

Analyse von Ruby on Rails 3 Web Content Management Systemen

Diplomarbeit

vorgelegt am: 23. Oktober 2011

**am Fachbereich Medien der Hochschule für Technik,
Wirtschaft und Kultur Leipzig**

Name:	Stephan Keller
Fachbereich:	Medien
Studiengang:	Medientechnik
Studienjahrgang:	2006
Erstgutachter:	Prof. Dr.-Ing. Robert Müller
Zweitgutachter:	Prof. Dr.-Ing. Jörg Bleymehl

Inhaltsverzeichnis

1	Einleitung	5
1.1	Ausgangslage	5
1.2	Motivation und Zielsetzung	6
1.3	Aufbau der Arbeit	7
2	Grundlagen	9
2.1	Entwicklung mit Ruby on Rails	9
2.1.1	Don't-Repeat-Yourself (DRY)	10
2.1.2	Convention over Configuration	10
2.1.3	Model-View-Controller (MVC)	11
2.1.4	REST	13
2.1.5	Rack und Middleware	16
2.1.6	Generatoren	20
2.2	Web Content Management, Content Life Cycle und Web-Publishing . . .	20
2.3	Kriterienkatalog	23
2.3.1	Erstellung	24
2.3.2	Kontrolle	25
2.3.3	Freigabe	26
2.3.4	Publikation	26
2.3.5	Terminierung und Archivierung	27
3	Analyse des Leistungsfähigkeit von Ruby on Rails Web Content Management Systemen	28
3.1	Vorbetrachtung	28
3.2	Bezugsquellen	30
3.3	Alchemy CMS	32
3.3.1	Funktionsprinzipien	32

3.3.2	Erweiterungen	35
3.3.3	Verwendete Technologien	35
3.4	Browser CMS	37
3.4.1	Funktionsprinzipien	38
3.4.2	Erweiterungen	40
3.4.3	Verwendete Technologien	41
3.5	Locomotive CMS	42
3.5.1	Funktionsprinzipien	42
3.5.2	Erweiterungen	44
3.5.3	Verwendete Technologien	46
3.6	Refinery CMS	46
3.6.1	Funktionsprinzipien	47
3.6.2	Erweiterungen	49
3.6.3	Verwendete Technologien	50
3.7	Durchführung der funktionalen Analyse	50
3.7.1	Erstellung	51
3.7.2	Kontrolle	61
3.7.3	Freigabe	64
3.7.4	Publikation	66
3.7.5	Terminierung/Archivierung	71
3.8	Auswertung der Ergebnisse	73
4	Beschreibung ausgewählter Implementierungsdetails	74
4.1	Erweiterungen	74
4.2	Nutzeroberfläche	78
5	Lösungsvorschläge	81
5.1	Implementierung eines Ruby on Rails Content Repository	81
5.1.1	Idee und Konzept	81
5.1.2	Das Java Content Repository (JCR)	82
5.1.3	Umsetzungsvarianten innerhalb von Ruby on Rails	84
5.1.4	Vorteile für die gewählten Ruby on Rails WCMS	84
5.2	Übertragung des Typo3 5.0 Phoenix User-Interfaces in Rails 3.1	85
5.2.1	Typo3 5.0	86

5.2.2	Ext JS und Ext Direct	88
5.2.3	Umgesetzte Rails-Anwendung	90
5.2.4	Vorteile der Typo3-Nutzeroberfläche	90
6	Zusammenfassung	92
6.1	Fazit	92
6.2	Ausblick	92
7	Anhang	94
7.1	Liste bestehender Rails 2/3 Web Content Management Systeme bzw. Blogging-Software	94
7.2	Funktionsumfang der untersuchten Rails WCMS	95
7.3	Crudify-Methode von Refinery CMS 1.0.8	100
7.4	Ext-Direct Spezifikation für Ext Js 3.0	107
7.5	Nutzung von Extr in einer Rails 3.1-Anwendung	115
8	Literaturverzeichnis	118

1 Einleitung

1.1 Ausgangslage

Die Skriptsprache PHP gehört weltweit zu den meist genutzten serverseitigen Programmiersprachen. Im August 2011 werden über 75 Prozent der dynamisch generierten Internetseiten mit dem PHP Hypertext Preprocessor erzeugt¹.

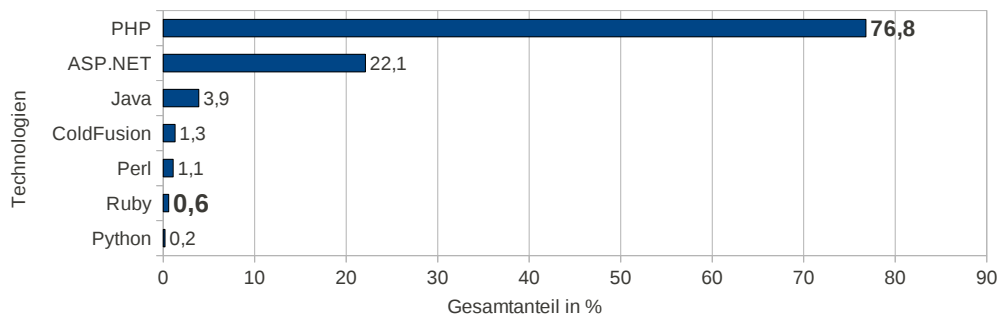


Abbildung 1.1: Nutzung verschiedener Programmiersprachen auf Servern
Quelle: Eigene Darstellung nach [w3t11]

Auch im Bereich der Web Content Management Systeme² spiegelt sich diese Dominanz wider. Betrachtet man die Angaben des Content Management Portals cmsmatrix.org³,

¹W3tech erstellt täglich eine aktualisierte Auflistung über die Verwendung von serverseitigen Programmiersprachen. Es werden dabei die nach dem Alexia Ranking eine Million beliebtesten Internetseiten auf ihre Konfiguration untersucht.

