplexität einer SPA zu bewältigen, haben sich in den vergangenen Jahren Teams von Entwicklern mit dem Ziel zusammen getan, Webframeworks zu entwickeln, um auf deren Grundlage die Entwicklung künftiger SPAs zu vereinfachen.
Folgend soll zunächst auf Werkzeuge und Techniken eingegangen werden, deren Einsatz in allen Webframeworks möglich ist. Anschließend folgt eine Auflistung von Eigenschaften, die ein Webframework mitbringen sollte, damit sich eine Neuentwicklung einer SPA hiermit lohnt. Anschließend wird eine Auswahl an Webframeworks genannt und näher erläutert.

3.1 Werkzeuge und Techniken

Die Entwicklung eines SPA wird durch den richtigen Einsatz entsprechender Werkzeuge vereinfacht und automatisierte Vorgänge werden zum gegebenen Zeitpunkt ausgeführt. Folgend werden einige Werkzeuge samt entsprechenden Beispielprogrammen gelistet.

3.1.1 Node.js

Node.js ermöglicht es serverseitige Webanwendungen in JavaScript zu verfassen [Fed15, S. 4]. Auch bietet Node.js eine Erweiterung an, welche TypeScript automatisiert in JavaScript umwandelt. Dies beschleunigt die Entwicklung von in TypeScript verfassten Webanwendungen, da eine Umwandlung bei Änderung des Quellcodes nicht manuell angestoßen werden muss.

3.1.2 Paketverwaltung

Für Node.js gibt es ein breites Spektrum an Erweiterungen. Zur Verwaltung dieser Erweiterungen existiert der Node Package Manager (npm), welches derzeit über 330.000

Pakete umfasst [DeB16]. Unter anderem wird so die einfache Installation von Werkzeugen für die Entwicklung oder auch das Laden von JavaScript-Bibliotheken und Frameworks ermöglicht. Alternativ kann über npm auch die Installation anderer Paketverwaltungs-Programme wie Bower [Twi16] erfolgen.

3.1.3 Automatisierungswerkzeuge

Oft muss eine Webanwendung, sobald diese in ihre produktive Umgebung überführt wird, einigen Anpassungen unterliegen. Hierzu zählen Dinge wie die Minimierung von Quellcode, Anpassen von Pfaden oder Konkatenieren von Dateien. Damit dies nicht der Entwickler händisch übernehmen muss und sich dabei mögliche Flüchtigkeitsfehler einschleichen, existieren speziell für JavaScript als Anwendungen entworfene Werkzeuge für die Automatisierung. Als populäre Beispiele hierfür wären Grunt [Alm16] und Gulp [Fra16] zu nennen [SS15, S. 246 f.].

3.1.4 Modularisierung und Verwaltung von Abhängigkeiten

Nicht selten besitzt ein Codestück Abhängigkeiten zu anderen Programmcodes, damit dieses korrekt funktioniert. Dies sind zum Beispiel Bibliotheken, Frameworks oder Codestücke von anderen Entwicklern. Zu gewährleisten, dass alle diese Abhängigkeiten korrekt aufgelöst werden, mag bei kleinen Projekten noch ohne Weiteres vom Entwickler allein zu bewältigen sein. Hierfür muss er alle Skripte in der korrekten Reihenfolge in das HTML-Dokument einbinden. Sobald ein Projekt aber wächst und mehrere Entwickler daran beteiligt sind, steigt auch die Komplexität und somit auch die Herausforderung, besagte Abhängigkeiten aufzulösen.
Lösungen hierfür sind Entwurfsmuster wie Dependency Injection (DI), Inversion of Control (IoC) [GD13, S. 25 ff.] oder das Entwickeln gegen Schnittstellen, wie der Asynchronous Module Definiton (AMD) [SS15, S. 266 ff.]. Das Ziel ist dabei, den Code zu modularisieren, die Entkopplung von Abhängigkeiten und eine komponentenorientierte Entwicklung [SS15, S. 261]. Für die Zusammenführung der einzelnen Module haben sich gewisse Bibliotheken etabliert. Eine Zusammenfassung der derzeit bedeutendsten Bibliotheken und für welche Art von Modulen diese geeignet sind, ist Tabelle 3 zu entnehmen.

Bibliothek	Geeignet für
browserify	npm-Module
RequireJS	ES 2015 Module, AMD
SystemJS	ES 2015 Module, AMD, CommonJS
Webpack	CommonJS, AMD

Tabelle 3: Möglichkeiten für Dependency Injection

In jedem Modul wird deklariert, welche anderen Modulabhängigkeiten dieses besitzt. Die Syntax jener Deklaration ist von dem genutzten Entwurfsmuster bzw. der genutzten Schnittstelle abhängig. Die verwendete Bibliothek löst die Abhängigkeiten auf und übergibt den Modulen Instanzen ihrer benötigten Module.

3.2 Anforderungen

Bei der Integration der Frameworks gilt es, gewisse Faktoren zu berücksichtigen. Dies sind Anforderungen, die ein Framework erfüllen muss, um einen langfristigen und zufriedenstellenden Einsatz gewährleisten zu können. Zudem müssen bei der Integration der final entwickelten Webanwendung einige Punkte Beachtung finden.
Die Vorgaben sind zumeist von extern, also von Kunden, und werden im Folgenden nicht immer erläutert, warum diese so und nicht anders ausgefallen sind.

3.2.1 Browserunterstützung

Da die Webanwendungen bei möglichst vielen Besuchern lauffähig sein sollen, ist die Browserunterstützung hier ein wichtiger Aspekt.
Die Mindestversion der Browser ist Tabelle 4 zu entnehmen.

Browser	Mindestversion	Veröffentlicht am
Chrome	23	25.09.12
Firefox	16	28.08.12
Safari	6	25.07.12
Internet Explorer / Edge	10	04.09.12
Android Browser	4.4	09.12.13
iOS Safari	6.1	28.01.13

Tabelle 4: Browsersupport der Webanwendungen

3.2.2 Integration

Die Autoren-Bedienoberfläche von AEM inkludiert bereits eine Reihe von JavaScript und CSS-Dateien. Diese sind vonnöten, um verschiedene Funktionalitäten, wie zum Beispiel die Konfiguration der Komponenten, zu gewährleisten. Es ist somit darauf zu achten, dass Skripte, die vom Framework, von der Webanwendung oder von der Integration der Webanwendung in das AEM stammen, nicht mit Skripten seitens AEM kollidieren.

3.2.3 Indizierung von Suchmaschinen

Das Internet ist eines der sich am schnellsten entwickelnden Medienlandschaften. Viele Unternehmen unterschiedlichster Größenordnung sind hier vertreten, manch eine Firma präsentiert sich sogar nur online [KRM15, S. 29 f.]. Da viele Benutzer täglich eine Suchmaschine verwenden oder eine als Startseite ihres Browsers festgelegt haben, sollte es für ein Unternehmen von hoher Wichtigkeit sein, dass seine Webpräsenz mit den richtigen Begriffen bei einer Suche möglichst weit oben erscheint [KRM15, S. 147].

3.2.3.1 Funktionsweise einer Suchmaschine

Damit eine Webpräsenz mitsamt ihren Webseiten überhaupt in einem Suchergebnis erscheint, werden diese von der jeweiligen Suchmaschine zunächst indiziert. Dies übernehmen Programme, sogenannte Crawler oder auch Robots. Diese durchsuchen das Internet und erstellen einen Katalog aller gefundenen Webseiten. Weitere Webseiten werden über interne und externe Hyperlinks, die sich in den zuvor gefundenen Webseiten befinden, erreicht. Externe Hyperlinks sind eine Möglichkeit, weitere Webpräsenzen zu finden. Auch diese werden wieder indiziert und auf weitere Hyperlinks durchsucht. Solch eine Navigationsstruktur kann wie in Abbildung 16 aussehen.

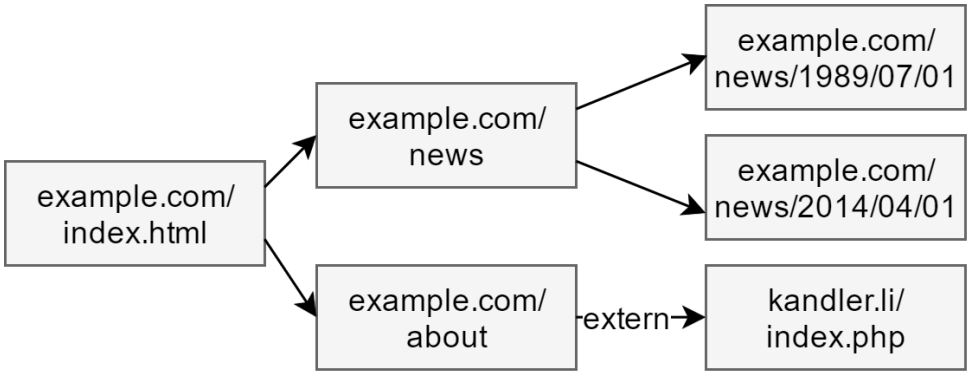

Abbildung 16: Mögliche Navigationsstruktur einer Webseite

Zu sehen ist, dass alle Webseiten von der Einstiegsseite (*index.html*) aus zu erreichen sind. Über diese wird auf die Nachrichtenübersicht (*/news*) und das Impressum (*/abbout*) verwiesen. Die beiden Nachrichteneinträge (*/news/1989/07/01* und */news/2014/04/01*) sind über die Nachrichtenübersicht erreichbar. Im Impressum ist ein externer Hyperlink (*kandler.li/index.php*) vorzufinden.

Anhand von Schlagwörtern und Algorithmen werden die Webseiten analysiert und in den Katalog der Suchmaschine aufgenommen. Besagter Algorithmus entscheidet, welche Webseite durch welche Suchbegriffe in der Reihenfolge der Suchergebnisse erscheint. Dieser Algorithmus ist in der Regel äußerst komplex und umfasst am Beispiel Google über 200 zu berücksichtigende Faktoren [KRM15, S.163 f.].

Nun sollte eine Suchmaschine in der Lage sein, eine Webpräsenz samt all ihrer Webseiten zu indizieren. Die Suche nach „www.example.com Impressum“ sollte hier im Idealfall als erstes Ergebnis das Impressum von *example.com* liefern.

