



HOCHSCHULE COBURG

Studiengang Informationstechnologie für Unternehmensanwendungen

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

Masterarbeit

vorgelegt von

Julian Kandler

am 02.02.2017

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

Mein Studium fing an im Oktober 2009. Jetzt, fast acht Jahre später, endet es. Zusammenfasst? Es hat Spaß gemacht. Es war die erfüllendste Zeit meines Lebens, in der ich viele Menschen getroffen habe, und auch viele Menschen mein Leben geprägt haben. Zunächst möchte ich Jiayi Li danken. Du hast mich immer dazu motiviert weiter zu machen und niemals aufzugeben.



Wem der kurze Comic gefallen hat, der kann sich diesen und viele weitere unter <https://xkcd.com/> ansehen.

Kurzzusammenfassung

Dieser Masterarbeit hat ihren Forschungsschwerpunkt darin Probleme zu beschreiben und entsprechende Lösungsansätze zu finden, welche die Integration von dynamischen Content aus externen Services in das statischen CMS Umfeld betreffen.

Hierbei treten Probleme bei der Zusammenführung von statischem Content, welcher oft sehr Marketing getrieben ist und von Redakteuren und Autoren gepflegt wird, und dynamischen Inhalten aus Applikation, welche oft im Unternehmensportal angesiedelt sind, auf.

In dieser Arbeit soll untersucht werden wie dynamischer Content aus dem Portalumfeld mittels Einsatz von modernen Webframeworks in den Adobe Experience Manager integriert werden können. Hierbei stellt sich die Herausforderung, dass unterschiedliche Themen wie zum Beispiel Suchmaschinen und Linkvalidierung beachtet und unterstützt werden müssen.

Inhaltsverzeichnis

Inhaltsverzeichnis	iv
Abbildungsverzeichnis	viii
Tabellenverzeichnis	ix
Codebeispielverzeichnis	x
Abkürzungsverzeichnis	x
1 Einleitung	1
1.1 Motivation	1
1.2 Zielsetzung	2
1.3 Aufgaben	2
1.4 Aufbau der Arbeit	3
2 Grundlagen	4
2.1 Begriffsdefinitionen	4
2.2 Technische Grundlagen	6
2.2.1 HTML	6
2.2.2 CSS	7
2.2.3 JavaScript	9
2.2.4 TypeScript	11
2.2.5 Webserver	11
2.2.6 URL	11
2.2.7 Servlet	13
2.2.8 HTTP	13
2.2.9 Ajax	15
2.2.10 XML und JSON	15
2.3 Fachliche Grundlagen	17
2.3.1 Webanwendung	17
2.3.1.1 Serverseitige Webanwendungen	18
2.3.1.2 Clientseitige Webanwendungen	19
2.3.1.3 Single-Page-Webanwendung	20
2.3.2 Framework / Bibliothek	20

2.3.3	MV*-Architekturmuster	21
2.3.4	Content Management System	22
2.3.4.1	Adobe Experience Manager	22
2.3.4.1.1	JCR	23
2.3.4.1.2	CRX	25
2.3.4.1.3	OSGi und Apache Felix	25
2.3.4.1.4	Apache Sling	25
2.3.4.1.5	Autoren-Bedienoberfläche	27
2.3.4.1.6	Administrator Bedienoberfläche	28
2.3.4.1.7	AEM-Anwendung	29
2.3.4.1.8	HTL	30
2.3.4.1.9	Komponenten	30
2.3.4.1.10	Clientlib	31
3	Webframeworks	33
3.1	Werkzeuge und Techniken	33
3.1.1	Node.js	33
3.1.2	Paketverwaltung	33
3.1.3	Automatisierungswerkzeuge	34
3.1.4	Modularisierung und Verwaltung von Abhängigkeiten	34
3.2	Anforderungen	35
3.2.1	Browserunterstützung	35
3.2.2	Integration	36
3.2.3	Indizierung von Suchmaschinen	36
3.2.3.1	Funktionsweise einer Suchmaschine	36
3.2.3.2	Probleme bei Single-Page-Webanwendungen	37
3.2.3.2.1	Nachladen mit JavaScript	37
3.2.3.2.2	Hyperlinks innerhalb der SPA	37
3.2.3.2.3	Lösung durch die pushState-Funktion	38
3.2.4	Lizenz	40
3.3	Auswahl & Bewertung	40
3.3.1	AngularJS	40
3.3.1.1	Datenbindung	40
3.3.1.2	Module	41
3.3.1.3	Service	41
3.3.1.4	Dependency-Injection	42
3.3.1.5	Templates	42
3.3.1.6	Direktiven	42

3.3.1.7	Scopes	42
3.3.2	AngularJS2	43
3.3.2.1	Unterschied zu AngularJS	43
3.3.2.2	Komponenten	44
3.3.2.3	Performance	44
3.3.2.4	Datenbindung	44
3.3.3	Aurelia	45
3.3.3.1	Sprachen	45
3.3.3.2	Templates	45
3.3.3.3	Dependency Injection	45
3.3.3.4	Support	46
3.3.4	React	46
3.3.4.1	Virtueller DOM	46
3.3.4.2	JSX	46
3.3.5	Vergleich	47
4	Analyse des Ist-Zustandes	49
4.1	Ziel	49
4.2	Entwicklungsprozess und bestehende Problematiken	50
4.2.1	IT-Landschaft der Zielpattform	51
4.2.2	Bereitstellen von Web-Ressourcen	51
4.2.3	Konflikte mit anderen Skripten	52
4.3	Anforderungen	52
5	Integration der Webframeworks in AEM	54
5.1	Testumgebung	54
5.2	Zu untersuchende Frameworks	54
5.3	Zu integrierende Webanwendungen	54
5.4	Möglichkeiten der Integration	55
5.4.1	Direkte Integration in das AEM	55
5.4.1.1	Lösungsansatz „Clientlib“	55
5.4.1.1.1	Anpassen zur AJC	56
5.4.1.1.2	Bewertung	57
5.4.1.2	Lösungsansatz „Content Ordner“	57
5.4.1.2.1	Bewertung	58
5.4.2	Integration über einen zusätzlichen Webserver	58
5.4.2.1	Lösungsansatz „Java-Servlet“	58
5.4.2.1.1	Erklärung	59

5.4.2.1.2	Anpassen des Webservers	60
5.4.2.1.3	Anpassen der Webanwendung	61
5.4.2.1.4	Ablauf	61
5.4.2.1.4.1	HTML-Seite aufbereiten	62
5.4.2.1.4.2	Ajax-Anfragen anpassen	63
5.4.2.1.5	Aufbereitungsaufgabe bei unterschiedlichen Frameworks	64
5.4.2.1.6	Bewertung	64
5.4.2.2	Lösungsansatz „JavaScript“	65
5.4.2.2.1	Synchronisation	65
5.4.2.2.2	Bewertung	66
5.4.3	Lösungsansatz „Proxy“	66
5.4.3.1	Erklärung	67
5.4.3.2	Bewertung	68
5.5	Möglichkeiten der Kombination	68
5.5.1	Clientlib und Content-Ordner	68
5.5.1.1	Bewertung	69
5.5.2	Proxy und Java	69
5.5.2.1	Bewertung	70
5.6	Weitere Werkzeuge und Vorgehensprozesse	71
5.6.1	Minimierung	71
5.6.2	Konkatinierung und weitere Laufzeitoptimierung	72
5.6.3	Übertragen von Ressourcen	73
5.6.3.1	WebDAV	73
5.6.3.2	HTTP	73
5.6.3.3	CRX-Paket	73
5.7	Suchmaschinen	74
5.8	Bekannte Konflikte zwischen Skripten	76
5.9	Bewertung der Lösungen	76
6	Zusammenfassung	78
6.1	Fazit	78
6.2	Ausblick	79
	Stichwortverzeichnis	79
	Literaturverzeichnis	80

Abbildungsverzeichnis

1	Beispiel HTML	7
2	HTML erweitert mit CSS	8
3	Vereinfachte Baumstruktur des DOM ohne Attribute	10
4	Kommunikationsbeispiel mit HTTP	13
5	Ladevorgang einer Webseite	14
6	Systematischer Ablauf beim Abruf einer Webseite	17
7	Ein einfaches Beispiel eines Onlinechats	18
8	Kommunikation von Server und Client beim Laden neuer Informationen bei serverseitigen Webanwendungen	18
9	Kommunikation von Server und Client beim Laden neuer Informationen bei clientseitigen Webanwendungen	19
10	MVC-Muster	21
11	MVVM-Muster	22
12	Baumstruktur des JCR	24
13	Auflösung einer Anfrage bei unter Sling, von [Ado16d]	26
14	Autoren-Bedienoberfläche	27
15	CRXDE Lite	29
16	Mögliche Navigationsstruktur einer Webseite	36
17	Mögliche Navigationsstreuung in einer SPA	38
18	Ablauf des Webseitenwechsels in SPAs mit Verwendung der pushState- Funktion	39
19	Ausgangssituation und Ziel	50
20	Integration vom Webserver in den AEM Server mittels einer AEM and JavaScript Component (AEM und JavaScript Komponente, AJC)	59
21	Ablauf bei CORS	60
22	Ungefährer Ablauf der AEM-Komponente auf Java-Basis	62
23	Funktionsweise des Proxy	67
24	Aufbau der Proxy und Java Kombination	70
25	Möglichkeit zum Auflösen von Anfragen unter AEM	76
26	Entscheidungsbaum für eine adäquate Lösung	77

Tabellenverzeichnis

1	Definitionen	6
2	Ausschnitt der Browserunterstützung von ECMAScript [ZAP16]	9
3	Aufbau einer URL	13
4	Möglichkeiten für Dependency Injection	34
5	Browsersupport der Webanwendungen	35
6	Vergleich der vier Frameworks	48
7	Webanwendungen	55
8	Beispiel für die interne Umleitung	75
9	Anforderungserfüllung	77

Codebeispielverzeichnis

1	Beispiel HTML	6
2	Beispiel CSS	7
3	Ein JavaScript Beispiel	10
4	Eine URL	11
5	Ein XML-Beispiel	15
6	JSON für eine einfache Musikdatenbank	16
7	Textuelle Darstellung von JCR	24
8	Exemplarische Darstellung einer AEM Anwendung	29
9	Ein HTL Beispiel	30
10	Exemplarische Darstellung einer Clientlib	31
11	Verwendung einer Clientlib in AEM	31
12	Datenbindung bei AngularJS	40
13	Zu generierender HTML-Code	46
14	Generierung von HTML in React	47
15	Generierung von HTML mit JSX	47
16	Eine Beispiel-AJC mit Clientlib	56
17	Ausgangssituation auf Server B	62
18	Aufbereiteter Quellcode	63
19	Nachladen einer JavaScript-Ressource	65
20	Laden und Synchronisieren von JavaScript-Ressourcen	65
21	Konfiguration des mod_rewrite Modules für den Proxy-Lösungsansatz	67
22	Eine Datei mit curl in das AEM übertragen	73
23	Konfigurationsbeispiel für den Apache Sling Resource Resolver	75

Abkürzungsverzeichnis

AEM	Adobe Experience Manager
Ajax	Asynchronous JavaScript and XML (Asynchrones JavaScript und XML)
AJC	AEM and JavaScript Component (AEM und JavaScript Komponente)
AMC	Adobe Marketing Cloud
AMD	Asynchronous Module Definiton
AoT	Ahead-of-Time (Vorzeitig)
API	Application Programming Interface (Programmierschnittstelle)
CMS	Content Management System (Inhaltsverwaltungssysteme)
CND	Compact Namespace and Node Type Definition
CORS	Cross-Origin Resource Sharing (Herkunftsübergreifende Ressourcenverteilung)
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)
HTTPS	Hypertext Transfer Protocol Secure (Sicheres 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

PHP PHP: Hypertext Preprocessor

REST Representational State Transfer

SCE Strong Contextual Escaping

SOP Same-Origin-Policy (Gleiche-Herkunft-Richtlinie)

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 den 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, die diese optisch, inhaltlich und funktionell 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 Systems (Inhaltsverwaltungssysteme, CMSe) 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 dieses zumeist auf serverseitigen Techniken und Programmiersprachen basiert. 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 wer-

den, wie sich Webanwendungen, die mit einem Webframework erstellt wurden, integrieren lassen.

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 die Sicherstellung, 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 bis jetzt eine AEM-Instanz erweitert wurde. So werden bereits etwaige erste Problematiken erkannt.

Anschließend wird der Versuch unternommen jeweils eine Webanwendung der vier zuvor genannten clientseitigen Webframeworks zu integrieren. Dabei ist es zielführend, Webanwendungen zu verwenden, die möglichst zahlreiche Funktionalitäten des jeweiligen Webframeworks nutzen. Der Integrationsversuch wird abschließend mit verschiedenen Konfigurationen der Webanwendung und des AEM wiederholt. Somit lassen sich weitere Schwierigkeiten mit gewissen Konfigurationen entdecken.

Für die gefundenen Probleme gilt es nun, Lösungsansätze zu erarbeiten und diese innerhalb dieser Arbeit zu erläutern. Sollte unter einer Konfiguration die Integration nicht gelungen sein, so wird für diese eine alternative Herangehensweise benötigt.

Schlussendlich werden alle Formen der Integration und Lösungsansätze gesammelt, dokumentiert und weitestgehend gegenübergestellt. Ziel soll es dabei nicht sein innerhalb dieser

Arbeit die Ergebnisse der Integration zu dokumentieren, sondern das dabei gewonnene Wissen, nämlich wie sich Probleme umgehen lassen und welche Lösungsansätze wann zu empfehlen wären.

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 erarbeiteten 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
Ressource	Eine Ressource ist alles, was von einem Programm geladen und weiter verarbeitet werden kann. Hierzu zählen Dateien, Datenbanken und vieles mehr. Je nach Kontext lässt sich eine Ressource auch genauer spezifizieren, wie z. B. als Web-Ressource oder als Server-Ressource.
Web-Ressource	Web-Ressourcen sind Ressourcen wie z. B. Webseiten, Bilder oder JavaScript, welche über eine eindeutige URL, wie z. B. <i>example.com/index.html</i> , <i>example.com/bilder/flowerpot.png</i> oder <i>example.com/jquery.js</i> , erreichbar sind. Der Begriff URL wird in Unterabschnitt 2.2.6 näher beschrieben. Diese werden über HTTP(S) übertragen, siehe hierzu Unterabschnitt 2.2.8.
Webpräsenz	Eine Webpräsenz ist der Zusammenschluss Web-Ressourcen unter einer Verantwortlichkeit. Diese wird zumeist unter einer Domain, wie zum Beispiel <i>example.com</i> betrieben. Für die Pflege des Inhalts sind zumeist eine oder mehrere Personen verantwortlich. Eine Webpräsenz wird im Englischen auch Website genannt.
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. Eine Webseite wird im Englischen auch Webpage genannt.
HTML-Seite	Eine HTML-Seite im engeren Sinne ist eine Web-Ressource, die dem Browser als HTML-Quellcode geliefert wird. Eine HTML-Seite im weiteren Sinne ist eine Web-Ressource, die dem Browser als HTML-Quellcode ergänzt um CSS- und JavaScript-Code geliefert wird.