3.2.3.2 Probleme bei Single-Page-Webanwendungen

Nun ist eine SPA so gestaltet, dass diese nur aus einer einzigen HTML-Seite besteht und ein Nachladen von Inhalten meist durch JavaScript und Ajax erfolgt. Wegen dieser Eigenschaften resultieren einige Probleme in der Indizierung.

3.2.3.2.1 Nachladen mit JavaScript

Suchmaschinen analysierten früher zur Indizierung lediglich den Quellcode einer Webseite. Alle JavaScript-Anweisungen wurden ignoriert und nicht wie in einem Webbrowser interpretiert, womit eine SPA für die Suchmaschine oft ohne bedeutenden Inhalt erschien. Im Jahr 2009 gab Google bekannt, nun auch JavaScript und Ajax beim Crawling zu berücksichtigen [Goo09].

3.2.3.2.2 Hyperlinks innerhalb der Single-Page-Webanwendung

Problematisch ist aber noch die Navigation zwischen den einzelnen Webseiten. Bei einer traditionellen MPA geschieht dies durch Hyperlinks. Über z. B. *http://www.example.com* wäre die Startseite der Webpräsenz zu erreichen, über *http://www.example.com/news* alle Neuigkeiten und hinter *http://www.example.com/about* könnte sich das Impressum verstecken, wie zum Beispiel in Abbildung 16 zu sehen. Jede Unterseite würde von einer

Suchmaschine korrekt indiziert werden. Ein Hyperlink innerhalb einer SPA wurde oft mit einer Sprungmarke [Con16], zu erkennen an der Raute (#), versehen. Die Neuigkeiten wären nun über *http://www.example.com/index.html#news* und das Impressum über *http://www.example.com/index.html#about* zu erreichen. Die Navigationsstruktur aus Abbildung 16 würde nun wie in Abbildung 17 aussehen.

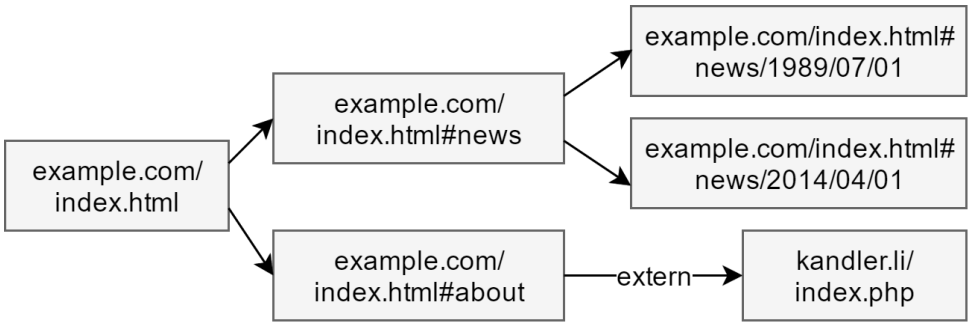

Abbildung 17: Mögliche Navigationsstreukur in einer SPA

Alle Seiten einer SPA wurden von gängigen Suchmaschinen nur als eine Seite, also *http://www.example.com/index.html*, interpretiert. Bis Oktober 2015 empfahl Google, um dieses Problem zu umgehen, die Raute um ein Ausrufezeichen (!) zu ergänzen, womit sich ein Hyperlink der Form *http://www.example.com/#!about* ergab. Der Crawler ersetzt nun die Zeichenkombination „#!“, welche auch Shebang genannt wird, durch „?_escaped_fragment_=“, was zu Adressen wie zum Beispiel *http://www.example.com/?_escaped_fragment_=about* führt. Der Crawler erwartet nun vom Server unter dieser Adresse eine reine HTML-Seite ohne JavaScript und mit den gleichen Inhalten, wie ein Besucher dies auch beim Aufruf von *http://www.example.com/#!about* nach der Ausführung aller JavaScript-Anweisungen erhalten hätte.

3.2.3.2.3 Lösung durch die pushState-Funktion

Die Indizierung durch Einsatz von Shebang funktioniert zwar noch, wird aber seit Oktober 2015 von Google offiziell nicht länger empfohlen. Anstelle dieser sollen Techniken und Methoden der progressiven Verbesserung Verwendungen finden, um Webpräsenzen auch zukünftig zuverlässig indizieren zu können. Hierbei kann unter anderem die in HTML5 eingeführte JavaScript-Funktion pushState Verwendung finden [Goo15]. Diese Funktion kann den Browserverlauf manipulieren und diesem einen neuen Eintrag hinzuzufügen, ohne einen neuen HTTP-Request auszulösen. Bei dieser Technik werden Hyperlinks wieder im traditionellen Format angelegt, also z. B. auf *http://www.example.com/about*. Zur Verdeutlichung des Ablaufes soll Abbildung 18 dienen.

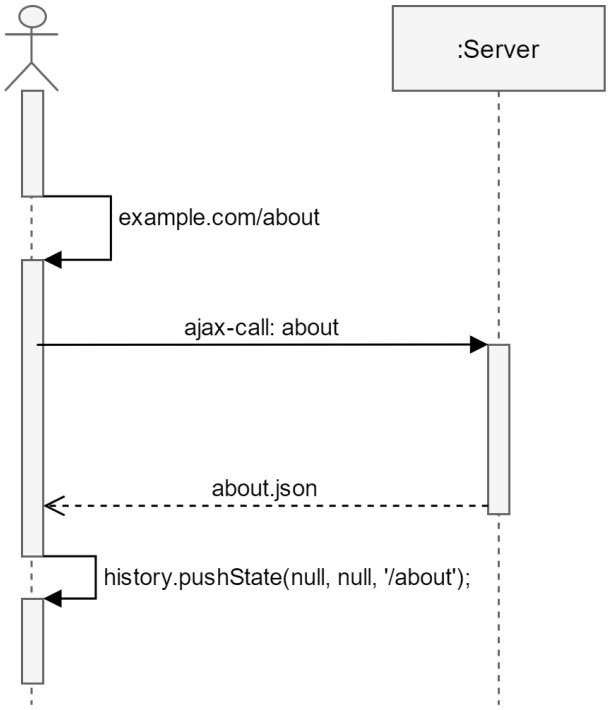

Abbildung 18: Ablauf des Webseitenwechsels in SPAs mit Verwendung der pushState-Funktion

Navigiert der Besucher nun von der Startseite auf das Impressum (*example.com/about*), wird der HTTP-Request abgefangen. Der benötigte Inhalt für das Impressum wird über Ajax geladen und in die Seite eingefügt. Im Anschluss wird durch die pushState-Funktion der Browserverlauf manipuliert. Nun ist der letzte Eintrag im Browserverlauf *http://www.example.com/news* und in der Adresszeile wird nun die vom Hyperlink angeforderte Adresse angezeigt. Für den Benutzer scheint es so, als ob er die alte HTML-Seite verlassen hätte und sich nun auf einer neuen befindet.
Nun muss der Webserver noch so konfiguriert werden, dass alle Anfragen, die mit der Basis-Adresse der SPA beginnen, auf selbige umgeleitet werden, damit ein direkter Aufruf, zum Beispiel von einer externen Seite, funktioniert. Sowohl *http://www.example.com/news* als auch *http://www.example.com/about* würde der Server nun auf *http://www.example.com/* weiterleiten. Die SPA würde nun den Zusatz der ursprünglich angeforderten Adresse */news* bzw. */about* erkennen und den entsprechenden Inhalt darstellen. Durch diesen Trick kann nun der Google Crawler jede Webseite einer SPA durch eine eindeutige Adresse erreichen.
Es ist somit darauf zu achten, dass das Framework korrekt von gängigen Suchmaschinen interpretiert wird und dass alle Webseiten durch Verwendung gültiger Links indiziert werden.

3.2.4 Lizenz

Da das Framework auch kommerziell genutzt werden soll, ist auch auf dessen Lizenz zu achten. Gängige Lizenzen, welche für eine kommerzielle Nutzung geeignet wären, sind zum Beispiel die 3-Klause-BSD- [Cal16] und MIT-Lizenz [Tec16].

3.3 Auswahl & Bewertung

Die in diesem Kapitel genannten Frameworks, also AngularJS, AngularJS2, Aurelia und React, sollen im Kapitel 5 versucht werden in das AEM zu integrieren. Zu beachten ist, dass im Folgenden React gelegentlich auch als Framework betitelt wird, auch wenn es sich hier streng genommen um eine Bibliothek handelt.

3.3.1 AngularJS

AngularJS ist ein von Google Inc. entwickeltes clientseitiges, Webframework. Mit dieser lassen sich SPAs entwickeln. Für den Entwickler ist es zudem möglich verschiedene MV*-Architektur zu realisieren, was durch vorgegebene Richtlinien zum Erstellen von Views, Controller Models erbracht wird. Hierzu gehören sowohl die MVC- als auch die MVVM-Entwurfsmuster [SS15].

3.3.1.1 Datenbindung

Eine Eigenschaft von AngularJS ist die automatisierte Datenbindung, um ein Model bzw. View-Model mit seiner View in beide Richtungen zu verknüpfen. Wird das Model geändert, so ändert sich auch die View entsprechend und umgekehrt. Siehe hierfür Codebeispiel 11.

```
1  <html ng-app>
2    <head>
3      <meta charset="utf8" />
4      <title>Datenbindung bei AngularJS</title>
5      <script src="angular.js"></script>
6    </head>
7    <body>
8      <form>
9        <input placeholder="Benutzername" type="text" ng-model="username">
10     </form>
```

```
11      <p>Hallo, {{username}}</p>
12    </body>
13  </html>
```

Codebeispiel 11: Datenbindung bei AngularJS

Um die Basisfunktionalität von AngularJS nutzen zu können, muss lediglich die Datei angular.js eingebunden werden, welche sich zum Beispiel von https://www.angularjs.org beziehen lässt.
AngularJS verwendet eine Palette von Attributen, welche mit „ng-“ beginnen. Diese Attribute werden auch als Direktiven bezeichnet [SS15, S. 40]. Die Direktive ng-app weist AngularJS darauf hin, dass es sich bei folgender HTML-Seite um eine AngularJS-Webanwendung handelt. Wird nun der Inhalt des mit ng-model versehenen Eingabefeldes verändert, so wird der Platzhalter {{username}} mit der entsprechenden Eingabe ersetzt.