(Web-)Server	Ein Webserver ist ein Programm auf einem Rechner, der Web-Ressourcen über das HTTP-Protokoll 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 eine Webpräsenz über das Internet zur Verfügung zu stellen. Siehe auch Unterabschnitt 2.2.5.
Server-Ressourcen	Server-Ressourcen sind alle Ressourcen, auf die ein Webserver zurückgreifen kann. Dies wären zum Beispiel HTML-Dokumente, Konfigurationsdateien oder Datenbanken. Viele der Server-Ressourcen werden dann zu Web-Ressourcen.
Inhalte (Content)	Inhalte, oft auch Content genannt, sind Ressourcen mit einen Informationsgehalt, der z.B. als Text oder als Bild dargestellt ist [MR06, S. 239]. Diese Inhalte können in separaten Dateien oder in anderen Server-Ressourcen eingebettet sein. Um diese leichter zu verwalten lässt sich ein CMS einsetzen.
Webbrowser	Ein Webbrowser, kurz auch Browser, ist ein Computerprogramm, das Web-Ressourcen über das HTTP-Protokoll herunterladen und darstellen kann.
Webanwendung	Eine Webanwendung ist ein Programm, das Web-Ressourcen erzeugt, die im Browser dargestellt werden oder via HTTP-Protokoll heruntergeladen werden können. Zumeist kann ein Besucher einer Webpräsenz interaktiv mit der Webanwendung interagieren. Dabei ist zwischen serverseitigen Webanwendungen und clientseitigen Webanwendungen zu unterscheiden. Bei einer Webpräsenz können alle Web-Ressourcen oder auch nur ein Teil über Webanwendungen erzeugt werden. Für eine detaillierte Beschreibung des Begriffs Webanwendung siehe Unterabschnitt 2.3.1.
App-Ressource	Eine App-Ressource sind alle Ressourcen, die für die korrekte Ausführung einer Webanwendung von Nöten sind. Der Begriff „App“ leidet sich von „Applikation“, die englische Übersetzung für „Anwendung“ ab.
SPA	Sonderform einer clientseitigen Webanwendung, die aus nur einer HTML-Seite im weiteren Sinne besteht.
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 sind mit Entwicklern meist eine oder mehrere Personen gemeint, die eine Webanwendung oder einen Teil einer Webanwendung, wie z.B. ein Framework oder eine Bibliothek für einen Kunden entwickeln.
Kunde	Der Kunde ist der Auftraggeber für die Entwicklung einer Webanwendung.

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 [W3C14b]. 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     
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   
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 Node.js (Unterabschnitt 3.1.1) 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 [Int16]. 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.

Browser \ Version	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 der Organisation Mozilla Foundation. 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.

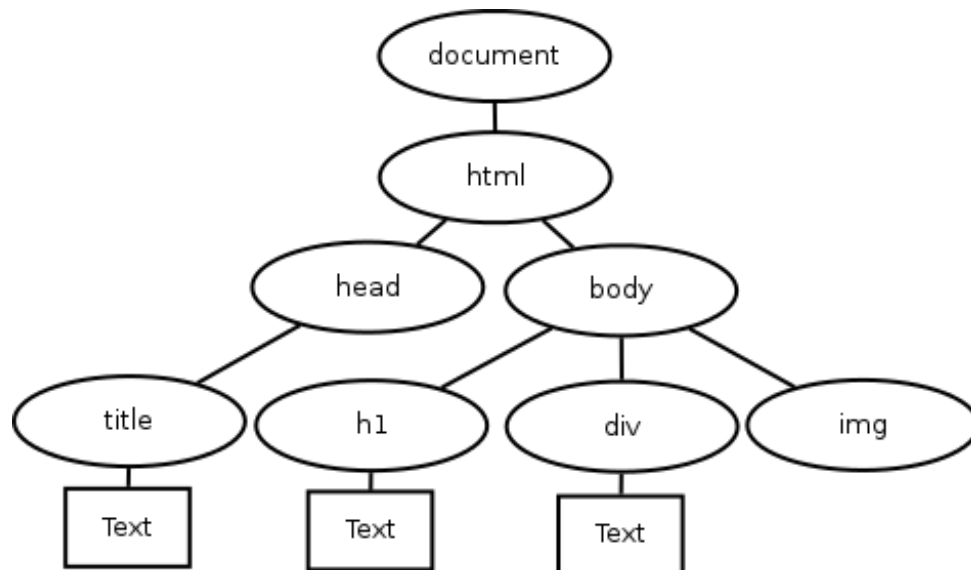


Abbildung 3: Vereinfachte Baumstruktur des DOM ohne Attribute

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 unterstützen 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 2.2.9, mit bei der Standardeinstellung gewisser Browser zu einem Same-Origin-Policy (Gleiche-Herkunft-Richtlinie, SOP)-Fehler führen [SS15, S. 45 f.].

Abhilfe schafft hier der Einsatz eines Webserver. 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 URL

Ein Uniform Resource Locator (URL) identifiziert und lokalisiert eine Ressource. In der Webtechnologie verweist eine URL auf eine Web-Ressource.

Eine URL setzt sich aus verschiedenen Teilen, wie das verwendete Protokoll, den Port und dem Hostname zusammen. Eine URL kann hierbei wie folgend aussehen.

```
1 http://www.example.com:3000/spa/index.html?page=news
```

Codebeispiel 4: Eine URL

Die URL aus Codebeispiel 4 in Tabelle 3 erklärt.

http	www	example.com	3000	spa/index.html	page=news
	Subdomain	Domain			
Schema/Protokoll	Hostname		Port	Pfad	Parameter

Tabelle 3: Aufbau einer URL

2.2.7 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 Clients 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.

2.2.8 HTTP

Beim Hypertext Transfer Protocol (Hypertext-Übertragungsprotokoll, HTTP) handelt es sich um ein zustandsloses (Netzwerk-)Protokoll zur Datenübertragung, dessen Haupteinsatzgebiet im Internet ist. Ein Protokoll definiert, wie Nachrichten in einem verteilten System untereinander verschickt werden. Eine Nachricht besteht in der Regel aus Meta-informationen, welche sich im Header (eng. für Kopf) befinden, und den Nutzdaten.

Über HTTP werden Web-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.

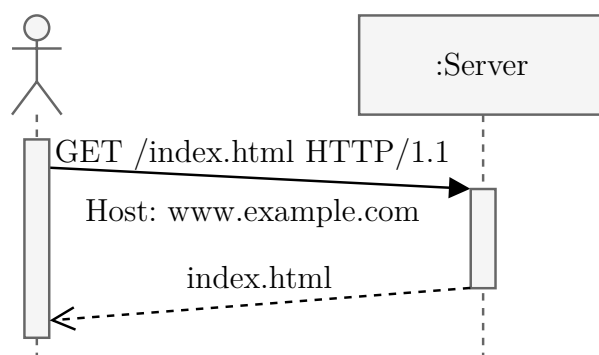


Abbildung 4: Kommunikationsbeispiel mit HTTP

Der Client fordert vom Server, dem der Hostname `www.example.com` zugewiesen ist, die gewünschte Web-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 Web-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.

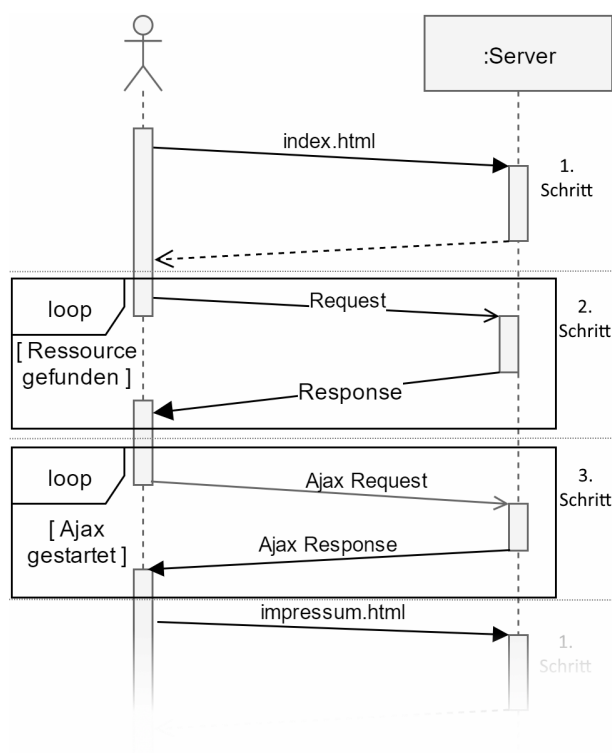


Abbildung 5: Ladevorgang einer Webseite

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 Web-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 erscheinende Bibliothek, um Abhängigkeitsfehler zu vermeiden. Weitere Web-Ressourcen kann der Client nun über Ajax anfordern, siehe hierzu Unterabschnitt 2.2.9. Möchte der Benutzer nun die Webseite 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.

- 1. HTML-Seite:** Die darzustellende HTML-Seite wird vom Server angefordert.
- 2. Web-Ressourcen:** Der Quellcode der HTML-Seite wird nach weiteren Web-Ressourcen, wie Grafiken, Skripte oder Stylesheets, durchsucht und diese werden geladen.
- 3. Ajax:** Ajax Anfragen werden ausgeführt.

Eine Erweiterung von HTTP wäre das Hypertext Transfer Protocol Secure (Sicheres Hypertext-Übertragungsprotokoll, HTTPS). Dieses ermöglicht zusätzlich eine verschlüsselte Kommunikation.

2.2.9 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 ein HTTP-Request 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.

2.2.10 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 5 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 5: 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 6 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 6: JSON für eine einfache Musikdatenbank

Zu beachten ist, dass Codebeispiel 6 keine 1:1-Umsetzung von Codebeispiel 5 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 dienen 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.

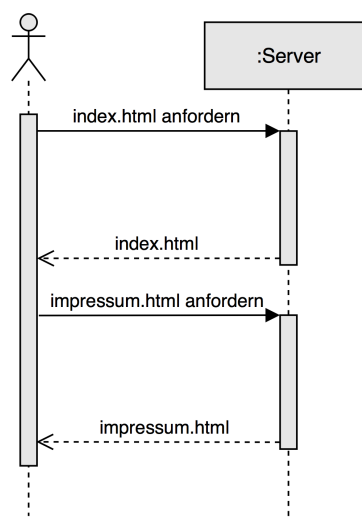


Abbildung 6: Systematischer Ablauf beim Abruf einer Webseite

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 Web-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.

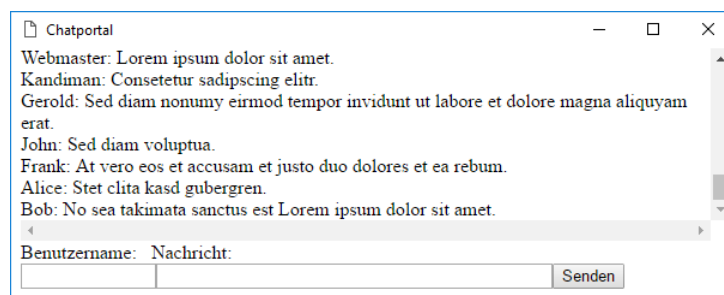


Abbildung 7: Ein einfaches Beispiel eines Onlinechats

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

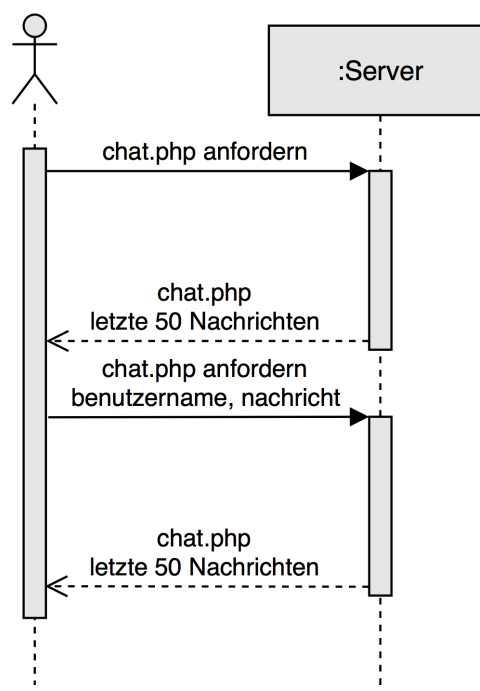


Abbildung 8: Kommunikation von Server und Client beim Laden neuer Informationen bei serverseitigen Webanwendungen

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 befüllen 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.

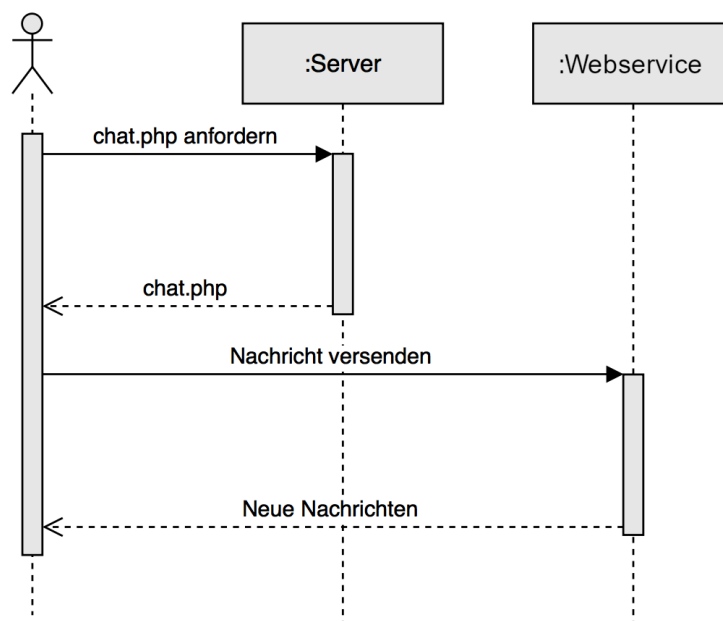


Abbildung 9: Kommunikation von Server und Client beim Laden neuer Informationen bei clientseitigen Webanwendungen

Ein Webservice ist hier ein Teil des Webserver, 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, mit 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.

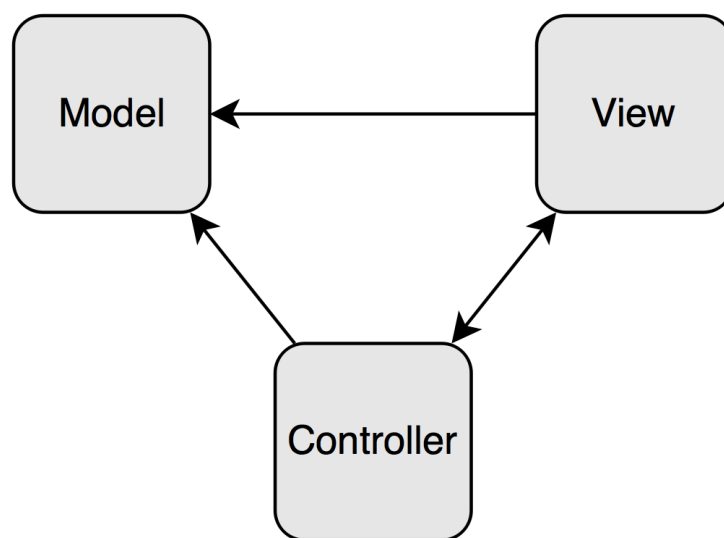


Abbildung 10: MVC-Muster

Die Model-Komponente beinhaltet die darzustellenden Daten. In der View-Komponente wird beschrieben, wie die Daten darzustellen sind. Die Controller-Komponente reagiert 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.

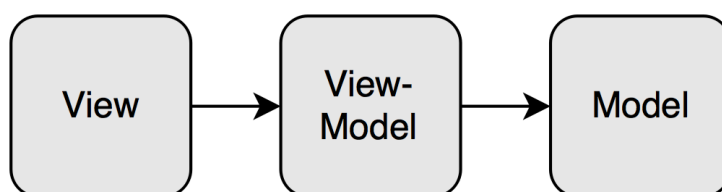


Abbildung 11: MVVM-Muster

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

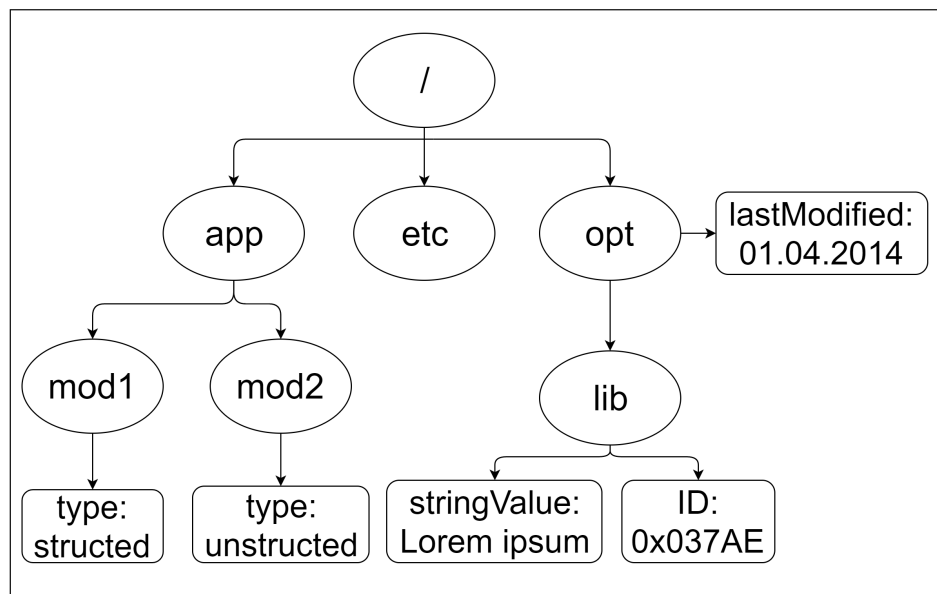
Namen von selbigem weiterentwickelt wird. Dieses ist in der Programmiersprache Java entwickelt, welche sich durch ihre Plattformunabhängigkeit auszeichnet. AEM gehört zu der Adobe Marketing Cloud (AMC) und ist mit den anderen darin enthaltenen Produkten kompatibel. Zu diesen gehören unter anderem Werkzeuge 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 veröffentlicht. 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 Repository 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 12 dienen.



Legende:

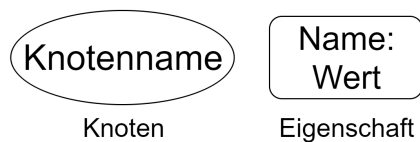


Abbildung 12: Baumstruktur des JCR

Eine JCR Struktur lässt sich textuell mit der Compact Namespace and Node Type Definition (CND) darstellen, welche von der Apache Software Foundation definiert wird [Fou17a]. 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 7: 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 Server-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 [SS15, S. 82 f.].

- Jede Web-Ressource ist über einen eindeutigen URL zu erreichen.
- Auf Web-Ressourcen können verschiedene Methoden angewandt werden. Bei HTTP wären dies z. B. POST, GET DELETE oder PUT. Zu beachten ist, dass REST jedoch kein HTTP voraussetzt.
- Die Darstellung von Web-Ressourcen kann in verschiedenen Formaten erfolgen.
- REST ist immer zustandslos.

Abbildung 13 beschreibt, wie Apache Sling einen Aufruf auflöst.

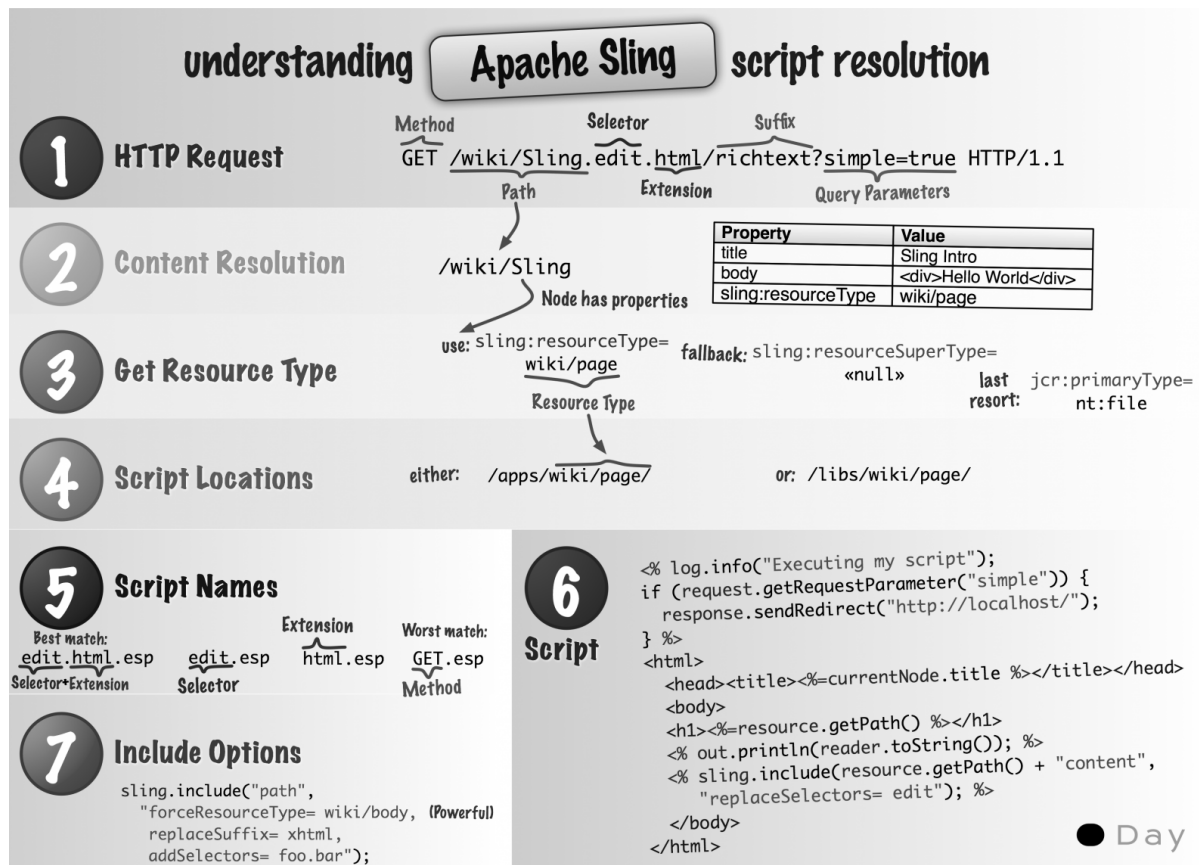


Abbildung 13: Auflösung einer Anfrage bei unter Sling, von [Ado16d]

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, eingefügt 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.

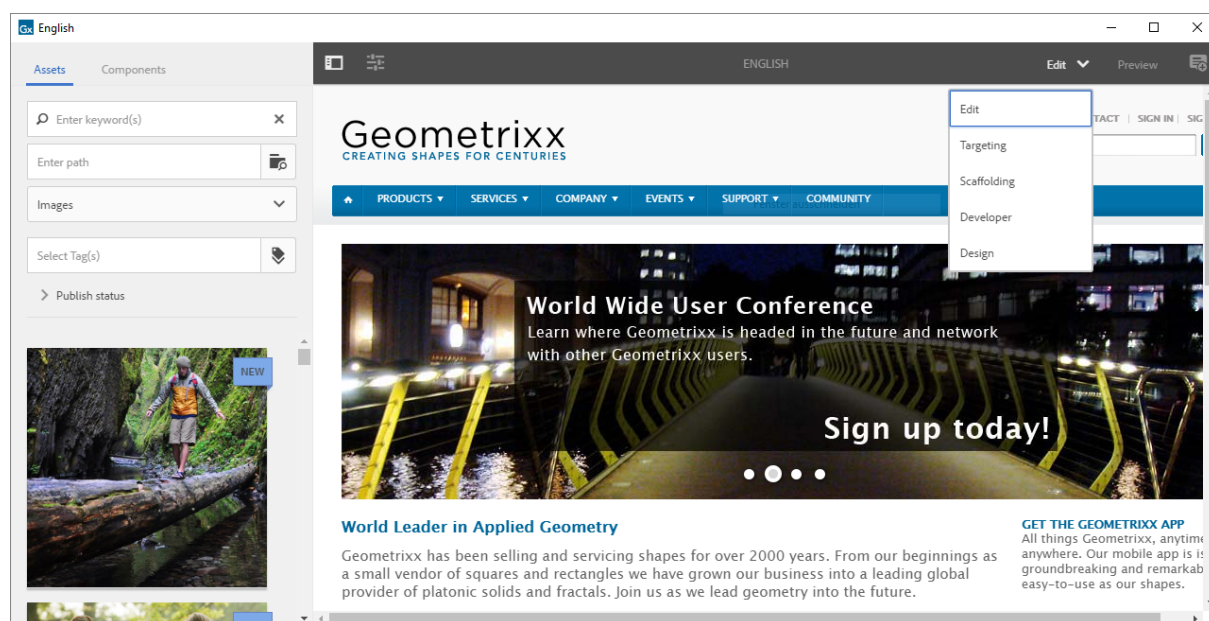


Abbildung 14: Autoren-Bedienoberfläche

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.

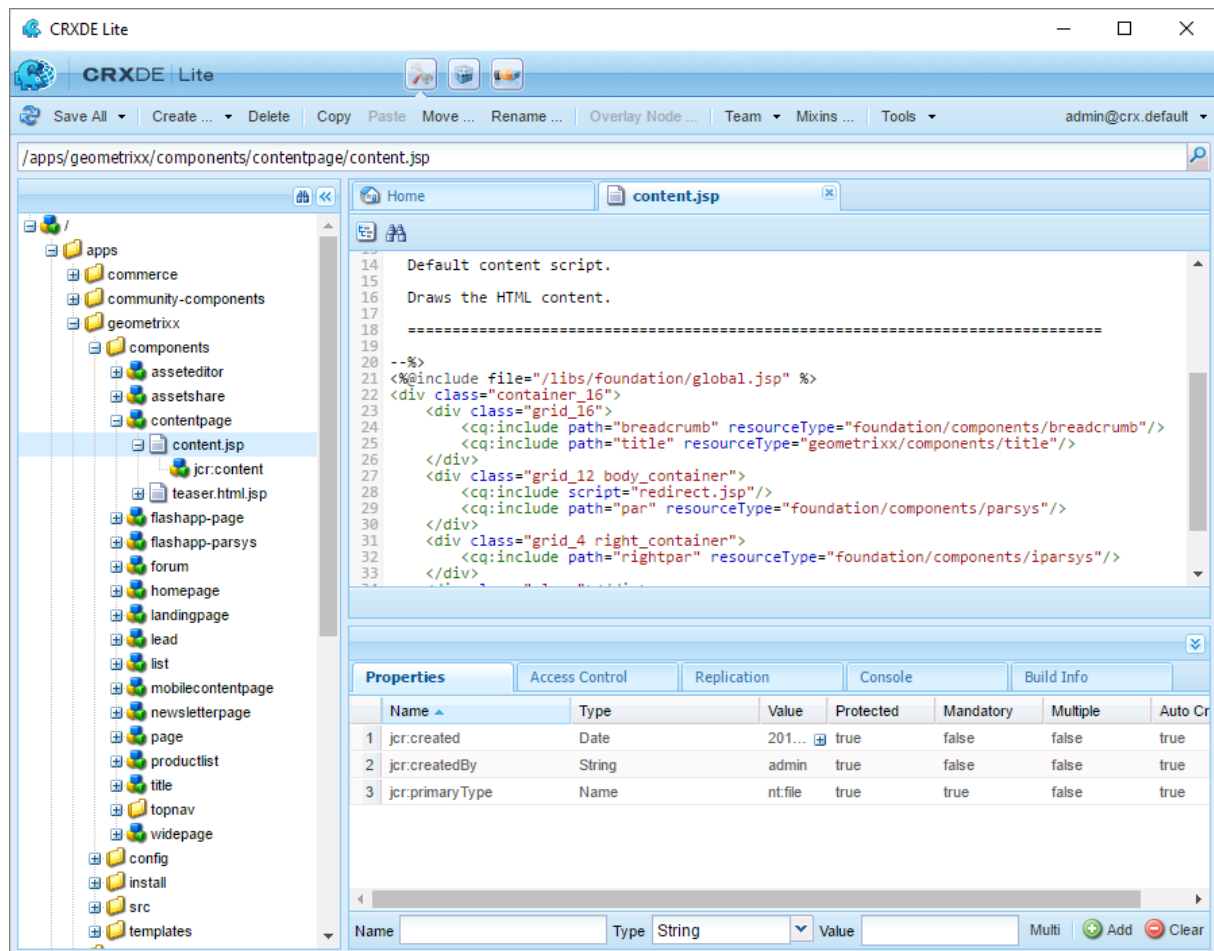


Abbildung 15: CRXDE Lite

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 8 zu sehen.

```

1 /apps/helloworld
2   + components
3   + templates

```

Codebeispiel 8: 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 9 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 9: 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 in der Regel innerhalb einer Komponente oder unter `/etc/clientlibs` befinden. Codebeispiel 10 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 identifiziert wird.
4  - dependencies (String[]) = [angularjs] # Mögliche Abhã
    ↳ 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 10: Exemplarische Darstellung einer Clientlib

Die Ordernamen 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 11 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 11: 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 der Clientlib 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 Paketverwaltungsprogramme 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 Definition (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 4 zu entnehmen.