3.3.1.2 Module

Module sind wieder auftretende Codestücke, welche vom Entwickler angelegt werden können. Zudem bietet AngularJS von Haus aus einige Module, wie zum Beispiel einen Controller für das MVC-Muster, verschiedene Filter, um Texte zu formatieren, und noch einige mehr. Die Aufteilung der Codestücke in einzelne Module reduziert die Komplexität und steigert somit auch die Wartbarkeit und vereinfacht das Testen.

3.3.1.3 Service

Ein Service dient zum Erstellen von Modulen und um die Geschäftslogik aus dem Controller auszulagern [SS15, S. 333]. So wird ein möglichst schlanker Controller erreicht. Zudem übernimmt der Service die mögliche Verwaltung von Zugriffsmethoden [SS15, S. 265].
In AngularJS kann ein Service mithilfe eines service-, eines factory- und eines provider-Konstrukts erstellt werden. Ein Service, der mittels des service-Konstrukts gebildet wurde, unterscheidet sich von einem Service, der mit dem factory-Konstrukt erstellt wurde, darin, dass seine Logik ausgelagert werden kann. Somit können komplexere Strukturen in Verbindung mit DI leichter erstellt werden und die Möglichkeit der Modularisierung wird geschaffen. Ein Provider hingegen erzeugt einen oder auch mehrere Services, die sich vor ihrer Initialisierung zunächst konfigurieren lassen [SS15, S. 340 f.].

3.3.1.4 Dependency-Injection

Alle Komponenten von AngularJS, also Module, Services etc. besitzen eine gewisse Abhängigkeit. So benötigt ein Controller oft einen Service zum Laden von Ressourcen und Module, um Daten weiter zu verarbeiten. Ein Modul kann wiederum von weiteren Modulen abhängig sein. Zudem benötigt eine View häufig einen Controller. All diese Abhängigkeiten werden bei AngularJS mithilfe von DI aufgelöst. Die Komponente muss sich hierbei nicht um die Bereitstellung ihrer benötigten Komponenten sorgen. Das entsprechende Subsystem von AngularJS kümmert sich um das Initialisieren der Komponenten, deren Abhängigkeiten und stellt sie anderen Komponenten zur Verfügung. Dank DI können leicht einzelne Komponenten ausgetauscht werden, somit wird die Testbarkeit vereinfacht [Goo16e].

3.3.1.5 Templates

Templates dienen bei AngularJS als Vorlage zum Erstellen von Views. Diese bestehen aus HTML und erweiterten dessen Spezifikation um weitere HTML-Tags und Attribute. Hierzu gehören Direktiven, siehe hierfür Unterunterabschnitt 3.3.1.6, Markups, wie sie zum Beispiel bei der Datenbindung vorkommen, diverse Filter, welche beispielsweise den Text eines Models formatieren, und Elemente zum Kontrollieren von HTML-Formularen. Diese Templates werden durch den HTML-Compiler von AngularJS in reguläres HTML umgewandelt [Goo16e].

3.3.1.6 Direktiven

Direktiven kommen in Templates zum Einsatz, um dem HTML-Compiler mitzuteilen, dessen DOM-Elementen ein gewisses Verhalten zu geben [Goo16e]. Zwei dieser Direktiven, und zwar ng-app und ng-model, wurden bereits in Codebeispiel 11 gezeigt. Neben weiteren Direktiven, welche AngularJS bereits mitbringt, können Entwickler auch eigene erstellen.

3.3.1.7 Scopes

Jedes Modul, jede Direktive und die Anwendung selbst besitzen ihren eigenen Scope, welcher Variablen beherbergt, die nur im jeweiligen Teil der Anwendung zur Verfügung stehen. Der Scope wird genutzt, um Objekte der View bereitzustellen [SS15, S. 37].

Innerhalb des Gültigkeitsbereiches kann der bereits vorhandene Scope verwendet oder es können auch untergeordnete Scopes erstellt werden. [SS15, S. 52]

3.3.2 AngularJS2

Zum Zeitpunkt dieser Arbeit ist Google Inc. dabei, den Nachfolger von AngularJS zu entwickeln, genannt AngularJS2. Hierbei handelt es sich um keine Weiterentwicklung von AngularJS, sondern es wird von Grund auf neu entwickelt.

3.3.2.1 Unterschied zu AngularJS

Eine Neuerung ist hier, neben einer anderen Syntax für Skripte und Templates, dass das gesamte Webframework unter anderem auch in TypeScript zur Verfügung gestellt wird. So stehen dem Entwickler einer Webanwendung neue Sprachelemente zur Verfügung. Da gängige Browser nativ kein TypeScript interpretieren können, ist es Voraussetzung, dieses zuerst mithilfe eines Transpilers in JavaScript umzuwandeln.
Für bestehende, in AngularJS verfasste Anwendungen, wird eine Anleitung für die Migration in AngularJS2 angeboten. Diese ist jedoch mit Vorsicht zu verwenden, da die Migration von komplexeren Anwendungen sich zum Teil schwieriger gestaltet und nicht immer auf Anhieb funktioniert [Goo16g].
Wurden in AngularJS Module noch in einer eigenen Syntax geschrieben, sind diese nun als ECMAScript 6-Module verfasst. Controller und Service sind nun nicht mehr in JavaScript-Funktionen gekapselt, dafür kommen ECMAScript 6-Klassen. Die in AngularJS verwendeten Direktiven und Controller wurde durch Komponenten ersetzt [Ran16, S. 16].
Folgende Liste zeigt einen Ausschnitt der wichtigsten Änderungen.

Template: Die Syntax der Templates wurde geändert. So würde die Anweisung für die Datenbindung aus Codebeispiel 11, Zeile 9, nicht mehr *ng-model="username"*, lauten, sondern *[(ngModel)]="username"*.

Module: Module in AngularJS2 entsprechen den ECMAScript 2015-Modulen.

Sytlesheets: Jeder AngularJS2-Komponente können eigene CSS-Dateien zugewiesen werden. Diese sind nur innerhalb der Komponente gültig und haben keinerlei Auswirkungen auf andere Komponenten.

Ein ausführlicher Vergleich beider Frameworks ist in der offiziellen Dokumentation von AngularJS2 zu finden [Goo16b].

3.3.2.2 Komponenten

Unter AngularJS2 versteht man eine Komponente für einen für den Besucher sichtbaren Teil, der innerhalb der Webanwendung wiederverwendet werden kann [Ran16, S. 67]. Die Verschachtelung von Komponenten und der Datenaustausch untereinander ist möglich [Ran16, S. 71]. Eine Komponente entspricht in etwa den in AngularJS verwendeten Direktiven und Controllern [Ran16, S. 16].
Die AngularJS2-Komponenten sind an Webkomponenten angelehnt. Eine Webkomponente ist ein Konzept aus der Webtechnologie, das vier Technologien vereint [Dee16, S. 3]. Dies sind folgende:

Templates: Bereits aus früheren Kapiteln bekannt.

Custom Elements: Eigens erstellte HTML-Elemente

Shadow DOM: Virtueller DOM, siehe Unterunterabschnitt 3.3.4.1.

HTML Imports: Importieren von HTML-Seiten von anderen HTML-Seiten aus.

3.3.2.3 Performance

Das Framework wurde im Hinblick auf mobile Endgeräte wie Tablets und Smartphone entwickelt, um auch hier einen möglichst stabilen und lauffähigen Code zu erlauben. Diese Entscheidung führt zu einer verbesserten Performance im mobilen und im Desktop-Bereich [Ran16, S. 14].

3.3.2.4 Datenbindung

In AngularJS2 hat sich der Mechanismus der Datenbindung drastisch geändert. Um Model und View zu synchronisieren, wurde in AngularJS das sogenannte Dirty Checking eingesetzt. Hierfür legte das Framework eine Liste an, in dem jede Eigenschaft eines Models, das in einer View projiziert wird, eingetragen ist. Es wird dann die Liste zyklisch durchlaufen und überprüft, ob sich eine Eigenschaft geändert hat. Dies konnte zur Folge haben, dass ein Model eine Direktive, eine Direktive ein Model, Direktiven andere Direktiven und Models andere Models anstoßen konnten, sich zu aktualisieren. In AngularJS2 wurde diese multidirektionale Aktualisierung abgeschafft. Stattdessen erfolgt diese nur noch in einer Richtung, sodass übergeordnete Komponenten immer vor dessen untergeordneten Komponenten aktualisiert werden. Dies vermeidet auch mögliche zyklische Änderungen, die in AngularJS noch möglich waren [Ran16, S. 144 ff.].

3.3.3 Aurelia

Hauptverantwortlicher für das Framework Aurelia ist Rob Eisenberg. Dieser entwickelte bereits das JavaScript Framework Durandal und half bei der Entwicklung von AngularJS2 mit. Da dessen Entwicklung Eisenberg jedoch nicht zufrieden stellte, entschloss er sich, das Google-Team zu verlassen und erneut ein neues Framework unter dem Namen Aurelia zu kreieren [Eis14].

3.3.3.1 Sprachen

Ähnlich wie bei AngularJS2 erlaubt es auch Aurelia, seine Webanwendungen in den Sprachen TypeScript und JavaScript zu schreiben. Bei JavaScript liegt die Wahl zwischen ECMAScript 6 und ESNext. Mit ESNext ist immer die neueste Version von ECMAScript gemeint, auch wenn diese sich noch in der Spezifikation befindet. Dies ist zum Zeitpunkt dieser Arbeit ECMAScript 7. Da diese zwar bereits spezifiziert, aber von den meisten Browsern noch nicht implementiert wird, siehe Tabelle 2, sollte bei Verwendung ebenfalls ein Transpiler, vergleichbar wie bei TypeScript, Einsatz finden, um den Quellcode in ECMAScript 5 oder 3, je nach gewünschtem Browsersupport, umzuwandeln.
Im Unterschied zu vielen anderen SPA-Frameworks kann mit Aurelia ein großer Teil der Webanwendung entwickelt werden, ohne die Aurelia-API zu verwenden. Somit wird erreicht, dass im Falle einer Migration zu einem anderen Framework nahezu die gesamte Geschäftslogik der Webanwendung beibehalten werden kann [Inc16b].