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

Tabelle 4: 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 genutz-

ten Schnittstelle abhängig. Die verwendete Bibliothek löst die Abhängigkeiten auf und übergibt den Modulen Instanzen ihrer benötigten Module.

In der Regel werden alle benötigten Module zu Anfang, also beim Aufruf der Webseite, vergleiche den ersten Schritt von Abbildung 5, geladen. Somit sind alle Ressourcen zu Beginn vorhanden. Dieser Ansatz wird auch Ahead-of-Time (Vorzeitig, AoT) genannt. Manche Bibliotheken beherrschen zusätzlich den Just-in-Time (Rechtzeitig, JiT)-Ansatz. Hierbei werden Module erst dann geladen, sobald diese benötigt werden.

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 5 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 5: 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.

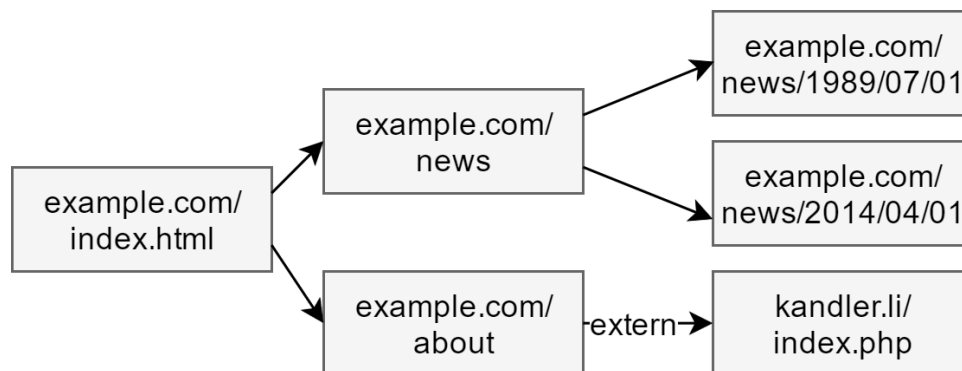


Abbildung 16: Mögliche Navigationsstruktur einer Webseite

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 SPA

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.

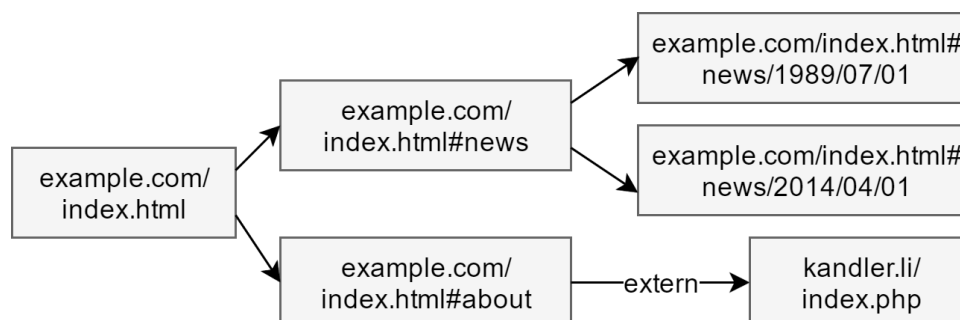


Abbildung 17: Mögliche Navigationsstruktur in einer SPA

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 URLs wie zum Beispiel `http://www.example.com/?_escaped_fragment_=about` führt. Der Crawler erwartet nun vom Server unter dieser URL 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.

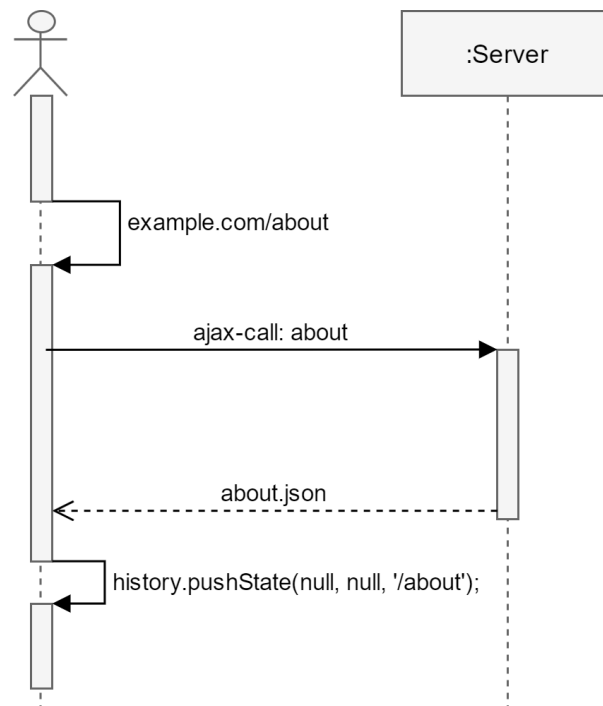


Abbildung 18: Ablauf des Webseitenwechsels in SPAs mit Verwendung der pushState-Funktion

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 URL 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-URL 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 URL */news* bzw. */about* erkennen und den entsprechenden Inhalt darstellen. Durch diesen Trick kann nun der Google Crawler jede Webseite einer SPA durch eine eindeutige URL 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-Klausen-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 12.

```
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 12: 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 Web-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 erweitern 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 12 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 12, Zeile 9, nicht mehr *ng-model*=“username“, lauten, sondern *[(ngModel)]*=“username“.

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

Sytleheets: 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 13: 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 14: Generierung von HTML in React

Das Codebeispiel 14 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 15: Generierung von HTML mit JSX

Somit wird durch den Einsatz von JSX das Codebeispiel 15 im Hintergrund in Codebeispiel 14 umgewandelt.

3.3.5 Vergleich

Folgende Tabelle 6 zeigt kurz zusammengefasst die wichtigsten Eigenschaft der zuvor genannten Webframeworks im Vergleich.

Eig. \ Webfr.	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 6: Vergleich der vier Frameworks

4 Analyse des Ist-Zustandes

Durch den Einsatz von AEM lassen sich bereits umfangreiche Webpräsenzen entwickeln. Dabei besteht jede hiermit erstellte Webseite erfahrungsgemäß aus einem statischen HTML-Rahmen, in dem sich mehreren wiederverwertbaren Komponenten befinden. Im Folgenden wird davon ausgegangen, dass bereits eine Webpräsenz innerhalb einer AEM-Instanz angesiedelt ist. Es besteht der Wunsch, diese durch weitere AEM-Komponenten, wie jene in Absatz 2.3.4.1.9 beschrieben, zu erweitern.

Besagte Komponenten basieren jedoch in erster Linie zunächst auf serverseitigen Technologien. Bei der Interaktion mit dem Besucher resultiert dies zu dem Ergebnis, 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. Er ist es gerade von mobilen Geräten wie Tablets und Smartphones gewohnt, dass deren Anwendungen 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.

4.1 Ziel

Der Gedanke ist hier, die gewünschten Eigenschaften von clientseitigen und serverseitigen Webframeworks zu kombinieren. Statische Inhalte wie Bilder und feste Texte werden über das AEM gepflegt. Sich häufig ändernde Inhalte, wie Datenbankinhalte oder Informationen von teils externen REST-Schnittstellen, lassen sich über besagte clientseitige Webanwendungen zur Laufzeit der HTML-Seite nachladen und anzeigen. Die clientseitige Webanwendung wird den Autoren in Form einer AEM-Komponente zur Verfügung gestellt, damit diese leicht über die Autoren-Bedienoberfläche zu integrieren ist. Ein AEM-Komponente, welche primär serverseitige Technologien verwendet, wird hier durch eine clientseitige Webanwendung erweitert. Somit wird aus einer statischen Webseite eine Webseite mit einer eingebetteten clientseitigen Webanwendung, wie in Abbildung 19. Das Ziel ist es, Webanwendungen, die mit clientseitigen Webframeworks entwickelt wurden, in Webseiten, die mit AEM entwickelt wurden, zu integrieren.

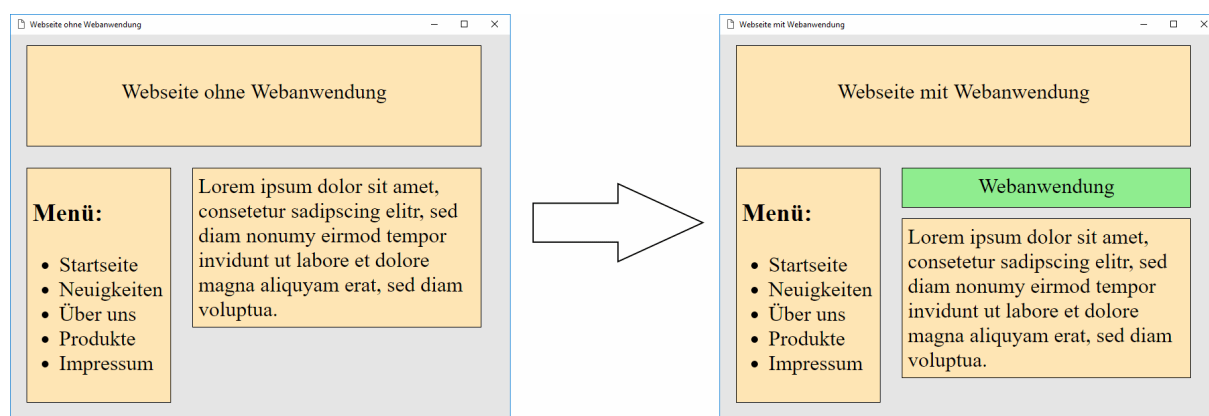


Abbildung 19: Ausgangssituation und Ziel

Zu beachten ist, dass in der Abbildung und im Folgenden der Begriff „Webanwendung“ kurz für eine „clientseitige Webanwendung“ steht. Weiterhin wird an dieser Stelle der Begriff AJC definiert. Eine AJC ist eine AEM-Komponente, die App-Ressourcen zur Verfügung stellt und damit eine clientseitige Webanwendung abstrahiert.

4.2 Entwicklungsprozess und bestehende Problematiken

Die Integration soll vorzugsweise in Form einer AJC erfolgen, um die Positionierung der Webanwendung innerhalb einer Webseite und deren Konfiguration für Autoren zu vereinfachen. Wird diese in eine Webseite eingebunden, sollen ohne weiteres zutun alle benötigten App-Ressourcen geladen werden und es wird gewährleistet, dass die clientseitige Webanwendung in der Webseite wie vorgesehen dargestellt wird. Der Entwicklungsprozess von einer Webanwendung in eine AJC ist hierbei wie folgt vorgegeben.

- 1. Entwicklung der Webanwendung** Zunächst wird die Webanwendung innerhalb einer lokalen Entwicklungsumgebung außerhalb einer AEM-Instanz entwickelt. Als Webserver dient zum Beispiel ein Apache HTTP Server oder ein Node.js Server.
- 2. Entwicklung der AJC** Nachdem die Webanwendung wie gewünscht funktioniert, wird diese an eine AJC angepasst. Dies geschieht jedoch nicht auf der Zielplattform, sondern innerhalb einer separaten AEM-Instanz, welche eigens für Entwicklungszwecke läuft.
- 3. Auslieferung der AJC** Sofern die AJC fertiggestellt ist, wird diese an den Kunden ausgeliefert und schlussendlich von ihm produktiv auf die Zielplattform gestellt.

Beim zweiten und dritten Schritt, also der Transformation einer Webanwendung in eine AJC und den Versuch der Integration können allerdings verschiedene Problematiken auftreten. Diese entstammen den Eigenschaften von AEM und den Webframeworks, resultieren aber auch durch gesetzte Rahmenbedingen und Restriktionen, welche die IT-Landschaft der Zielplattform und Integration betreffen.

4.2.1 IT-Landschaft der Zielplattform

Zumeist weisen die unterschiedlichen Zielplattformen, also jene Plattformen, auf denen 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 die WebDAV-Schnittstelle, deaktiviert sind. Eine weitere mögliche Rahmenbedingung kann sein, dass die App-Ressourcen der Webanwendung sich nicht im AEM, sondern auf einem 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 der gesetzten Rahmenbedingungen als mögliche Lösung gar ausgeschlossen.

4.2.2 Bereitstellen von Web-Ressourcen

Eine weitere Problematik kann das Bereitstellen der Web-Ressourcen der Webanwendung darstellen. 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 dem 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 erfordern Apache Sling und somit auch AEM ein fundamental anderes Programmierparadigma im Vergleich zu anderen Frameworks für die Erstellung von serverseitigen Webanwendungen. Es wird der Fokus stark 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 in das AEM meistens abgeändert werden muss. 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.

4.2.3 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 werden für AEM weitere JavaScript-Dateien geladen, welche es zum Beispiel 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 unterschiedliche 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 findet. 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.

4.3 Anforderungen

Die integrierte Webanwendung, bzw. die AJC müssen gewisse Anforderungen erfüllen.

- A1: Die Webanwendung soll als AJC in die Webseite integriert werden.
- A2: Die AJC soll sich über die Autoren-Bedienoberfläche innerhalb einer Webseite platzieren lassen.
- A3: Die AJC soll sich über die Autoren-Bedienoberfläche, sofern erforderlich, konfigurieren lassen.
- A4: Die AJC soll innerhalb der Autoren-Bedienoberfläche genau wie im Webbrowser des Besuchers dargestellt und bedienbar sein.

A5: Die AJC wird in verschiedenen Formen benötigt, die sich in der Art der Integration unterscheiden.

A5.1: Erstellung einer AJC als vollständige AEM-Komponente.

A5.2: Erstellung einer AJC und Laden von App-Ressourcen mit Java.

A5.3: Erstellung einer AJC und Laden von App-Ressourcen mit JavaScript.

A6: Die Webanwendung soll vor der Integration optimiert werden.

A7: Die Webanwendung soll von gängigen Suchmaschinen korrekt indiziert werden.

A8: Es soll die Möglichkeit geben Anfragen vom Server zu manipulieren.

A9: Es sollen keine Konflikte zwischen Skripten entstehen.

A10: Hyperlinks und Referenzen sollen korrekt sein.

5 Integration der Webframeworks in AEM

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

5.1 Testumgebung

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) in der Version 1.8.0_111-b14.

5.2 Zu untersuchende Frameworks

Im Folgenden soll der Fokus auf die vier Webframeworks aus Abschnitt 3.3 gelegt werden. 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 Entwicklern des jeweiligen Webframeworks und gebrauchen bereits zahlreiche der jeweils verfügbaren Funktionalitäten. Der Quellcode liegt als eigens eingerichtetes GIT Respository zur Einsicht vor. Eine Übersicht der Webframeworks bietet Tabelle 5.3.

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 1.1.0	Quick Start [Eis17]	[Kan16c]
React 15.2.0	Redux Tetris [Lug16]	[Kan16d]

Tabelle 7: Webanwendungen

Im Folgenden wird vermehrt der Begriff „die Webanwendung“ verwendet. Hiermit ist eine beliebige der vier genannten Webanwendungen genannt. Die Aussage „Es wird versucht eine Webanwendung in das AEM zu integrieren“ bedeutet somit, dass nacheinander versucht wird die vier in genannten Webanwendungen in ein AEM zu integrieren.

5.4 Möglichkeiten der Integration

Abhängig von verschiedenen Faktoren wurden unterschiedliche Möglichkeiten erarbeitet, um eine Webanwendung in eine AEM-Instanz als AJC zu integrieren. Neben dem zugrunde liegenden clientseitigen Webframework ist der wichtigste Entscheidungsfaktor dafür, welche Lösung genutzt werden kann, die IT-Landschaft der Ziellplattform. Je nach dessen Konfiguration bieten sich gewisse Lösungen mehr an oder sind auch als mögliche Lösung von vornherein ausgeschlossen. Auch gewisse Wünsche des Kunden, wie die Bereitstellung der App-Ressourcen auf einen separaten Webserver wird berücksichtigt werden.

5.4.1 Direkte Integration in das AEM

Innerhalb dieses Abschnittes sollen Lösungsansätze dargestellt werden, welche die direkte Integration in die JCR-Struktur betreffen. Das bedeutet, dass alle benötigten App-Ressourcen in die JCR abgelegt werden. Somit sind alle Ressourcen, die für die Ausführung der Webanwendung vonnöten sind, über das AEM erreichbar.

Um die Bereitstellung der App-Ressourcen für die Webanwendung zu ermöglichen, wurden zwei Lösungsansätze erarbeitet. In der ersten Lösung werden alle App-Ressourcen in eine Clientlib gepackt. Die zweite sieht als Lösung vor, diese unter dem Content-Ordner abzulegen.

5.4.1.1 Lösungsansatz „Clientlib“

Der erste Lösungsansatz beschäftigt sich mit der Realisierung als Clientlib.

5.4.1.1.1 Anpassen zur AJC

Zunächst empfiehlt es sich, die App-Ressourcen vor der Anpassung zur AJC aufzubereiten. Gerade das Konkatenieren der Ressourcen erleichtert die spätere Arbeit ungemein. Dieser Vorgang wird in Unterabschnitt 5.6.2 erklärt. Auch sollte ab hier auf den AoT-Ansatz verzichtet und dafür der JiT-Ansatz bevorzugt werden. Beim AoT werden die Module in jeweils einzelne Dateien aufgeteilt, wohingegen bei JiT die Module zu einer Datei zusammengeschlossen werden können.

Codebeispiel 16 zeigt, wie sich eine AJC mit Clientlib zusammensetzen könnte.