3.3.3.2 Templates

Bei den Templates strebt Aurelia ebenfalls den Versuch an, sich möglichst nahe an die Spezifikationen von ECMAScript zu halten. Innerhalb des Templates werden daher die in ECMAScript 6 eingeführten String-Interpolationen als Platzhalter für das Model eingesetzt. Anstelle von MVC wird hier häufig das MVVM-Entwurfsmuster verwendet. Schlussfolgernd wird, um Model und View interagieren zu lassen, ein ViewModel benötigt [Inc16c]. Mit Aurelia ist es jedoch auch möglich, eine Webanwendung mit dem MVC-Entwurfsmuster zu erstellen [Inc16b].

3.3.3.3 Dependency Injection

Auch Aurelia unterstützt das Konzept von DI. Es werden eine Reihe an Bibliotheken und APIs unterstützt. Innerhalb der Onlinedokumentation liegt der Fokus auf SystemJS,

zudem werden auch alle auf AMD-basierende Lösungen wie RequireJS, Cajo oder Dojo, und Modul-Bundler wie Webpack unterstützt [Inc16b].

3.3.3.4 Support

Aurelia bietet als einziges SPA-Framework eine kommerzielle Unterstützung an. Google(AngularJS/AngularJS2) und Facebook(React) bieten zwar den Quellcode an, jedoch ohne garantierte Funktionalität.
Zudem wird ein zusätzlicher Support angeboten, unter anderem Training, Consulting, Code-Review und weitere sich noch in Entwicklung befindliche Produkte [Inc16a].

3.3.4 React

Einen etwas anderen Ansatz verfolgt React. Es handelt sich bei dem von Facebook und Instagram entwickelten Code nicht um ein Framework, sondern um eine Bibliothek.

3.3.4.1 Virtueller DOM

React verfolgt einen anderen Ansatz der Datenbindung als die zuvor genannten Frameworks. Änderungen werden hier nicht direkt in den DOM eingetragen, sondern zunächst in einen virtuellen DOM. Dieser stellt eine abstrakte Kopie zum regulären DOM dar. Sollte ein Element verändert werden, wird diese Änderung zunächst im virtuellen DOM eingetragen. React ermittelt nun den Unterschied zum virtuellen und realen DOM und fügt in Letzteres nur die benötigten Änderungen ein. Dieser Vorgang des Vergleichens und dass nur die benötigten Teile aktualisiert werden, ist in der Regel sehr schnell [Fed15, S. 19].

3.3.4.2 JSX

Da es sich bei React um eine Bibliothek und kein Framework handelt stellt diese auch keine MV-* Architektur zur Verfügung. Vielmehr dient React mehr dazu die Präsentationsebene, also die View, zu erstellen. Hierfür kommt JavaScript XML (JSX) zum Einsatz. Das Erstellen neuer Elemente wird mit der Funktion React.createElement bewerkstelligt. Als Beispiel soll folgender HTML-Code generiert werden.

```
1 <Nav color="blue">
2   <Profile>
3     click
```

```
4     </Profile>
5   </Nav>
```

Codebeispiel 12: Zu generierender HTML-Code

Mit React würde dies mit der Funktion React.createElement geschehen.

```
1 var Nav, Profile;
2 var app = React.createElement(
3 Nav,
4 {color:"blue"},
5 React.createElement(Profile, null, "click")
6 );
```

Codebeispiel 13: Generierung von HTML in React

Das Codebeispiel 13 kann auch mit JSX geschrieben werden. Diese in React enthaltene Bibliothek wandelt XML-ähnlichen Code in JavaScript um.

```
1 var app = <Nav color="blue"><Profile>click</Profile></Nav>;
```

Codebeispiel 14: Generierung von HTML mit JSX

Somit wird durch den Einsatz von JSX das Codebeispiel 14 im Hintergrund in Codebeispiel 13 umgewandelt.

3.3.5 Vergleich

Folgende Tabelle 5 zeigt kurz zusammengefasst die wichtigsten Eigenschaft der zuvor genannten Webframeworks im Vergleich.

Webfr. / Eig.	**AngularJS**	**AngularJS2**	**Aurelia**	**React**
Entwurfsmuster	MVC, MVVM	MVC, MVVM	MVC, MVVM	keines
Lizenz	MIT	MIT	MIT	3-Klause-BSD
Sprache	JS	JS/TS	JS/TS	JS
Besonderheiten	Datenbindung	Verbesserte Laufzeit	ESNext, Support	Shadow DOM
Entwickler	Google	Google	Blue Spire	Facebook, Twitter
Webpräsenz	*http://angularjs.org/*	*http://angular.io*	*http://aurelia.io*	*http://facebook.github.io/react/*

Tabelle 5: Vergleich der vier Frameworks

4 Analyse des Ist-Zustand

Durch den Einsatz von AEM lassen sich bereits umfangreiche Webpräsenzen entwickeln. Dabei besteht jede Webseite erfahrungsgemäß aus mehreren wiederverwertbaren Komponenten.
Im Folgenden wird davon ausgegangen, dass bereits eine Webpräsenz innerhalb einer AEM-Instanz angesiedelt ist. Es besteht der Wunsch diese durch weitere AEM-Komponenten zu erweitern. Dabei wird abgesondert von der Zielplattform innerhalb einer Entwicklungsumgebung besagte Komponente entwickelt. Nach der Fertigstellung erfolgt die Integration der Komponente in die AEM-Instanz der Zielplattform. Die Komponente kann nun ein Autor innerhalb der Autoren-Bedienoberfläche aufrufen und in eine Webseite integrieren.
Besagte Komponenten basieren jedoch in erster Linie auf serverseitigen Technologien. Bei der Interaktion mit dem Besucher resultiert dies zu der Problematik, dass neue Inhalte nur serverseitig generiert werden können. Somit wird hier die Webseite komplett neu geladen, wie in Unterunterabschnitt 2.3.1.1 beschrieben wurde. Dieses Verhalten kann sich negativ auf das Nutzererlebnis des Besuchers auswirken. Dieser ist gerade von mobilen Geräten wie Tablets und Smartphones gewohnt, dass diese schnell und ohne größere Wartezeiten auf Benutzerinteraktionen reagieren und das gewünschte Resultat anzeigen [Riz13, S. 78]. Um dies auch in Webanwendungen zu ermöglichen, lassen sich clientseitige Webframeworks verwenden um so clientseitige Webanwendungen, wie jene in Unterunterabschnitt 2.3.1.2 beschrieben, zu verfassen.
Der Gedanke ist hier die vorteilhaften Eigenschaften von clientseitigen und serverseitigen Webframeworks zu kombinieren. Statische Inhalte wie Bilder und Nachrichten werden über das AEM gepflegt. Sich häufig ändernde Inhalte, wie Datenbankinhalte oder Informationen von teils externen REST-Schnittstellen lassen sich über besagte clientseitige Webframeworks zur Laufzeit der HTML-Seite nachladen und anzeigen.
Zu beachten ist, dass im Folgenden der Begriff „Webanwendung“ kurz für „clientseitige Webanwendung“ steht. Eine mit AEM verfasste Webanwendung wird AEM-Webanwendung genannt.

4.1 Ziel und bestehende Problematiken

Das Ziel ist es Webanwendungen, die mit clientseitigen Webframeworks entwickelt wurden in Webseiten, die mit AEM entwickelt wurden, zu integrieren. Die Integration soll hierbei in Form einer AEM-Komponente erfolgen. Die besagte Komponente soll wie gewohnt über die Autoren-Bedienoberfläche aufzurufen und bedienbar sein. Wird diese in einer Webseite eingebunden, werden autonom alle benötigten Ressourcen geladen und gewährleistet, dass die clientseitige Webanwendung in der Webseite wie vorgesehen dargestellt wird.
Der Entwicklungsprozess sieht hierbei vor, dass zunächst die clientseitige Webanwendung unter Ausschluss einer AEM-Instanz entwickelt wird. Erst nach Fertigstellung wird diese als AEM-Komponente angepasst und ggf. erweitert, um beispielsweise die Konfigurierbarkeit innerhalb der Autoren-Bedienoberfläche zu ermöglichen. Abschließend erfolgt die Integration in die Zielplattform.
Bei der Transformation in eine AEM-Komponente und dem Versuch der Integration können allerdings verschiedene Problematiken auftreten. Diese entstammen den Eigenschaften von AEM und den Webframeworks, aber auch durch gesetzte Rahmenbedingen und Restriktionen, welche die IT-Landschaft der Zielplattform und Integration betreffen.

4.1.1 IT-Landschaft der Zielplattform

Zumeist weisen die unterschiedlichen Zielplattformen, also jene Plattformen, auf dem eine AEM-Instanz läuft und schlussendlich eine Webanwendung integriert wird, untereinander abweichende Konfigurationen auf. Dies kann das Bereitstellen der Webanwendungen beträchtlich erschweren. Unterschiede in der Konfiguration wären zum Beispiel, dass bei manchen Zielplattformen aus Sicherheitsgründen gewisse Dienste, beispielsweise das Web-based Distributed Authoring and Versioning (WebDAV)-Protokoll, deaktiviert sind. Eine weitere mögliche Rahmenbedingung kann sein, dass die Ressourcen der Webanwendung sich nicht im AEM, sondern auf einen separaten Webserver befinden sollen.
Somit ergeben sich, abhängig von den Rahmenbedingen und der Konfiguration, unterschiedliche Herangehensweisen und Lösungen für die gestellte Aufgabe. Manche Lösungen sind hierbei für gewisse Zielplattformen besser oder schlechter geeignet, oder wegen den gesetzten Rahmenbedingungen als mögliche Lösung gar ausgeschlossen.