```
1  /apps/integration/components/
2    + aurelia-app
3      + aurelia-app.html
4      + cq:editConfig
5        - cq:actions (String[]) = [edit, -, delete, insert]
6
7  /etc/design/integration/aurelia-example/
8    + clientlib
9      - jcr:primaryType (Name) = cq:ClientLibrary
10     - catagories (String[]) = integration.aurelia.example
11     - dependencies (String[]) = aurelia
12     + css
13       + style.css
14     + css.txt
15     + javascript
16     + app.aurelia.js
17     + js.txt
18     + resources
19     + templates
20       + app.html
21     + images
22       + logo.png
23
24  /lib/clientlibs/
25    + aurelia
26      + categories (Sting[]) = aurelia
27      + javascript
28        + aurelia.min.js
29      + js.txt
```

Codebeispiel 16: Eine Beispiel-AJC mit Clientlib

Das gezeigte Beispiel stellt eine Aurelia-Anwendung dar und ist in drei Teile aufgeteilt. Der Knoten ab Zeile 24 beinhaltet die Ressourcen für das Webframework, hier Aurelia.

Wichtig ist dabei, diese mindestens einer entsprechenden Kategorie, wie in Zeile 26 zu sehen, zuzuordnen.

Alle weiteren App-Ressourcen werden unter den Knoten ab Zeile 7 abgelegt. In Zeile 11 werden die benötigten Abhängigkeiten zu anderen Clientlibs eingetragen. Diese werden anhand der jeweiligen Kategorie aufgelöst, wie diese im Beispiel in Zeile 26 eingetragen wurde. Weitere App-Ressourcen wie HTML-Templates oder Bilder lassen sich auch unter diesen Knoten deponieren.

Der letzte Knoten, gezeigt in Zeile 1 bis Zeile 5, stellt die AJC dar und beinhaltet in `relia-app.html` den Einstieg für die Webanwendung. Hier wird weiterhin der Aufruf für die Clientlib analog zum Codebeispiel 11 ausgelöst. Der Knoten in Zeile 4 dient dazu, dass die Komponente in der Autoren-Bedienoberfläche genutzt und an ihre entsprechende Stelle platziert werden kann.

Um das Erstellen der beiden Clientlibs zu automatisieren, lässt sich die Grunt-Erweiterung „grunt-aem-clientlib-generator“ [wcm17] nutzen. Diese lässt sich dahingehend konfigurieren, die beiden Ordner „css“ und „javascript“ und die beiden Dateien „css.txt“ und „js.txt“ automatisiert zu erstellen.

5.4.1.1.2 Bewertung

Die Lösung als Clientlib ist wie folgt zu bewerten.

Vorteile:

- Lässt sich gut als CRX-Paket exportieren.
- Alles in einem System.
- Konfiguration über Autoren-Bedienoberfläche möglich.

Nachteile:

- Bestehende Verzeichnisstruktur muss abgeändert werden.
- Relative Pfade müssen angepasst werden.
- Wird etwas an der Webanwendung geändert, so muss diese auch in der AJC geändert werden.

5.4.1.2 Lösungsansatz „Content Ordner“

Eine simple Lösung wäre es, die Webanwendung unter den Content-Ordner abzulegen. Jeglicher Inhalt, der unter `/content` abgelegt wird, ist direkt für den Client aufrufbar. Beispielsweise wäre unter der Standardkonfiguration von AEM die Datei `/content/myapp/index.html` unter der URL `http://<Server-Hostname>:<AEM-Port>/content/myapp/index.html` erreichbar. Somit wäre es möglich, die Verzeichnisstruktur der Webanwendung 1:1 beizubehalten, sofern die komplette Webanwendung unter `/content`

t abgelegt wird. Dieser Lösungsansatz verzichtet jedoch darauf, die Webanwendung als AJC zu realisieren, was deren freie Positionierung und Konfigurierung über die Autoren-Bedienoberfläche ausschließt. Der Lösungsansatz „Content Ordner“ empfiehlt sich somit nur für Webanwendungen, welche eine gesamte Webseite des AEM ausmachen und keine Konfiguration benötigen.

5.4.1.2.1 Bewertung

Die Lösung mit dem Content Ordner ist wie folgt zu bewerten.

Vorteile:

- Beibehalten der Verzeichnisstruktur.
- Schnelle Integration.
- AoT-Ansatz möglich

Nachteile:

- Keine AJC, somit auch kein einbetten in eine bestehende Webseite über die Autoren-Bedienoberfläche des AEM möglich.
- Keine Konfiguration über die Autoren-Bedienoberfläche möglich.

5.4.2 Integration über einen zusätzlichen Webserver

Neben der Lagerung der App-Ressourcen in dem JCR lässt sich diese ebenfalls auf einen zusätzlichen Webserver ablegen und über eine AEM-Komponente in eine Webseite des AEM integrieren.

5.4.2.1 Lösungsansatz „Java-Servlet“

Hier ist die Grundidee, dass sich die App-Ressourcen nicht im JCR des AEM befinden, sondern auf einem separaten Webserver vorliegen. Ziel ist es, dass die Webanwendung unabhängig auf dem separaten Webserver zum Testen lauffähig ist. Über eine AEM-Komponente wird diese in das AEM importiert. Die AEM-Komponente bezieht die App-Ressourcen vom Webserver und passt diese ggf. an, so dass die Webanwendung korrekt in der Webseite des AEM dargestellt wird, wie in Abbildung 20 illustriert wird. Da auch diese AEM-Komponente eine Webanwendung abstrahiert, wird diese im Folgenden ebenfalls als AJC bezeichnet.

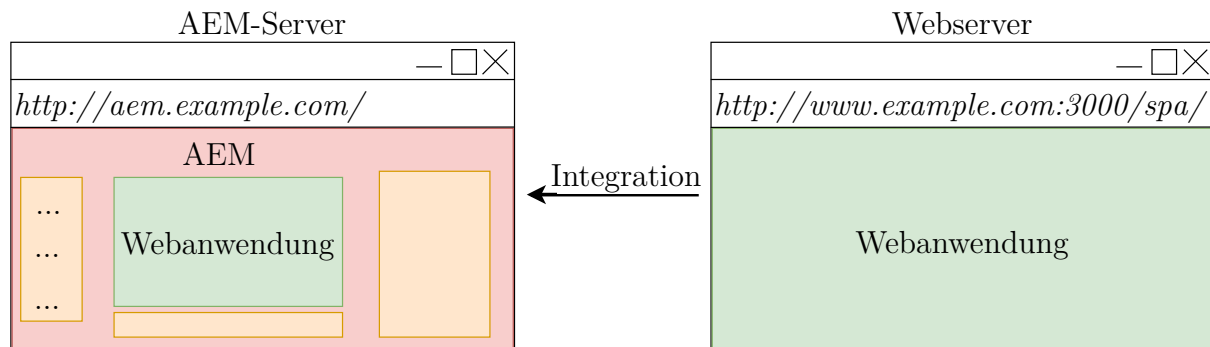


Abbildung 20: Integration vom Webserver in den AEM Server mittels einer AJC

Dabei werden die App-Ressourcen zur Laufzeit der AJC, also wenn die Webseite geladen wird, jeweils erneut vom Webserver geladen. Ein Ablegen in das JCR ist nicht vorgesehen. Ein temporäres Zwischenspeichern (Cachen) wäre von Grund auf denkbar ist, wird innerhalb dieser Arbeit jedoch nicht weiter behandelt.

Im Folgenden wird der separate Webserver, auf dem sich die App-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.

5.4.2.1.1 Erklärung

In der Regel erfolgt das Laden einer Webanwendung genau wie bei einer Webseite, wie in Unterabschnitt 2.2.8 beschrieben, in drei Schritten. Zunächst wird die HTML-Seite angefordert, anschließend wird der Quellcode der HTML-Seite nach weiteren App-Ressourcen durchsucht und auch diese werden geladen. Weitere App-Ressourcen werden nun über Ajax angefordert und vom Server retourniert. Es wird davon ausgegangen, dass diese Schritte unter `http://www.example.com:3000/spa/` korrekt durchlaufen werden und somit die Webanwendung wie gewünscht im Browser erscheint.

Nun ist es Aufgabe der AJC, die App-Ressourcen so aufzurufen und zu bearbeiten, dass die Webanwendung korrekt in einer Webseite des AEM dargestellt wird. Dafür müssen die HTML-Seite, die Ajax-Anfragen und ggf. auch weitere App-Ressourcen manipuliert werden. Die AJC dient somit als Bindeglied zwischen AEM und dem Webserver.

Zuvor muss jedoch der Webserver und die Webanwendung gewissen Anpassungen unterliegen. Diese werden in Absatz 5.4.2.1.2 und Absatz 5.4.2.1.3 beschrieben.

5.4.2.1.2 Anpassen des Webserver

Gängige Webbrowser verbieten, dass JavaScript und somit auch Ajax auf Web-Ressourcen anderer Webserver zugreifen dürfen, als dem, unter dem das Skript gerade ausgeführt wird. Grund dafür ist das Sicherheitskonzept der Same-Origin-Policy (Gleiche-Herkunft-Richtlinie, SOP). Die Herkunft (eng. Origin) setzt sich hierbei aus dem verwendeten Protokoll, Hostname und Port der URL zusammen. Unterscheidet sich die Herkunft von der Webseite, von der aus die Anfrage gestartet wurde, und von der angefragten Web-Ressource, so wird die Anfrage vom Webbrowser geblockt. Im zuvor beschriebenen Beispiel würden sich `http://aem.example.com/` und `http://www.example.com:3000/spa/` in dem Hostnamen (`www.example.com` anstelle von `aem.example.com`) und dem Port (`3000` anstelle vom Standardport für HTTP `80`) unterscheiden. Eine Ajax-Anfragen vom AEM-Server an den Webserver würde einen Fehler erzeugen.

Damit der Zugriff doch funktioniert, lässt sich der von W3C definierte Mechanismus des Cross-Origin Resource Sharing (Herkunftsübergreifende Ressourcenverteilung, CORS) anwenden.

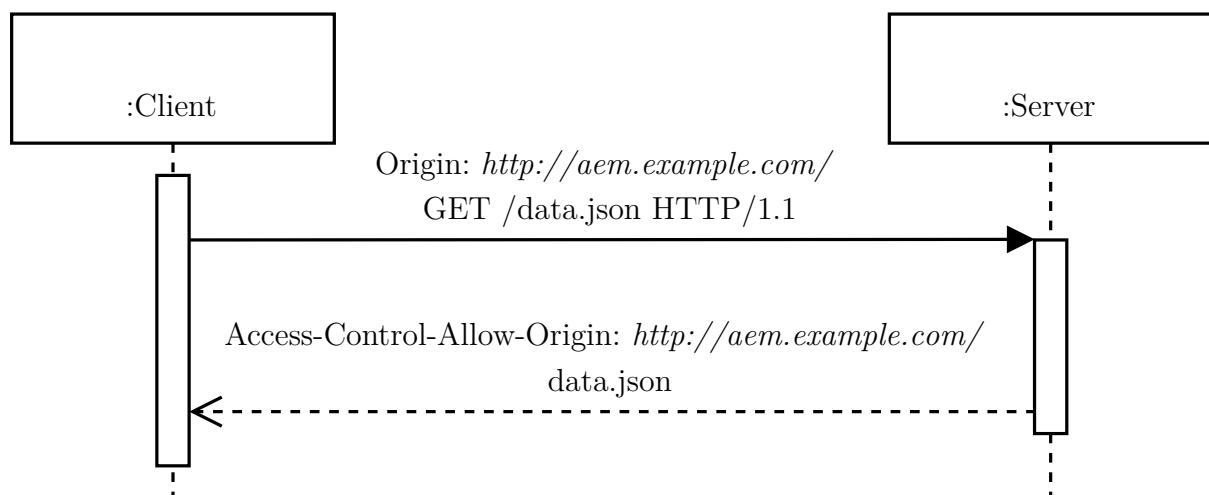


Abbildung 21: Ablauf bei CORS

Wie in Abbildung 21 sendet der Browser des Clients bei seiner Anfrage seine Herkunft (Origin) mit. Am Beispiel hier wäre die Herkunft die URL der Webseite, von der die Anfrage ausgeführt wird, also `http://aem.example.com/`. Der Server antwortet mit einer Liste von erlaubten Hostnamen und der angeforderten Web-Ressource. Der Browser überprüft nun, ob sich die Herkunft der Ajax-Anfrage in der Liste befindet. Im Erfolgsfall wird die Ajax-Anfrage erfolgreich beendet, ansonsten wird eine Fehlermeldung erzeugt.

Neben der gezeigten Variante lässt sich CORS auch mit anderen Zugriffsregeln verwenden. Durch `Access-Control-Allow-Origin:*` werden alle Anfragen genehmigt, durch

`Access-Control-Allow-Methods:GET` werden alle mit der Zugriffsmethode GET gestellten Anfragen erfolgreich ausgeführt [W3C14a].

Die Konfiguration der erlaubten Herkünfte ist Webserverspezifisch und in der jeweiligen Dokumentation nachzulesen. Für Apache kann hier das Modul „`mod_headers`“ [Fou17b] genutzt werden.

5.4.2.1.3 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 Ladevorgang zu einer Strong Contextual Escaping (SCE)-Fehlermeldung kommen. 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?

5.4.2.1.4 Ablauf

Der ungefähre Ablauf ist Abbildung 22 zu entnehmen. Dieser besteht grob zusammengefasst aus zwei Schritten. Der erste Schritt wird im folgenden Unterabsatz 5.4.2.1.4.1, der zweite Schritt im darauf folgenden Unterabsatz 5.4.2.1.4.2 erklärt.

Die beiden Java-Ressourcen `AppBuilder.java` und `GetData.java` lassen sich mit einem OSGi-Bundle für das AEM bereitstellen. Die JavaScript-Ressource `ajaxdirect.js` lässt sich in einer AJC in Verbindung mit einer Clientlib ausliefern. Bei `GetData.java` handelt es sich um ein Servlet. Das bedeutet, dass ein Browser dies über eine bestimmte URL aufrufen kann und eine entsprechende Antwort erhält.

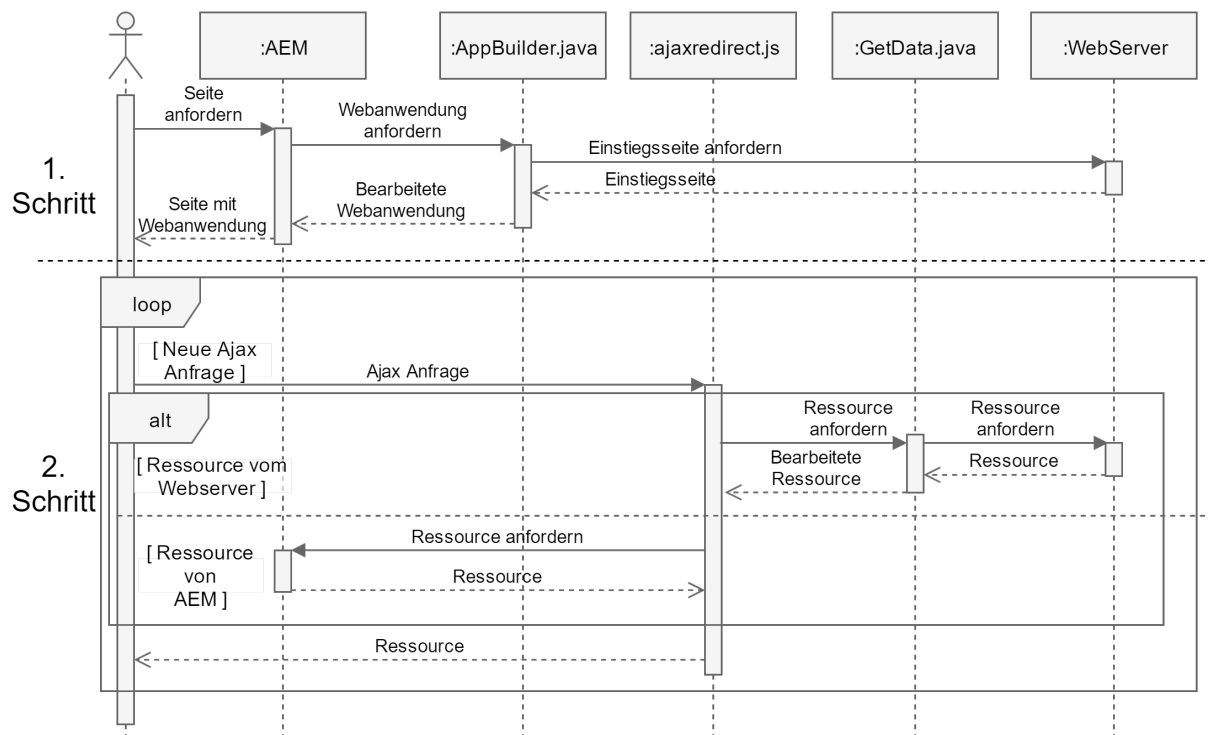


Abbildung 22: Ungefährer Ablauf der AEM-Komponente auf Java-Basis

5.4.2.1.4.1 HTML-Seite aufbereiten

Der erste Schritt dient dazu, die Einstiegsseite der Webanwendung anzufordern.

Der Benutzer fordert hierbei zunächst die Webseite `http://aem.example.com/` an. In dieser befindet sich die AJC, an deren Stelle die Webanwendung von `http://www.example.com:3000/spa/` erscheinen soll.

Nun wird im ersten Schritt `AppBuilder.java` damit beauftragt, den HTML-Quellcode von `http://www.example.com:3000/spa/` anzufordern. `AppBuilder.java` muss jetzt den erhaltenen HTML-Quellcode für die Integration aufbereiten. Einige dieser Aufbereitungsaufgaben sind anwendungsspezifisch und vom jeweils verwendeten Webframework abhängig. Mehr zu den Unterschieden ist dem Absatz 5.4.2.1.5 zu entnehmen. Was immer geschieht, ist, dass relative Referenzen auf JavaScript und CSS-Dateien durch absolute ersetzt werden. Eine Referenz auf `./app.module.js` sollte nach besagtem 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 17 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 17: Ausgangssituation auf Server B

Aus der Aufbereitung würde ein Quellcode wie in Codebeispiel 18 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 18: Aufbereiteter Quellcode

AEM setzt nun an die Stelle der AJC den aufbereiteten Quellcode. Diese HTML-Seite ist nun fertig bearbeitet 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.

5.4.2.1.4.2 Ajax-Anfragen anpassen

Im zweiten Schritt werden Ajax-Anfragen umgeleitet. Diese sind zumeist ebenfalls relativ und würden somit versuchen eine App-Ressource unter *http://aem.example.com/* anzufordern, die sich jedoch unter *http://www.example.com:3000/spa/* befindet. Gerade DI-Bibliotheken laden häufig Skripte nach, aber auch Templates werden häufig nachträglich geladen.

Für die Umleitung überschreibt *ajaxredirect.js* das XMLHttpRequest (XHR)-Objekt. Die XHR-Schnittstelle wird vom W3C spezifiziert und ist ein wesentlicher Be-

standteil von Ajax, das dazu verwendet werden kann, Daten asynchron zwischen Server und Client auszutauschen.

Das überschriebene XHR-Objekt überprüft, ob es sich während einer Ajax-Anfrage bei der angeforderten URL um eine relative oder absolute handelt. Falls diese relativ ist, hat dies zu bedeuten, dass die App-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 Absatz 5.4.2.1.2 erläuterte Sicherheitskonzept SOP, erleichtert. Ohne Servlet würde der Client direkt auf Server-Ressourcen des Webserver 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 dahingehend konfiguriert werden, dass er nur Anfragen vom AEM-Server erlaubt.

Zum anderen können an dieser Stelle noch Änderungen an den angefragten App-Ressourcen erfolgen. Beispielsweise könnten hier die Templates einer AngularJS-Webanwendung nach Hyperlinks durchsucht und diese angepasst werden.

5.4.2.1.5 Aufbereitungsaufgabe bei unterschiedlichen Frameworks

Fehlt komplett...

5.4.2.1.6 Bewertung

Die Lösung mit Java ist wie folgt zu bewerten.

Vorteile:

- Die Struktur der Webanwendung bleibt erhalten.
- Die Webanwendung ist unabhängig von AEM testbar.
- Die Entwickelte AJC lässt sich für ähnliche Webanwendungen wiederverwenden.

Nachteile:

- Betreiben zweier Dienste nötig (AEM und Webserver).
- Ggf. zusätzliche Konfiguration von Webserver und Webanwendung nötig.
- Zusätzliche Laufzeit (gering, ggf. vernachlässigbar).
- Neue AJC für stark abweichende Webanwendungen nötig.

5.4.2.2 Lösungsansatz „JavaScript“

Dieser Lösungsansatz verzichtet auf die beiden Java-Ressourcen vom Lösungsansatz "Java-Servlet" und erfüllt das gewünschte Ziel lediglich mit JavaScript, welche sich ebenfalls in einer AEM-Komponente befindet. Im Gegensatz zur zuvor vorgestellten Variante wird nun der erste Schritt von Abbildung 22 durch eine JavaScript-Komponente ersetzt. Das Prinzip ist hierbei das gleiche, jedoch gibt es hier die Besonderheit, dass es die zu ladenden App-Ressourcen zu synchronisieren gilt.

5.4.2.2.1 Synchronisation

Eine JavaScript-Ressource nachträglich zu laden lässt sich erreichen, indem mittels JavaScript ein `script`-Element in den `head` eingefügt wird. Dies würde in etwa wie in Codebeispiel 19 aussehen.