4.1.2 Bereitstellen von Ressourcen

Ein Hindernis ist das Bereitstellen der Ressourcen der Webanwendung. Der Grund hierfür ist die mehrstufige Auflösung einer Anfrage des Apache Sling Webframeworks, wie der

Abbildung 13 zu entnehmen ist.
Clientseitige Webanwendungen werden zumeist in lokalen Entwicklungsumgebungen erstellt und anschließend in das bestehende serverseitige System integriert. Bei Webservern wie dem Apache HTTP Server reicht es häufig die fertige Webanwendung auf den Webserver in ein entsprechendes Verzeichnis hochzuladen, um diese den Besucher zur Verfügung zu stellen. Das heißt die komplette Verzeichnisstruktur kann 1:1 erhalten bleiben.
Da bei AEM jedoch Ressourcen in das JCR abgelegt und über Apache Sling freigegeben werden, ist eine exakte Beibehaltung der Struktur nicht ohne weiteres möglich.
Allgemein erfordert Apache Sling, und somit auch AEM, ein fundamental anderes Programmierparadigma im Vergleich zu anderen Frameworks für die Erstellung von serverseitigen Webanwendungen. Es wird sehr der Fokus auf die Konfiguration gelegt und jede Konvention lässt sich durch entsprechende Einstellungen umgehen.
So ergibt es sich, dass die ursprüngliche Verzeichnisstruktur der Webanwendung bei der Integration abgeändert wird. Ressourcen werden in Gruppen wie Skripte und Bilder eingeteilt und unter verschiedenen Knoten des JCR abgelegt. Somit verschiebt sich die Verzeichnisstruktur und die relativen Pfade untereinander verändern sich. Zudem ist es das Ziel die Webanwendung in eine AEM-Komponente einzubetten, was eine zusätzliche Konfiguration voraussetzt.

4.1.3 Konflikte mit anderen Skripten

Innerhalb der Autoren-Bedienoberfläche wird die Seite so dargestellt, wie diese auch bei dem Besucher der Webseite in seinem Browser erscheinen würde. Im Hintergrund lädt AEM weitere JavaScript-Dateien, welche es den Autoren erlauben die Seite zu bearbeiten und zu konfigurieren. Es gilt zu überprüfen, ob eine integrierte Webanwendung möglicherweise zu Problemen in der Autoren-Bedienoberfläche führt. Dies beinhaltet zum einen Konflikte zwischen Skripten von AEM und der Webanwendung und zum anderen auch Darstellungsprobleme, also unterschiedlichen Darstellungen innerhalb der Autoren-Bedienoberfläche und beim Besucher der Webpräsenz.
Weiterhin können Versionskonflikte auftreten, falls die Webanwendung mit einer älteren Version des Webframeworks entwickelt wurde, innerhalb von AEM jedoch eine neuere Version Verwendung finden. Es gilt zu prüfen, ob derartige Konflikte bei den ausgewählten Webframeworks bestehen und auftreten können. Weiterhin bedarf es der Untersuchung, was geschieht wenn innerhalb einer Webseite das gleiche Webframework in unterschiedlichen Versionen, oder mehrere unterschiedliche Webframeworks Verwendung finden und ob dies womöglich zu Kollisionen führt.

5 Integration der Webframeworks in AEM

In diesem Kapitel werden verschiedene Möglichkeiten beschrieben, ein Webframework in eine mit AEM erstellte Webpräsenz zu integrieren. Dies umfasst auch die Aufbereitung der Webanwendung vor der Integration und Lösungen zur Problemen, welche nur indirekt mit der Integration zusammenhängen. Zum Ende dieses Kapitels erfolgt ein Vergleich der Lösungen mit einen kurzen Ausblick darauf wie diese sich auch miteinander kombinieren ließen.

5.1 Rahmenbedingungen

Der Versuch der Integration erfolgt in ein AEM der Version 6.0.0.20140515, ausgeführt unter Windows 10 x64 Version 1607, Build 14393.479 mit der Java Runtime Environment (JRE) Version 1.8.0_111-b14.

5.2 Zu untersuchende Frameworks

Im Folgenden soll der Fokus auf die vier Webframeworks aus Abschnitt 3.3 liegen. Diese sind AngularJS (Unterabschnitt 3.3.1), AngularJS2 (Unterabschnitt 3.3.2), Aurelia (Unterabschnitt 3.3.3) und React (Unterabschnitt 3.3.4).

5.3 Zu integrierende Webanwendungen

Als Basis für den Versuch der Integration wurde für jedes Webframework eine Webanwendung ausgewählt. Diese stammen größtenteils direkt von den Entwickeln des jeweiligen Webframeworks und gebrauchen bereits zahlreiche Funktionen der jeweils verfügbaren

Funktionalitäten. Der Quellcode liegt als eigens eingerichtetes GIT Respository zur Einsicht vor. Eine Übersicht der Webframeworks bietet Tabelle 6.

Framework	Basiert auf	GIT Respository
AngularJS 1.5.8	angular-phonecat [Ang16]	[Kan16a]
AngularJS2 2.0.0-rc.3	Tutorial: Tour of Heroes [Goo16f]	[Kan16b]
Aurelia		
React		

Tabelle 6: Webanwendungen

Im Folgenden wird vermehrt der Begriff „die Webanwendung“ fallen. Auch wenn hier der Singular verwendet wird, sind damit alle vier Webanwendungen gemeint.

5.4 Aufbereiten der Daten

Vor der Integration ist es gelegentlich ratsam die entwickelte Webanwendung entsprechend aufzubereiten. Dies kann die Performance verbessern und den Prozess der Integration vereinfachen. In diesen Schritt finden Werkzeuge wie Grunt und Gulp aus Unterabschnitt 3.1.3 ihre Verwendung.

5.4.1 Minimierung

Für den Produktiveinsatz ist es ratsam die Quelldateien zu minimieren und konkatenieren. Bei der Minimierung werden zum Beispiel unnötige Leerzeichen entfernt, oder auch lange Variablennamen durch kürzere ersetzt. Bei der Konkatenierung wird die Anzahl der einzelnen Dateien minimiert, indem diese zu einer großen Datei zusammen gefügt werden. Beides erhöht die Übertragungszeit vom Server zum Client, da weniger Anfragen gestellt werden müssen, und auch das zu übertragende Volumen sinkt. Weiterhin existieren so auch weniger Dateien, die es in das AEM zu integrieren gilt, was zum Beispiel die direkte Integration wie in Unterabschnitt 5.5.1 beschrieben erleichtert.
Weiterhin lassen sich bei manchen Frameworks die HTML-Templates in JavaScript-Dateien umwandeln. Somit ist auch hier eine anschließende Minimierung und Komprimierung möglich.

Allgemein Generell existieren für die Minimierung von JavaScript-Quellcode verschiedenste Werkzeuge. Eines davon wäre UglifyJS [Baz16]. Hiermit lässt sich JavaScript-Quellcode unter anderem minimieren und konkatenieren. Das Werkzeug wird unter

der BSD-Lizenz veröffentlicht und wird über npm installiert. Bei der Konkatenierung ist darauf zu achten, die JavaScript-Dateien in der korrekten Reihenfolge anzugeben, um Fehler in der Abhängigkeit zu vermeiden.
Um solche Abhängigkeitsfehler zu vermeiden lässt sich hier das Konzept der DI nutzen. Wurde die Webanwendung unter Verwendung von DI entwickelt, so ist in den JavaScript-Datei bereits deren Abhängigkeit syntaktisch hinterlegt. Je nach Art der verwendeten DI darf der Entwickler auf unterschiedliche Werkzeuge zurückgreifen. Alle gängigen Bibliotheken aus Tabelle 3 liefern bereits Werkzeuge für die Konkatenierung mit.

AngularJS Alle AngularJS-Templates, welche sich im Normalfall in einer jeweils eigenen HTML-Datei befinden, lassen sich in eine eigene JavaScript-Datei packen. Als Werkzeug wird hier Grunt in Verbindung mit dem npm-Packet Namens grunt-angular-template [Cle16] verwendet. Dies nimmt die AngularJS-Templates und wandelt diese in eine JavaScript-Datei um, welche den Template-Cache [Goo16d] von AngularJS nutzt.

AngularJS2 In der Dokumentation von AngularJS2 wird der Ahead-of-Time (Vorzeitig, AoT)-Ansatz erklärt [Goo16a]. Im Gegensatz zum Just-in-Time (Rechtzeitig, JiT)-Ansatz, bei dem Skripte und Ressourcen erst dann geladen werden, wenn sie benötigt sind, wird hier alles, was später benötigt werden könnte, zum Anfang geladen.

Aurelia Auch Aurelia bietet ein derartiges Tool an. Dieses nutzt Gulp und ist über npm mit den Namen aurelia-bundler [Aur16] zu finden.

React Um die Laufzeit der Webanwendung zu optimieren, kann man diesen zuvor in normale JavaScript-Anweisung umwandeln. Hierfür empfiehlt sich der Einsatz von Babel [Bab16]. Das Werkzeug wird über npm installiert, kann TypeScript und neueres ECMAScript in ECMAscript 3 und 5 umwandelt, und beherrscht auch JSX. Die Umwandlung lässt sich manuell, oder auch wahlweise durch Gulp und Grunt anstoßen.

5.5 Möglichkeiten der Integration

Abhängig von verschiedenen Faktoren wurden unterschiedliche Möglichkeiten erarbeitet um eine Webanwendung in eine AEM-Instanz zu integrieren. Neben dem zugrunde liegenden clientseitigen Webframework ist der wichtigste Entscheidungsfaktor dafür, welche Lösung genutzt werden kann, die IT-Landschaft der Zielplatz. Je nach dessen Konfiguration bieten sich gewisse Lösungen mehr an, oder sind auch als mögliche Lösung von vorne

herein ausgeschlossen.

5.5.1 Direkte Integration in das AEM

Sofern der Zugriff auf das Ziel-AEM, insbesondere auf dessen JCR-Strukur, besteht, lässt sich die Webanwendung direkt in selbiges durch die Verwendung von Clientlibs integrieren.
Im ersten Schritt wird das gewünschte JavaScript-Framework als Clientlib, wie in Absatz 2.3.4.1.10 beschrieben, bereitgestellt. Sollte das Framework mehrmals Verwendung finden empfiehlt es sich dies unter /etc/clientlibs zu platzieren.
Um die Bereitstellung der Ressourcen für die Webanwendung zu ermöglichen existieren mehrere Ansätze. Je nach verwendetem Framework lassen sich bestimmte Ansätze leichter erreichen.

5.5.1.1 Verwendung als Clientlib

Besteht die Webanwendung lediglich aus JavaScript und Stylesheets, bestellt die Möglichkeit diese komplett als Clientlib umzusetzen. Hierfür wird wie in Absatz 2.3.4.1.10 beschrieben im JCR eine entsprechende Struktur benötigt. Das Übertragen der Ressourcen der Webanwendung in das JCR kann über mehrere Wege erfolgen.

5.5.1.1.1 Transferieren der Ressourcen in das AEM

Zunächst lässt sich über die WebDAV-Schnittstelle mit einem entsprechenden Programm die Struktur des JCR anzeigen und bearbeiten. Neben dem Löschen und Verschieben von Knoten ist hier auch das Hochladen von lokalen Dateien möglich.
Sofern die WebDAV-Schnittstelle auf der Zielplattform nicht zugänglich sein sollte, weil diese zum Beispiel deaktiviert wurde, gibt es die Alternative die Ressourcen als CRX-Paket bereit zu stellen. Hierfür werden zunächst wieder alle Inhalte in das JCR, zum Beispiel über WebDAV, geladen, dieses mal jedoch in dem vom Entwickler. Anschließend kann über die Autoren-Bedienoberfläche, siehe Absatz 2.3.4.1.5, ein CRX-Paket erstellt werden. Dieses ist ein Zip-Archiv mit Metainformationen, wie Name und Version des CRX-Pakets.
Das Erstellen eines CRX-Paketes erfolgt über eine entsprechende Seite des CRXDE Lite. Hier werden besagte Metainformationen gesetzt und alle benötigten Knoten für das CRX-Paket angegeben. Nun erfolgt der Export des CRX-Pakets. Ist dies geschehen wird

das CRX-Paket in die Zielplattform importiert. Der Inhalt des CRX-Paketes wird nun in das JCR geschrieben. Bereits bestehende Knoten werden bei diesem Vorgang ggf. überschrieben.

5.5.1.2 Content Ordner

Jeglicher Inhalt, der unter /content abgelegt wird, ist direkt über einen HTTP-Request aufrufbar. Unter der Standardkonfiguration von AEM ist die Datei `/content/myapp/index.html` zu erreichen über `http://<Server-Adresse>:<Port>/content/myapp/index.app`.

5.5.1.3 Hybride

Innerhalb einer Clientlib lassen sich lediglich JavaScript und CSS hinterlegen. Falls eine Webanwendung aber noch weitere Ressourcen wie Bilder und HTML beinhaltet, so werden diese unter einen anderen Knoten abgelegt. Es bietet sich an diese unter `/content` abzulegen, da wie zuvor beschrieben der Zugriff auf hier abgelegte Dateien direkt zur Verfügung steht.

5.5.2 Ansatz Java

Hier ist die Grundidee, dass die Webanwendung sich nicht im AEM befindet, sondern sich auf einen separaten Webserver liegt. Ziel ist es, dass die Webanwendung autonom auf dem separaten Webserver lauffähig ist und über eine AEM-Komponente auf Basis von Java und JavaScript in das AEM integriert wird.
Im Folgenden wird der separate Webserver, auf dem sich die Ressourcen für die Webanwendung befinden, kurz Webserver, und der Server mit der lauffähigen AEM-Instanz kurz AEM-Server genannt. Die Integration erfolgt somit vom Webserver in den AEM-Server. Die Webanwendung ist über *http://www.example.com:3000/spa/* aufrufbar und soll später über den AEM-Server über *http://aem.example.com/* erreichbar sein.

5.5.2.1 Erklärung

In der Regel erfolgt das Laden einer Webanwendung wie in Unterabschnitt 2.2.7 beschrieben in drei Schritten. Zunächst wird die HTML-Seite angefordert, anschließend wird der Quellcode der HTML-Seite nach weiteren Ressourcen durchsucht und auch diese geladen.

Weitere Ressourcen werden nun über Ajax angefordert und vom Server retourniert.
Es wird davon ausgegangen, dass die Schritte unter *http://www.example.com:3000/spa* / korrekt durchlaufen werden und somit die Webanwendung wie gewünscht im Browser erscheint.
Innerhalb von AEM hat die AEM-Komponente die Aufgabe die Webanwendung vom Webserver zu laden. Dabei muss jedoch die HTML-Seite, die Ajax-Anfragen und ggf. auch weitere Ressourcen manipuliert werden. Die Komponente dient somit als Bindeglied zwischen AEM und dem Webserver.
Zuvor muss jedoch der Webserver und die Webanwendung gewissen Anpassungen unterliegen. Diese werden in Unterunterabschnitt 5.5.2.2 und Unterunterabschnitt 5.5.2.3 beschrieben.

5.5.2.2 Anpassen des Webservers

Gängige Webbrowser verbieten, dass JavaScript, und somit auch Ajax, auf externe Webserver zugreifen dürfen. Grund dafür ist das Sicherheitskonzept der Same-Origin-Policy (SOP).
Unterscheiden sich die URL von der Webseite von der aus die Anfrage gestartet wurde und vom Ziel in Protokoll, Domain oder Port, so wird die Anfrage vom Webbrowser geblockt. Im zuvor beschriebenen Beispiel würden sich *http://aem.example.com/* und *http://www.example.com:3000/spa/* im Host (www.example.com anstelle von aem.example.com) und dem Port (3000 anstelle von 80) unterscheiden.
Damit der Zugriff doch funktioniert lässt sich der Mechanismus des Cross-Origin Resource Sharing (CORS) verwenden.

CORS etwas detailierter erklären

Hierfür antwortet der Zielwebserver der Anfrage mit einen entsprechenden HTTP-Header. Hier sind alle URLs gelistet, von denen gestattet sind eine Anfrage zu auszuführen. Der Browser vergleicht jetzt den HTTP-Header mit der URL, von dem die Anfrage gestartet wurde. Im Erfolgsfall blockiert der Browser nicht und der Aufruf wird zu Ende ausgeführt.

5.5.2.3 Anpassen der Webanwendung

Auch die Webanwendung muss unter Umständen noch vor dem Produktiveinsatz konfiguriert werden.
Sollten HTML-Templates in AngularJS Verwendung finden, kann es hier bei dem La-

denvorgang zu einer Strong Contextual Escaping (SCE)-Fehlermeldung führen. Dies geschieht, da der HTML-Code von extern kommt und potenziell unsichereren Code mitführen kann. AngularJS blockt daher per Standardkonfiguration den Ladevorgang. Doch durch entsprechende Anpassungen der Webanwendung lassen sich Ausnahmen hinzufügen, oder das SCE auch vollständig deaktivieren [Goo16c].

Andere Frameworks?

5.5.2.4 Ablauf der AEM-Komponente

Den ungefähren Ablauf der AEM-Komponente ist Abbildung 19 zu entnehmen.

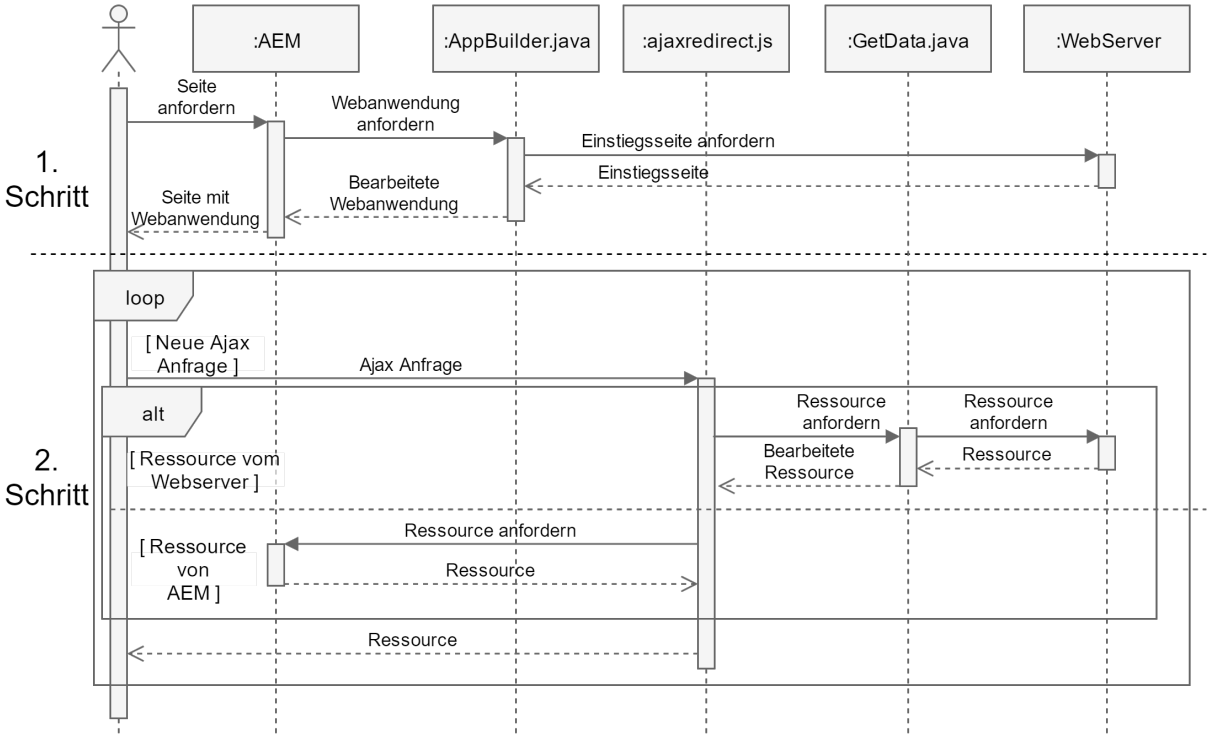

Abbildung 19: Ungefährer Ablauf der AEM-Komponente auf Java-Basis

Der erste Schritt wird im folgenden Absatz 5.5.2.4.1, der zweite Schritt im darauffolgenden Absatz 5.5.2.4.2 erklärt.