```
1 var s = document.createElement('script');
2 s.src = "http://www.example.com/jquery.js";
3 document.head.appendChild(s);
```

Codebeispiel 19: Nachladen einer JavaScript-Ressource

Wird dieser Code jedoch wiederholt mit unterschiedlichen URLs aufgerufen, so können Abhängigkeitsproblemen auftreten, da mehrere JavaScript-Ressourcen nahezu gleichzeitig geladen und ausgeführt werden. Somit kann es passieren, dass eine JavaScript-Datei, die eine Webanwendung beinhaltet, vor dem Webframework ausgeführt wird, obwohl die Reihenfolge anders angegeben wurde. Für die Lösung dieses Problems lässt sich die Promise-Schnittstelle verwenden, wie nachfolgendes Codebeispiel zeigt.

```
1 var i = 0, scriptUrls = ...;
2 getScript();
3 function getScript() {
4     loadScript(scriptUrls[i++]).then(getScript);
5 }
6
7 function loadScript(src) {
8     return new Promise(function(resolve) {
9         var s = document.createElement('script');
10        s.src = src;
11        s.onload = resolve;
12        document.head.appendChild(s);
13    });
14 }
```

Codebeispiel 20: Laden und Synchronisieren von JavaScript-Ressourcen

Es wird davon ausgegangen, dass die Variable `scriptUrls` bereits ein Array mit den absoluten Adressen der zu ladenden JavaScript-Ressourcen in der korrekten Reihenfolge beinhaltet. Diese wurden beispielsweise zuvor in der HTML-Seite durch den Einsatz von regulären Ausdrücken gefunden. Anschließend wird die Funktion `getScript` aufgerufen. Die Zeile 4 bewirkt nun zwei Dinge. Zunächst wird die erste JavaScript-Ressource geladen und ausgeführt. Anschließend wird, sobald der JavaScript-Code ausgeführt wurde, die Funktion erneut aufgerufen, um die nächste JavaScript-Ressource zu laden.

Damit nicht fälschlicherweise eine JavaScript-Ressource geladen wird, bevor die vorherige nicht zum Ende ausgeführt wurde, wird hier ein Promise (eng. Versprechen) verwendet, einen in ECMAScript 6 eingeführten Standard. Die Funktion bewirkt, dass die JavaScript-Ressource geladen und ausgeführt wird. Anschließend bewirkt der Promise, dass die Funktion `loadScript` innerhalb von `then(...)` in Zeile 4 ausgeführt wird.

5.4.2.2 Bewertung

Die Lösung mit JavaScript ist wie folgt zu bewerten.

Vorteile:

- Verzicht von Java, somit erfolgt die Integration ausschließlich mit JavaScript.

Nachteile:

- Zusätzliche Ladezeit, da gerade bei den Skripten auf den richtigen Ablauf geachtet werden muss.

5.4.3 Lösungsansatz „Proxy“

Ein weiterer Lösungsansatz wird durch Hinzunahme eines Proxy realisierbar. Durch den Einsatz eines Proxy lassen sich die vom Client angeforderten App-Ressourcen während des HTTP-Request-Response-Zyklus manipulieren.

Im allgemeinen ist ein Proxy ein Rechner, der als Mittelsmann zwischen Client und Server dient. Der Client stellt hierbei die Anfragen an den Proxy. Der Proxy leitet anschließend die Anfrage weiter an den Server. Client und Server kommunizieren somit nie direkt miteinander, sondern nur indirekt über den Proxy.

cite

Im Rahmen dieser Arbeit handelt es sich bei einem Proxy um einen weiteren speziell konfigurierten Webserver. Die Web-Ressourcen werden dabei von dem eigentlichen Webserver über den Proxy an den Client ausgeliefert. Die angefragten App-Ressourcen lassen sich hierbei über den Proxy manipulieren.

5.4.3.1 Erklärung

Die Lösung sieht hierbei vor, dass ein weiterer Webserver als Proxy unter dem Hostnamen *cdn.example.com* so angepasst wird, dass dieser alle Anfragen wie in Abbildung 23 über ein zentrales Skript umleitet. Besagtes Skript kann in einer beliebigen, vom Proxy unterstützten Programmiersprache realisiert werden, wie hier in PHP. Die Konfiguration ließe sich beim Apache HTTP-Server mit dem Modul „mod_rewrite“ realisieren [Fou17c]. Die entsprechende Konfiguration ist Codebeispiel 21 zu entnehmen.