5.5.2.4.1 HTML-Seite aufbereiten

Der Benutzer fordert zunächst die Seite *http://aem.example.com/* an. In dieser befindet sich die AEM-Komponente, welche zuvor so konfiguriert wurde, dass an dessen Stelle die Webanwendung von *http://www.example.com:3000/spa/* erscheinen soll. Nun wird

`AppBuilder.java` damit beauftragt den HTML-Quellcode der Einstiegsseite der Webanwendung anzufordern, welche sich unter *http://www.example.com:3000/spa/index.html* befindet. `AppBuilder.java` muss jetzt den Quellcode für die Integration aufbereiten. Einige dieser Aufbereitungsaufgabe sind Anwendungsspezifisch und vom jeweils verwendeten Webframework abhängig. Mehr zu den Unterschieden ist dem Unterunterabschnitt 5.5.2.5 zu entnehmen. Was immer geschehen muss, ist, dass relative Referenzen auf JavaScript und CSS-Dateien durch absolute ersetzt werden. Eine Referenz auf *./app.module.js* sollte nach besagten Schritt auf *http://www.example.com:3000/spa/app.module.js* verweisen.
Der Quellcode von *http://www.example.com:3000/spa/index.html* könnte wie in Codebeispiel 15 aussehen.

```
1  <!DOCTYPE html>
2  <html ng-app="phonecatAPP">
3    <head>
4      <link rel="stylesheet" href="app.css" />
5      <script src="bower_components/angular/angular.js"></script>
6      <script src="app.module.js"></script>
7      <title>My Webbapplication</title>
8    </head>
9    <body>
10     <div class="view-container">
11       <div ng-view class="view-frame"></div>
12     </div>
13   </body>
14 </html>
```

Codebeispiel 15: Ausgangssituation auf Server B

Aus den Aufbereitung würde ein Quellcode wie in Codebeispiel 16 resultieren.

```
1 <link rel="stylesheet" href="http://www.example.com:3000/spa/app.css" />
2 <script src="http://www.example.com:3000/spa/bower_components/angular/
    ↪ angular.js"></script>
3 <script src="http://www.example.com:3000/spa/app.module.js"></script>
4 <div ng-app="phonecatApp">
5   <div class="view-container">
6     <div ng-view class="view-frame"></div>
7   </div>
8 </div>
```

Codebeispiel 16: Aufbereiteter Quellcode

AEM ersetzt nun den Platzhalter der Komponente mit den aufbereiteten Quellcode. Diese HTML-Seite ist nun fertig gerendert und wird wieder an den Client geliefert.

`AppBuilder.java` nutzt die Java-Bibliothek jsoup [Hed16]. Dieser HTML-Parser bietet Klassen und Funktionen zum Laden, Traversieren und Manipulieren von HTML-Seiten an. Der DOM der geladenen HTML-Seite kann mithilfe von CSS-Selektoren nach HTML-Elementen, wie den script- und link-Elementen, durchsucht werden. Die Installation der Bibliothek zur Nutzung innerhalb von AEM erfolgt als OSGi-Bundle.

5.5.2.4.2 Ajax-Anfragen anpassen

Im zweiten Schritt werden Ajax-Anfragen umgeleitet. Diese sind zumeist ebenfalls relativ und würden somit versuchen eine Ressource unter *http://aem.example.com/* anfordern, die sich jedoch unter *http://www.example.com:3000/spa/* befindet. Gerade DI-Bibliotheken laden läufig Skripte nach.
Für die Umleitung überschreibt `ajaxredirect.js` das XMLHttpRequest (XHR)-Objekt. Die XHR-Schnittstelle wird vom W3C spezifiziert und ist ein wesentlicher Bestandteil von Ajax, da hiermit Daten asynchron zwischen Server und Client auszutauschen.
Das überschriebene XHR-Objekt überprüft ob es sich bei der angeforderten URL um eine relative oder absolute handelt. Falls diese relativ ist hat dies zu bedeuten, dass die Ressource sich unter *http://www.example.com:3000/spa/* befindet. In dem Fall wird die Anfrage nun über ein Servlet, hier mit dem Namen `GetData.java`, umgeleitet. Die Umleitung über das Servlet anstelle einer direkten Anfrage an den Webserver hat zwei Vorteile.
Zum einen werden hier Probleme mit Zugriffsrechten, wie das in Unterunterabschnitt 5.5.2.2 erläuterte Sicherheitskonzept SOP, erleichtert. Ohne Servlet würde der Client direkt auf Ressourcen des Webservers zugreifen, was in der Regel zu einem Fehler führt. Der Webserver müsste somit allen Clienten den Zugriff autorisieren. Über ein Servlet jedoch ist es so, dass der AEM-Server die Anfrage an den Webserver stellt. Somit kann der Webserver so konfiguriert werden, dass er nur Anfragen vom AEM-Server erlaubt.
Zum anderen können an dieser Stelle noch Änderungen an den angefragten Ressourcen erfolgen. Zum Beispiel könnten hier die Templates einer AngularJS-Webanwendung nach relativen Hyperlinks durchsucht und diese angepasst werden.

Ist sie relativ, wird dieser die *http://www.example.com:3000/spa/* am Anfang angefügt. Je nach Anwendung wird auch die Adresse vom AEM, also *http://aem.example.com/* , angehängt, zum Beispiel wenn sich die Bilder dort befinden. Zuletzt wird die Ajax-Anfrage normal ausgeführt.

5.5.2.5 Aufbereitungsaufgabe bei unterschiedlichen Frameworks

Fehlt komplett...

5.5.2.6 Konfiguration

Des weilen wird gewünscht die Komponente zu konfigurieren. Da sich diese nicht direkt im AEM befindet bedarf es einer Schnittstelle für den Austausch der Konfigurationen zwischen AEM und dem Webserver.

Mehr Text

5.5.2.7 Varianten

Fehlt komplett...

5.5.3 Ansatz JavaScript

Im Gegensatz zur Java-Variante wird nun der erste Schritt von Abbildung 19 durch eine JavaScript Variante ersetzt. Das Prinzip ist hierbei wie bei dem Java Ansatz, jedoch gibt es bei dem zweiten Schritt eine Besonderheit. Und zwar ist darauf zu achten, dass die in der HTML-Seite referenzierten Ressourcen in der korrekten Reihenfolge geladen werden.

Mehr Text

5.6 Proxy

Fehlt komplett...

5.7 Suchmaschinen

Falls die Webanwendung die in Unterabschnitt 3.2.3 beschriebene pushState-Funktion verwendet, so ist eine entsprechende Konfiguration des AEM Server von Nöten, um alle Anfragen, welche die SPA betreffen, dementsprechend umleiten.
Als Beispiel soll eine AngularJS-Anwendung dienen, welche unter `/content/angular/`

liegt. Sie soll über *http://localhost:4502/spa* zu erreichen sein. Somit sind alle Ressourcen der Webanwendung auch unter besagter Adresse erreichbar.
Es wird angenommen, dass alle Adressen, die einen Punkt haben, einer Ressource wie Skripte, Bilder etc. entsprechen. Ansonsten handelt es sich um eine Webseite innerhalb der SPA. Alle Webseiten werden auf die HTML-Seite der SPA umgeleitet, im Beispiel auf `index.html`. Alle weiteren Ressourcen erhalten eine relative Umleitung. Tabelle 7 zeigt beispielhaft einige vom Browser angeforderte Ressourcen und unter welchen JCR-Knoten diese zu finden sind.

Angeforderte Adressen	Ressourcetyp	Hat Punkt	Ziel JCR-Knoten
http://localhost:4502/spa/	Webseite	Nein	*/content/angular/index.html*
http://localhost:4502/spa/news/	Webseite	Nein	*/content/angular/index.html*
http://localhost:4502/spa/about/	Webseite	Nein	*/content/angular/index.html*
http://localhost:4502/img/logo.png	Bild	Ja	*/content/angular/img/logo.png*
http://localhost:4502/spa/ressources/style.css	Stylesheet	Ja	*/content/angular/ressources/style.css*

Tabelle 7: Beispiel für die interne Umleitung

Hier kommt der Apache Sling Resource Resolver zum Einsatz [Fou16]. Über das JCR werden Regeln für die Umleitung der angeforderten Adressen gesetzt. In der Standardkonfiguration befinden sich diese unter `/http/map/`. Jede Regel setzt sich auch der hierarchischen Struktur und verschiedenen Eigenschaften zusammen. Um die anforderten Adressen aus Tabelle 7 entsprechend umzuleiten empfiehlt sich die Struktur wie in Codebeispiel 17.

```
1  /etc/map/http/
2  + localhost.4502/
3    - jcr:primaryType (Name) = sling:Mapping
4    + spa/
5      - jcr:primaryType (Name) = sling:Mapping
6      - sling:internalRedirect (String) = /content/angular
7      + path/
8        - jcr:primaryType (Name) = sling:Mapping
9        - sling:match (String) = ([^.]*)
10       - sling:internalRedirect (String) = /content/angular/index.html
```

Codebeispiel 17: Konfigurationsbeispiel für den Apache Sling Resource Resolver

Die Regeln basieren auf regulären Ausdrücken. Bei regulären Ausdrücken handelt es sich um eine Notation zur Beschreibung von Textmustern. Durch die Hinzunahme einer geeigneten Programmiersprache ist es möglich Texte nach dem Textmuster zu durchsuchen und diesen so zu erweitern, zu reduzieren und zu manipulieren [Fri09, S. 1 f.].
Der reguläre Ausdruck wird aus den konkatenierten Namen der Knoten, bzw. dem Wert der Eigenschaft sling:match, falls ein Knoten diese besitzt, zusammengesetzt. Der Wert der Eigenschaft sling:internalRedirct steht für die Zieladresse im JCR.
Aus der gegebenen Konfiguration resultieren zwei Regeln. Der Knoten `/etc/map/http/localhost.4502/spa/path/` beschreibt, dass alle Adresse, die mit *http://localhost:4502/spa/path/* beginnen, aber keinen Punkt in der Adresse, auf `/content/angular/index.html` verweisen. Ansonsten gilt die erstellte Regel unter Knoten `/etc/map/http/localhost.4502/spa/`

Wie die Auflösung verschiedener Adressen geschehen würde ist der Abbildung 20 zu entnehmen.