```
1 RewriteEngine on
2 RewriteRule ^/app/(.*)$ /proxy.php?url=$1 [L]
```

Codebeispiel 21: Konfiguration des mod_rewrite Modules für den Proxy-Lösungsansatz

Der Ablauf könnte hierbei wie in Abbildung 23 erfolgen.

Die Konfiguration definiert, dass, sofern eine App-Ressource angefragt wird, deren Pfad mit */app/* beginnt, die Anfrage über *proxy.php* umgeleitet wird. Der restliche Pfad, hier *templates/news.html*, wird dem Skript als Parameter mit übergeben.

Dieses ermittelt nun die benötigte App-Ressource vom eigentlichen Webserver, der unter dem Hostnamen und Port *www.example.com:3000* zu erreichen ist. Etwaige Abweichungen vom angeforderten Pfad und der Pfad auf dem Webserver lassen sich über das Skript wie im Beispiel bewerkstelligen und korrigieren.

Anschließend bearbeitet das Skript die App-Ressource entsprechend und liefert die bearbeitete App-Ressource als Ergebnis zurück. Diese Aktionen bleiben dem Client verborgen. Aus dessen Sicht wird ihm lediglich die angeforderte App-Ressource geliefert.

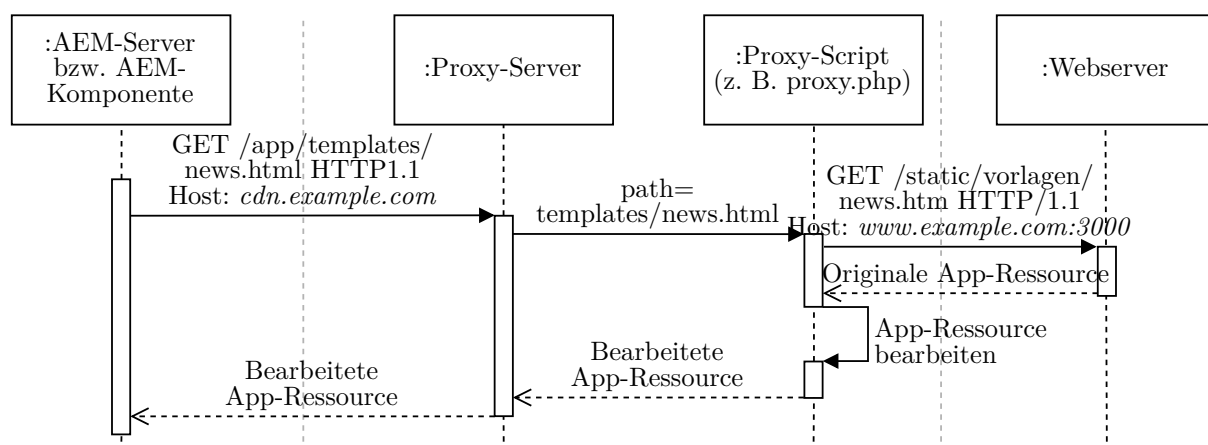


Abbildung 23: Funktionsweise des Proxy

Die Lösung ähnelt dem Servlet aus Unterunterabschnitt 5.4.2.1, da auch hier zur Laufzeit Änderungen vorgenommen werden können.

Seitens AEM kommt auch hier eine AEM-Komponente zum Einsatz, welche die initiale Anfrage an die HTML-Seite der Webanwendung stellt, was in etwa dem ersten Schritt aus Abbildung 22 entspräche. Die Logik der AJC vom Lösungsansatz „Java-Servlet“ wird hier auf das Proxy-Skript ausgelagert. Innerhalb der AEM-Komponente wird lediglich die Start-URL des Proxy definiert.

5.4.3.2 Bewertung

Die Lösung mit einem Proxy ist wie folgt zu bewerten.

Vorteile:

- Die Struktur der Webanwendung bleibt erhalten.
- Die Webanwendung ist unabhängig von AEM testbar.
- Die entwickelte AJC lässt sich wiederverwenden.
- Die Logik des Skriptes ist weitestgehend frei wählbar und nicht von AEM abhängig.

Nachteile:

- Betreiben von zwei bzw. drei Diensten nötig (AEM und Webserver und Proxy).
- Ggf. zusätzliche Konfiguration von Webserver und Webanwendung nötig.
- Zusätzliche Laufzeit (gering, ggf. vernachlässigbar).

5.5 Möglichkeiten der Kombination

Jeder der genannten Lösungen ist generell für sich alleine stehend funktionell und erfüllt richtig angewendet das gewünschte Ziel. Es macht aber auch durchaus Sinn einige Lösungsansätze zu kombinieren.

5.5.1 Clientlib und Content-Ordner

Diese Kombination sieht vor, dass die App-Ressourcen in zwei Gruppen unterteilt werden. Zum einen wären hier alle App-Ressourcen, die bereits in der HTML-Seite referenziert sind, also zu Beginn geladen werden. Diese CSS- und JavaScript-Ressourcen werden als Clientlib realisiert. Zum anderen werden alle weiteren Ressourcen unter */content* abgelegt. Folgendes Beispiel zur Erklärung. Es wird angenommen, dass sich die AJC unter *h*

`http://aem.example.com/content/app` befindet. Über die AJC wurde bereits die Clientlib und somit die im ersten Schritt benötigten App-Ressourcen geladen. Nun wird über Ajax versucht eine JavaScript-Ressource unter der relativen URL `./templates/app.html` aufzurufen. Somit ergibt sich die absolute Adresse `http://aem.example.com/content/app/templates/app.html`, was innerhalb des JCR die Knoten `/content/app/templates/app.html`. Der genaue Ablageknoten unter `/content` ist somit von der URL abhängig, unter dem die Webanwendung auffindbar sein soll.

5.5.1.1 Bewertung

Die Lösungskombination ist wie folgt zu bewerten.

Vorteile:

- Schnell zu realisieren.
- Keinerlei Anpassung der Webanwendung nötig. Relative Pfade werden korrekt aufgelöst.
- Die Realisierung erfolgt als AJC, somit lässt sich diese frei innerhalb einer Webseite platzieren.

Nachteile:

- App-Ressourcen müssen innerhalb eines entsprechenden Knoten unter `/content` liegen. Wird die Anwendung in eine andere Webseite eingebettet, müssen die App-Ressourcen in einen anderen Knoten verschoben werden.

5.5.2 Proxy und Java

Der Lösungsansatz „Java-Servlet“ hat den Nachteil, dass für jedes Webframework, oder gar für jede Webanwendung eine eigene AJC benötigt wird. Denn je nach Art der Webanwendung muss die AJC diese unterschiedlich vor der Integration bearbeiten.

Weiterhin vom Nachteil ist, dass die Webanwendung über den Proxy für Besucher voll zugänglich ist.

Um die jeweiligen Vorteile zu vereinen werden beide Lösungsansätze hier vereint.

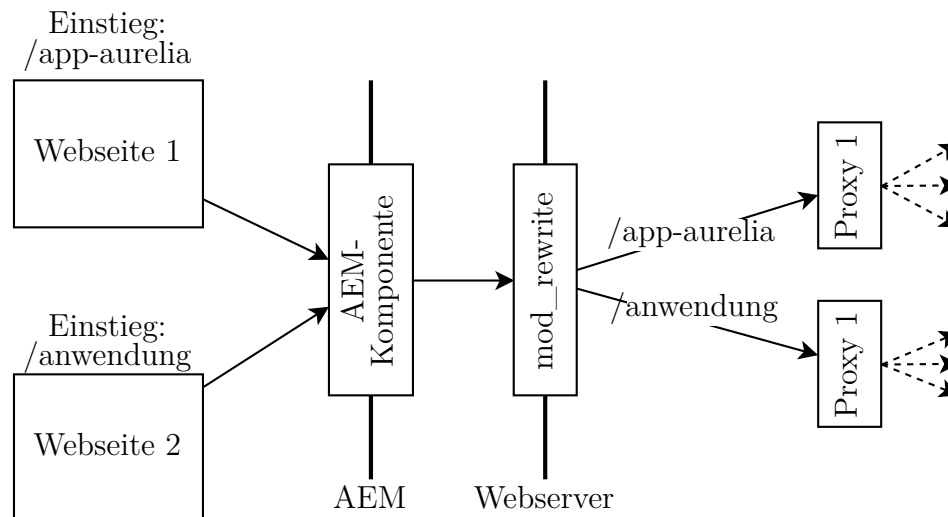


Abbildung 24: Aufbau der Proxy und Java Kombination

Hierbei wird die Logik, welche für die Anpassung der Webanwendung benötigt wird, in den Proxy verschoben. Somit genügt eine AEM-Komponente, in welcher nur die Einstiegsseite der Webanwendung zu konfigurieren ist. Die Dateien `AppBuilder.java`, `GetData.java` und `ajaxredirect.js` werden auch hier eingesetzt, dienen jedoch nur zum Auslösen der HTTP-Anfragen.

Für jede Webanwendung lässt sich nun auf dem Server ein eigenes Proxy-Skript erstellen, welches sich z. B. bei Apache durch „mod_rewrite“ einem entsprechenden URL-Muster zuordnen lässt.

Neben der Modularität und das nur eine AEM-Komponente Verwendung findet besteht die Kommunikation hier nur zwischen der AJC und dem Webserver. Der Client kommuniziert nie direkt mit dem Proxy. Somit lässt sich der Webserver konfigurieren, dass nur der AEM-Server und ggf. einige Test-Nutzer auf diesen Zugriff haben. Allen anderen Nutzern wird der Zugriff verweigert.

5.5.2.1 Bewertung

Die Lösungskombination ist wie folgt zu bewerten.

Vorteile:

- Nur eine AEM-Komponente.
- Striktere Trennung. Alle, was die Webanwendung betrifft, auch die Logik der Integration, ist außerhalb von AEM.
- Jeder Proxy kann in einer eigenen Skriptsprache erfolgen.
- Ggf. weitere Logik in der AEM-Komponente möglich, wie z. B. eine zentrale Komprimierung der vom Proxy erhaltenen Ressourcen.

Nachteile:

- Externer Webserver wird benötigt.

5.6 Weitere Werkzeuge und Vorgehensprozesse

Neben der Integration gibt es noch Aufgaben, die es vor und nach der Integration zu erfüllen gilt. So besteht vor der Integration die Möglichkeit, die entwickelte Webanwendung entsprechend aufzubereiten. Dies kann die Performance verbessern und den Prozess der Integration vereinfachen. In diesem Schritt finden Werkzeuge wie Grunt und Gulp aus Unterabschnitt 3.1.3 ihre Verwendung.

Weiterhin sollte gerade bei einer SPA der AEM-Server entsprechend konfiguriert werden, um Suchmaschinen die korrekte Indizierung der einzelnen Webseiten zu gestatten.

5.6.1 Minimierung

Für den Produktiveinsatz lassen sich die JavaScript-Ressourcen minimieren. Hierbei werden zum Beispiel unnötige Leerzeichen entfernt oder auch lange Variablennamen durch kürzere ersetzt. Dies minimiert die Übertragungszeit vom Server zum Client, da das zu übertragende Volumen sinkt.

Generell existieren für die Minimierung von JavaScript-Quellcode verschiedenste Werkzeuge. Eines davon wäre UglifyJS [Baz16]. Hiermit lässt sich JavaScript-Quellcode unter anderem minimieren. Das Werkzeug wird unter der BSD-Lizenz veröffentlicht und über npm installiert.

5.6.2 Konkatinierung und weitere Laufzeitoptimierung

Für den Produktiveinsatz lassen sich die Quelldateien konkatenieren. Hierbei wird die Anzahl der einzelnen Dateien minimiert, indem diese zu einer oder mehreren großen Dateien zusammengefügt werden. Dies minimiert die Übertragungszeit vom Server zum Client, da weniger HTTP-Request-Response-Zyklen durchlaufen werden müssen. Weiterhin existieren so auch weniger Dateien, die es in das AEM zu integrieren gilt, was zum Beispiel die direkte Integration, wie in Unterabschnitt 5.4.1 beschrieben, erleichtert.

Weiterhin lassen sich bei manchen Frameworks die HTML-Templates in JavaScript-Dateien umwandeln. Somit ist auch hier eine anschließende Minimierung und Kompriemierung möglich.

Allgemein Generell existieren für die Konkatinierung von JavaScript-Quellcodes verschiedenste Werkzeuge. Auch hierfür lässt sich UglifyJS verwenden. Bei der Konkatenierung ist bei dessen Einsatz 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-Dateien bereits deren Abhängigkeit syntaktisch hinterlegt. Je nach Art der verwendeten DI darf der Entwickler auf unterschiedliche Werkzeuge zurückgreifen. Alle gängigen Bibliotheken aus Tabelle 4 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 umzuwandeln. Als Werkzeug wird hier Grunt in Verbindung mit dem npm-Paket 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 Im Gegensatz zum JiT-Ansatz, bei dem Skripte und App-Ressourcen erst dann geladen werden, wenn sie benötigt werden, wird beim AoT-Ansatz alles, was später benötigt werden könnte, zu Anfang geladen. In der Dokumentation von AngularJS2 wird der AoT-Ansatz erklärt [Goo16a] und wie dieser angewandt wird.

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

React Um die Laufzeit der Webanwendung zu optimieren, kann man diese zuvor in eine 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 umwandeln, und beherrscht auch JSX. Die Umwandlung lässt sich manuell, oder auch wahlweise durch Gulp und Grunt anstoßen.

5.6.3 Übertragen von Ressourcen

Die Übertragung von Ressourcen in das AEM kann über mehrere Wege erfolgen.

5.6.3.1 WebDAV

Zunächst lässt sich über die WebDAV-Schnittstelle mit einem entsprechenden Programm die Struktur des JCR anzeigen und bearbeiten. Bei Web-based Distributed Authoring and Versioning (WebDAV) handelt es sich um einen offenen Standard für die Bereitstellung von Daten über ein Netzwerk. Neben dem Löschen und Verschieben von Knoten ist hier auch das Hochladen von lokalen Dateien möglich.

5.6.3.2 HTTP

Weiterhin lassen sich Ressourcen auch über HTTP übertragen. Ein hilfreiches Programm hierfür ist „curl“ [Ste17]. Über die Kommandozeile lassen sich so Dateien in das AEM übertragen. Folgender Befehl in Codebeispiel 22 würde zum Beispiel eine Datei mit dem Namen `flowerpot.png` in das JCR unter `/content/flowerpot.png` ablegen. Für die Authentifizierung wird ein gültiger Benutzername mit Passwort, hier beides „admin“, benötigt.