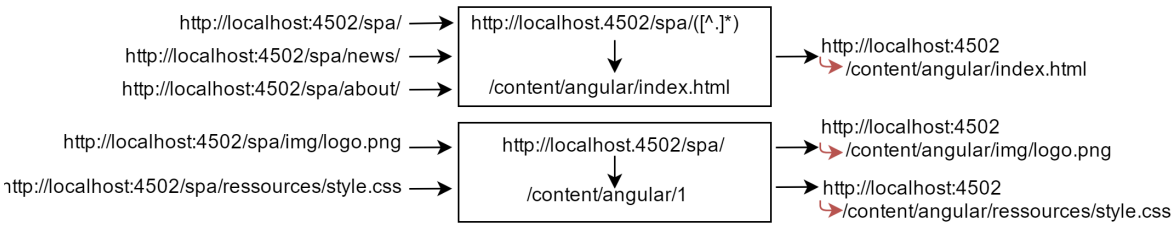

Abbildung 20: Möglichkeit zum Auflösen von Anfragen unter AEM

Text

5.8 Bewertung der Lösungen

Fehlt komplett...

6 Zusammenfassung

Fehlt komplett...

6.1 Fazit

Fehlt komplett...

6.2 Ausblick

Fehlt komplett...

Literaturverzeichnis

[Ado16a] Adobe: *Components*. https://docs.adobe.com/docs/en/aem/6-2/develop/components.html. Version: dec 2016

[Ado16b] Adobe: *Experience Server (CRX) and Jackrabbit*. https://docs.adobe.com/docs/en/aem/6-2/develop/the-basics.html#ExperienceServer(CRX)andJackrabbit. Version: dec 2016

[Ado16c] Adobe: *HTML Template Language*. https://docs.adobe.com/docs/en/htl/overview.html. Version: dec 2016

[Ado16d] Adobe: *Sling API*. https://docs.adobe.com/docs/en/aem/6-2/develop/the-basics.html#SlingAPI. Version: dec 2016

[Alm16] Alman, Ben: *gruntjs.com*. http://gruntjs.com/. Version: dec 2016

[Ang16] Angular: *AngularJS Phone Catalog Tutorial Application*. https://github.com/angular/angular-phonecat. Version: dec 2016

[Aur16] Aurelia: *GitHub - aureliabundler*. https://github.com/aurelia/bundler. Version: nov 2016

[Bab16] Babel: *Babel - The compiler for writing next generation JavaScript*. https://babeljs.io/. Version: dec 2016

[Baz16] Bazon, Mihai: *GitHub - mishoo UglifyJS*. https://github.com/mishoo/UglifyJS. Version: dec 2016

[BGP00] Baresi, Luciano ; Garzotto, Franca ; Paolini, Paolo: *Conceptual Modeling for E-Business and the Web*. Springer Verlag, 2000

[BS11] Bongers, Frank (Hrsg.) ; Stoeckl, Andreas (Hrsg.): *Einstieg in TYPO3 4.5*. Gali, 2011

[Cal16] California, University of: *The BSD 3-Clause License*. https://opensource.org/licenses/BSD-3-Clause. Version: oct 2016

[Cle16] Clemmons, Erric: *GitHub - ericclemmonsgrunt-angular-template*. https://github.com/ericclemmons/grunt-angular-templates#license. Version: nov 2016

[Con16] Consortium, World Wide W.: *Links*. https://www.w3.org/TR/html4/struct/links.html. Version: oct 2016

[DeB16] http://www.modulecounts.com/

[Dee16] Deeleman, Paplo: *Learning Angular 2*. Packt Publishing, 2016

[Eis14] Eisenberg, Rob: *Leaving Angular*. http://eisenbergeffect.bluespire.com/leaving-angular/. Version: 11 2014

[Fed15] Fedosejev, Artemij: *React.js Essentials*. Packt Publishing, 2015

[Fou16] Foundation, Apache S.: *Mappings for Resource Resolution*. https://sling.apache.org/documentation/the-sling-engine/mappings-for-resource-resolution.html. Version: nov 2016

[Fra16] Fractal: *gulpjs.com*. http://gulpjs.com/. Version: oct 2016

[Fri09] Friedl, J.E.F.: *Reguläre Ausdrücke*. O'Reilly Verlag, 2009

[Gar11] Garofalo, Raffaele: *Building Enterprise Applications with Windows Presentation Foundation and the MVVM*. O'Reilly Verlag, 2011

[GD13] Goll, Joachim ; Dausmann, Manfred: *Architektur- und Entwurfsmuster der Softwaretechnik*. Springer Vieweg, 2013

[GHJV09] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Entwurfsmuster - Elemente wiederverwendbarer objektorientierter Software*. Addison-Wesley, 2009

[Goo09] Google: *Making AJAX applications crawlable*. https://webmasters.googleblog.com/2009/10/proposal-for-making-ajax-crawlable.html. Version: oct 2009

[Goo15] Google: *Deprecating our AJAX crawling scheme*. https://webmasters.googleblog.com/2015/10/deprecating-our-ajax-crawling-scheme.html. Version: oct 2015

[Goo16a] Google: *Ahead-of-Time Complilation*. https://angular.io/docs/ts/latest/cookbook/aot-compiler.html. Version: oct 2016

[Goo16b] Google: *Angular 1 to 2 Quick Reference*. https://angular.io/docs/ts/latest/cookbook/a1-a2-quick-reference.html. Version: nov 2016

[Goo16c] Google: *AngularJS: API: $sceDelegateProvider*. https://docs.angularjs.org/api/ng/provider/$sceDelegateProvider. Version: nov 2016

[Goo16d] Google: *AngularJS: API: $templateCache*. https://docs.angularjs.org/api/ng/service/$templateCache. Version: nov 2016

[Goo16e] Google: *Developer Guide*. https://docs.angularjs.org/. Version: Juni 2016

[Goo16f] Google: *TUTORIAL: TOUR OF HEROES*. https://angular.io/docs/ts/latest/tutorial/. Version: dec 2016

[Goo16g] Google: *Upgrading from 1.X*. https://angular.io/docs/ts/latest/guide/upgrade.html. Version: oct 2016

[Hed16] Hedley, Jonathan: *jsoup Java HTML Parser*. https://jsoup.org/. Version: nov 2016

[Inc15] Incorporated, Adobe S.: *Adobe Experience Manager - Student Workbook*. 2015

[Inc16a] Inc., Blue S.: *Business Advantages*. http://aurelia.io/hub.html#/doc/article/aurelia/framework/latest/business-advantages/. Version: oct 2016

[Inc16b] Inc., Blue S.: *Quick Start*. http://aurelia.io/hub.html#/doc/article/aurelia/framework/latest/quick-start/. Version: oct 2016

[Inc16c] Inc., Blue S.: *Templating Basics*. http://aurelia.io/hub.html#/doc/article/aurelia/templating/latest/templating-basics/. Version: oct 2016

[JG16] Jacobsen, Jens ; Gidda, Matthias: *Webseiten erstellen fuer Einsteiger*. Rheinwerk Verlag, 2016

[Kan16a] Kandler, Julian: *AngularJS-Example*. https://github.com/KandlerLi/AngularJS-Example. Version: dec 2016

[Kan16b] Kandler, Julian: *AngularJS2-Example*. https://github.com/KandlerLi/AngularJS2-Example. Version: dec 2016

[KRM15] Kessler, Esther ; Rabsch, Stefan ; Mandic, Mirco: *Erfolgreiche Websites - SEO, SEM, Online Marketing, Usability*. Rheinwerk Verlag, 2015

[KRR03] Kappe, Gerti ; Reich, Birgit Proelland S. ; Retschitzegger, Werner: *Web Engineering - Systematische Entwicklung von Webanwendungen*. dpunkt.verlag, 2003

[Lub07] Lubkowitz, Mark: *Webseiten programmieren und gestalten*. Galileo Computing, 2007

[New08] Newman, Scott: *Django 1.0 Template Development*. Packt Publishing, 2008 (From technologies to solutions)

[Ran16] Rangle.io: *Rangle's Angular 2 Training Book*. GitBook, 2016 https://www.gitbook.com/book/rangle-io/ngcourse2/details

[Rie09] Rieber, Philipp: *Dynamische Webseiten in der Praxis: mit PHP 5, MySQL 5, XHTML, CSS, JavaScript und AJAX, 2. Auflage*. Mitp, 2009

[Riz13] Rizvanoglu, Kerem: *Research and Design Innovations for Mobile User Experience*. IGI Global, 2013

[Seb10] Sebestyen, Thomas: *XML: Einstieg für Anspruchsvolle*. Addison-Wesley, 2010

[Spo09] Spoerrer, Stefan: *Content-Management-Systeme: Begriffsstruktur und Praxisbeispiel*. Kölner Wissenschaftsverlag, 2009

[SS15] Steyer, M. ; Softic, V.: *Angular JS: Moderne Webanwendungen und Single Page Applications mit JavaScript*. O'Reilly Verlag, 2015. – ISBN 9783955619510

[Ste11] Steyer, R.: *jQuery: das JavaScript-Framework für interaktives Design*. Pearson Deutschland, 2011 (Open source library). – ISBN 9783827330727

[Tec16] Technology, Massachusetts I.: *The MIT License (MIT)*. https://opensource.org/licenses/mit-license.php. Version: oct 2016

[Twi16] Twitter: *Bower.io*. https://bower.io/. Version: oct 2016

[W3C14] W3C: *Open Web Platform Milestone Achieved with HTML5 Recommendation*. https://www.w3.org/2014/10/html5-rec.html.en. Version: oct 2014

[Wis12] Wissmann, Dieter: *JavaServer Pages: Dynamische Websites mit JSP erstellen*. W3L-Verlag, 2012

[ZAP16] Zaytsev, Juriy ; Arnott, Leon ; Pushkarev, Denis: *ECMAScript compatibility table*. http://kangax.github.io/compat-table/. Version: Juni 2016