```
1 curl -u admin:admin -T flowerpot.png http://localhost:4502/content/  
   ↪ flowerpot.png
```

Codebeispiel 22: Eine Datei mit curl in das AEM übertragen

5.6.3.3 CRX-Paket

Sofern die WebDAV-Schnittstelle auf der Zielplattform nicht geöffnet sein sollte, gibt es die Alternative, die App-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 die der Entwicklungsumgebung. 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. Alternativ lässt sich das CRX-Paket auch mit Maven erstellen. Durch Maven lassen sich standardisierte Anwendungen erstellen und verwalten. Eine entsprechende Anleitung zum Erstellen eines Maven-Projektes und wie sich hiermit ein CRX-Paket erschaffen lässt stellt Adobe zur Verfügung [Inc17];

5.7 Suchmaschinen

Falls die Webanwendung als SPA mit Hinzunahme der in Absatz 3.2.3.2.3 beschriebenen `pushState`-Funktion entwickelt wurde, so ist eine entsprechende Konfiguration des AEM-Servers vonnöten, damit die Webseiten der Webanwendung korrekt aufgerufen werden.

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 App-Ressourcen der Webanwendung auch unter besagter URL erreichbar.

Es wird angenommen, dass alle URLs, die einen Punkt besitzen, einer App-Ressource wie Skripte, Bilder etc. entsprechen. Ansonsten handelt es sich um eine Webseite innerhalb der SPA. Da eine SPA aus einer HTML-Seite, z. B. `index.html`, besteht, sollen alle Webseiten-Anfragen auf diese verweisen. Alle weiteren App-Ressourcen erhalten eine relative Umleitung. Tabelle 8 zeigt beispielhaft einige vom Browser angeforderte Web-Ressourcen und unter welchen JCR-Knoten diese zu finden sind.

Angeforderte URLs	Ressourcotyp	Hat Punkt	Ziel JCR-Knoten
<code>http://localhost:4502/spa/</code>	Webseite	Nein	<code>/content/angular/index.html</code>
<code>http://localhost:4502/spa/news/</code>	Webseite	Nein	<code>/content/angular/index.html</code>
<code>http://localhost:4502/spa/about/</code>	Webseite	Nein	<code>/content/angular/index.html</code>
<code>http://localhost:4502/spa/img/logo.png</code>	Bild	Ja	<code>/content/angular/img/logo.png</code>
<code>http://localhost:4502/ws/news/</code>	Webseite	Nein	Keine Weiterleitung, da Pfad nicht mit <code>/spa</code> beginnt.
<code>http://localhost:4502/spa/ressources/style.css</code>	Stylesheet	Ja	<code>/content/angular/resources/style.css</code>

Tabelle 8: 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 URLs gesetzt. In der Standardkonfiguration befinden sich diese unter `/http/map/`. Jede Regel setzt sich aus der hierarchischen Struktur und verschiedenen Eigenschaften zusammen. Um die angeforderten URLs aus Tabelle 8 entsprechend umzuleiten, empfiehlt sich die Struktur wie in Codebeispiel 23.

```

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 23: 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 diese so zu erweitern, zu reduzieren und zu manipulieren [Fri09, S. 1 f.].

Der reguläre Ausdruck wird aus den Namen der Knoten bzw. deren Werten der Eigenschaft

sling:match, falls ein Knoten diese besitzt, zusammengesetzt. Der Wert der Eigenschaft sling:internalRedirect steht für die Ziel-URL im JCR.

Aus der gegebenen Konfiguration resultieren zwei Regeln. Der Knoten `/etc/map/http/localhost.4502/spa/path/` beschreibt, dass alle URLs, die mit `http://localhost:4502/spa/` beginnen, aber keinen Punkt in der URL haben, auf `/content/angular/index.html` verweisen. Ansonsten gilt die erstellte Regel unter Knoten `/etc/map/http/localhost.4502/spa/`.

Wie die Auflösung verschiedener URLs geschehen würde, ist der Abbildung 25 zu entnehmen.

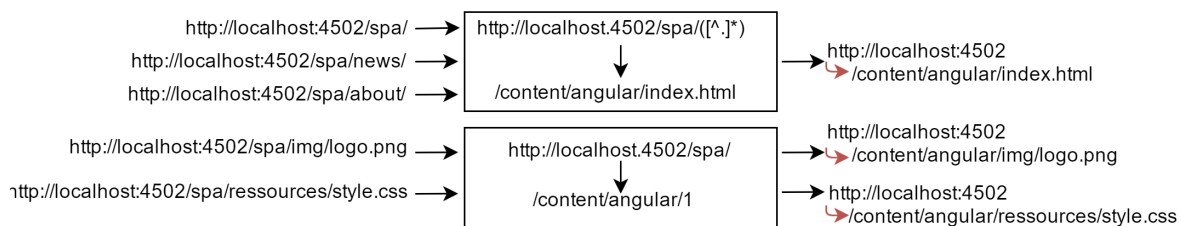


Abbildung 25: Möglichkeit zum Auflösen von Anfragen unter AEM

Sollten die Ressourcen über einen Proxy geleitet werden, so ließen sich diese Regeln auch mit dem Modulen „mod_rewrite“ abbilden, sofern es sich bei dem Proxy um einen Apache HTTP-Server handelt.

5.8 Bekannte Konflikte zwischen Skripten

Während der Integration sind keinerlei Problematiken zwischen den Skripten von AEM und Skripten seitens der Webanwendung bzw. eines Webframeworks aufgetreten.

Fehlt komplett...

5.9 Bewertung der Lösungen

In den vorherigen Abschnitten wurden verschiedene Lösungsansätze erarbeitet. Nachfolgende Abbildung 26 dient als kleine Hilfe, welche Lösung für eine vorstehende Integration am geeignetsten wäre.

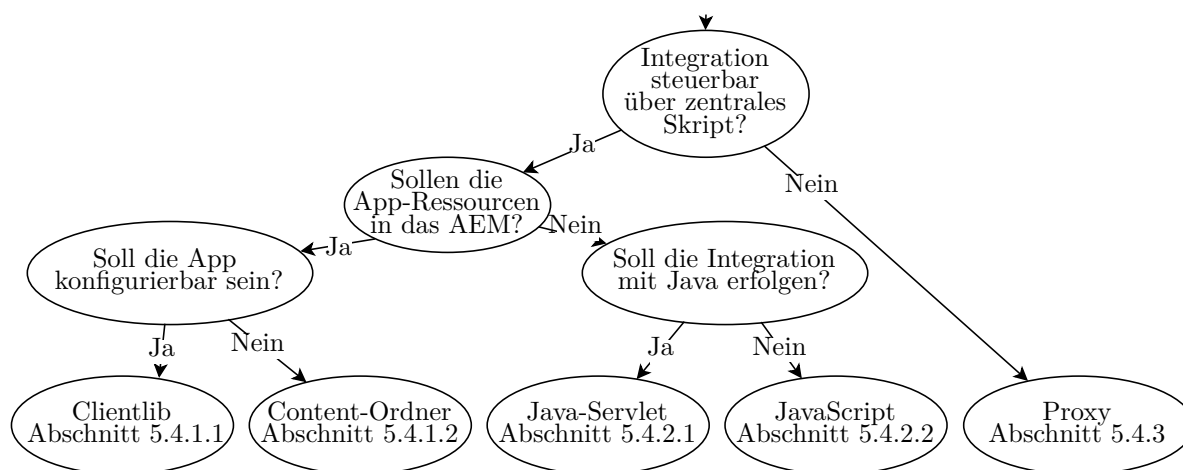


Abbildung 26: Entscheidungsbaum für eine adäquate Lösung

Fehlt komplett...

Folgende Tabelle 9 listet erneut die Kürzel der in Abschnitt 4.3 gestellten Anforderung und vergleicht, in welchem Kapitel dieser Arbeit diese erfüllt bzw. gelöst wurden.

Anforderung	Gelöst in Kapitel
A1	5.4
A2	5.4
A3	-
A4	5.4
A5	5.4
A5.1	5.4.1
A5.2	5.4.2.1
A5.3	5.4.2.2
A6	5.6
A7	5.7
A8	??
A9	??

Tabelle 9: Anforderungserfüllung

6 Zusammenfassung

Diese Arbeit hatte als Ziel, Lösungsansätze zu erarbeiten, eine Webanwendung, die unter Verwendung moderner clientseitiger Webframeworks entwickelt wurde, in den Adobe Experience Manager zu integrieren.

Nach ersten Integrationsversuchen stellte sich heraus, dass je nach gesetzten Rahmenbedingungen unterschiedliche Lösungsansätze zu erarbeiten sind. So wurden entsprechend angepasste Lösungsansätze entwickelt, um das Ziel zu erreichen, eine zuvor entwickelte Webanwendung innerhalb einer mit AEM erstellten Webseite anzuzeigen. Diese Lösungsansätze wurden miteinander verglichen, um eine Entscheidungshilfe zu schaffen. Weiterhin wurden Themen untersucht und behandelt, die nicht die eigentliche Integration betreffen, von dieser aber betroffen sind. All diese Punkte wurden gemeinsam im Kapitel 5 behandelt, nachdem im vorhergehenden Kapitel 4 auf die bestehenden Probleme eingegangen wurde.

6.1 Fazit

Nachdem im ersten Kapitel auf die Beweggründe dieser Arbeit eingegangen wurde, folgten die Erklärung der benötigten Grundlagen und eine nähere Beschreibung der vier betrachteten clientseitigen Webframeworks. Anschließend wurde näher auf den Ist-Zustand eingegangen, woraufhin die Vorstellung der Lösungsansätze folgte.

Die fünf vorgestellten Lösungsansätze „Clientlib“, „Content Ordner“, „Java-Servlet“, „JavaScript“ und „Proxy“ ermöglichen es, unter gewissen Umständen das gesetzte Ziel zu erreichen. Jeder Lösungsansatz besitzt, damit dieser praktisch umgesetzt werden kann, bestimmte Anforderungen, die zuvor erfüllt werden müssen. Daher sind manche Lösungsansätze nur unter gewissen Einschränkungen realisierbar.

Insgesamt wurden verschiedene Lösungsansätze erarbeitet, die eine Ablage der App-Ressourcen sowohl in die JCR-Struktur des AEM als auch auf einen separaten Webserver erlauben. Somit kann je nach Wunsch die Lösung kompakt erfolgen, in dem nur das AEM Verwendung findet oder diese durch weitere Schichten erweitert werden. Jeder Ansatz hat seine Vor- und Nachteile. Daher empfiehlt es sich, bei komplexeren Webanwendungen eine Kombination aus mehreren Lösungsansätzen. Somit lassen sich benötigte Anpassungen und Manipulationen an der Webanwendung an mehreren Stellen vornehmen und die Webanwendung strukturiert integrieren. Für einfachere Lösungsansätze hingegen reicht

zumeist eine einfache Clientlib.

Weiterhin wurde gelöst, wie die Webanwendung korrekt von einer Suchmaschine indiziert werden kann, trotz nachgeladenem Content via JavaScript und einer möglichen Umsetzung als Single Page Application.

6.2 Ausblick

Die Lösungsansätze innerhalb dieser Arbeit wurden mit den aus Abschnitt 5.3 aufgelisteten Webanwendungen getestet. Diese sind oft Beispielanwendungen direkt von den jeweiligen Entwicklern, weswegen diese sehr nahe an den Standards der Webframeworks entwickelt wurden. Um die Tragfähigkeit der Lösungsansätze weiter zu untersuchen, sollten die Verfahren auch mit Webanwendungen getestet werden, die von anderen Entwicklern stammen.

Weiterhin konnte innerhalb dieser Arbeit nicht weiter auf das Thema der Konfiguration innerhalb der Autoren-Bedienoberfläche eingegangen werden.

Schlussendlich ist der eigentliche Beweis der Lösungsansätze und somit auch der Beweis für den Mehrwert dieser Masterarbeit, dass diese innerhalb eines Projektes genutzt und zu guter Letzt im Produktiveinsatz Verwendung findet.

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](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 - mishooUglifyJS*. <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 - ericclemonsgrunt-angular-template*. <https://github.com/ericclemons/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] DEELEMEN, Paplo: *Learning Angular 2*. Packt Publishing, 2016
- [Eis14] EISENBERG, Rob: *Leaving Angular*. <http://eisenbergeffect.bluespire.com/leaving-angular/>. Version: 11 2014
- [Eis17] EISENBERG, Rob: *Aurelia Docs - Quick Start*. <http://aurelia.io/hub.html#/doc/article/aurelia/framework/latest/quick-start/>. Version: jan 2017
- [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
- [Fou17a] FOUNDATION, Apache S.: *Apache Jackrabbit - Node Type Notation*. jan 2017
- [Fou17b] FOUNDATION, The Apache S.: *Apache Module mod_headers*. http://httpd.apache.org/docs/2.0/mod/mod_headers.html. Version: jan 2017
- [Fou17c] FOUNDATION, The Apache S.: *Apache Module mod_rewrite*. http://httpd.apache.org/docs/current/mod/mod_rewrite.html. Version: jan 2017
- [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 Compilation*. <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](https://docs.angularjs.org/api/ng/provider/$sceDelegateProvider). Version: nov 2016
- [Goo16d] GOOGLE: *AngularJS: API: \$templateCache*. [https://docs.angularjs.org/api/ng/service/\\$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
- [Inc17] INCORPORATED, Adobe S.: *How to Build AEM Projects using Apache Maven*. <https://docs.adobe.com/docs/en/aem/6-2/develop/dev-tools/ht-projects-maven.html>. Version: jan 2017
- [Int16] INTERNATIONAL, Ecma: *ECMAScript 2016 Language Specification*. <https://www.ecma-international.org/ecma-262/7.0/index.html>. Version: jun 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
- [Kan16c] KANDLER, Julian: *Aurelia-Example*. <https://github.com/KandlerLi/Aurelia-Example>. Version: dec 2016
- [Kan16d] KANDLER, Julian: *React-Example*. <https://github.com/KandlerLi/React-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
- [Lug16] LUGO, Sebastian: *redux-tetris*. <https://github.com/slugo/redux-tetris>. Version: dec 2016
- [MR06] MORVILLE, Peter ; ROSENFELD, Louis: *Information Architecture for the World Wide Web*. 3. O'Reilly Verlag, 2006
- [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
- [Ste17] STENBERG, Daniel: *curl - command line tool and library for transferring data with URLs*. <https://curl.haxx.se/>. Version: jan 2017
- [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
- [W3C14a] W3C: *Cross-Origin Resource Sharing*. <https://www.w3.org/TR/cors/>. Version: jan 2014
- [W3C14b] W3C: *Open Web Platform Milestone Achieved with HTML5 Recommendation*. <https://www.w3.org/2014/10/html5-rec.html.en>. Version: oct 2014
- [wcm17] WCMIO: *grunt-aem-clientlib-generator*. <https://www.npmjs.com/package/grunt-aem-clientlib-generator>. Version: jan 2017
- [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