²Im folgenden wird für den Begriff Web Content Management System(e) die Abkürzung WCMS verwendet.

³<http://cmsmatrix.org> ermöglicht eine Gegenüberstellung der Funktionalitäten von Content Management Systemen unterschiedlicher Programmiersprachen.

existieren neben den vor allem in Deutschland verwendeten Open Source-Lösungen TYPO3, Drupal, Contao oder Joomla! über 500 weitere in PHP implementierte Web Content Management Systeme unterschiedlichster Ausprägung und Qualität. Ruby als Programmiersprache findet hingegen nur bei etwa 1 Prozent der erfassten Server Verwendung. Die dabei umgesetzten Projekte sind jedoch meist individuelle, browser-basierte Applikationen, die für Unternehmen und deren spezifisches Geschäftsfeld entwickelt wurden. Bekannte Vertreter sind hier u.a. die webbasierte Projektmanagement-Applikation Basecamp von 37signals⁴, der Microblogging-Dienst Twitter⁵ und der webbasierte Hosting-Dienst Github⁶ für Software-Entwicklungsprojekte. Diese individuellen Lösungen werden dabei meist unter Zuhilfenahme eines Web Application Framework realisiert, das den Entwicklungsprozess unterstützt und vereinfacht.

1.2 Motivation und Zielsetzung

Ruby on Rails⁷ hat sich seit der Veröffentlichung der Version 1.0 im Juli 2004 zu einem der bekanntesten Webframeworks der Ruby Fangemeinde entwickelt. Startups⁸ sowie etablierte Unternehmen greifen dabei verstärkt⁹ auf das Rails Framework zurück, um ihre webbasierten Geschäftsideen und -modelle zu realisieren. Wird neben der Webapplikation zusätzlich eine Internetseite zur Repräsentierung des Unternehmens benötigt, haben sich in der Praxis folgende zwei Lösungsansätze herausgebildet:

1. Bei geringem Umfang der zusätzlichen Internetseite werden die Inhalte manuell in HTML-Dateien angelegt und anschließend in die Rails-Anwendung integriert. Komfortable Möglichkeiten der Content-Verwaltung werden nicht angeboten oder später rudimentär nach implementiert. Änderungen der Inhalte sind teilweise mit

⁴Projektseite von Basecamp: <http://basecamp.com/>

⁵ Großteile der Programmierung von Twitter basierten bis April 2011 auf dem Ruby on Rails Framework.

⁶Github greift neben Ruby on Rails noch auf andere Webframeworks und Technologiesysteme zurück.

⁷Im weiteren Verlauf dieser Arbeit wird für das Webframework Ruby on Rails die Kurzform Rails verwendet.

⁸Der Begriff Startup bezeichnet hier junge Unternehmen, die sich mit ihrem neuartigen, meist innovativen Produkt noch nicht am Markt etabliert haben.

⁹Unter <http://rubyonrails.org/applications> findet sich eine Übersicht ausgewählter Unternehmen, die auf Grundlage von Ruby on Rails teilweise gewinnerzielende Webanwendungen umgesetzt haben. Die Entwicklungseinschätzung erfolgte an Hand der Community und dem Arbeitsmarktangebot, der eine zunehmende Nachfrage für Rails-Entwickler erkennen lässt.

erhöhtem Aufwand verbunden oder erfordern zusätzliche Programmierkenntnisse¹⁰.

2. Komplexe Internetseiten mit vielen Inhalten und anspruchsvollem Layout werden über ein Web Content Management System eines Drittanbieters realisiert. Die Rails-Anwendung fungiert als Zwischenstation und leitet bestimmte Anfragen an das externe WCMS weiter.

Während der erste Lösungsansatz bei wenigen Inhalten noch vertretbar ist, erfordert die Verwendung eines externen WCMS zusätzlichen Installations- und Wartungsaufwand. Weiterhin erhöht sich der Bedarf an Programmierern, da neben Ruby nun auch andere Programmiersprachen (die des externen WCMS) Verwendung finden können.

Ziel der vorliegende Arbeit ist es daher, die Möglichkeiten einer komplett rails-basierten Web Content Management Verwaltung zu untersuchen, um so den Einsatz eines externen WCMS überflüssig werden zu lassen.

Dafür werden unter Verwendung der vergleichenden Methode die ausgewählten Ruby on Rails Content Management Systeme Alchemy CMS, Browser CMS, Locomotive CMS und Refinery CMS einem Kriterienkatalog gegenübergestellt, der allgemein gültige Anforderungen an Web Content Management Systeme formuliert. An Hand der Ergebnisse des Vergleichs kann abschließend eine Leistungsbeurteilung für die gewählten Systeme erfolgen, in wie weit sich diese für den Einsatz im Bereich des Web-Publishing eignen. Zusätzlich wird auf ausgewählte Implementierungsdetails der umgesetzten Systeme eingegangen.

1.3 Aufbau der Arbeit

Die vorliegende Diplomarbeit gliedert sich in sechs Abschnitte:

In Kapitel 2 werden die für die Implementierungsanalyse der Rails WCMS notwendigen theoretische Grundlagen zu dem Ruby on Rails Framework geschaffen. Darüber

¹⁰Änderungen am Quellcode von Rails-Anwendungen im Produktivmodus erfordern immer einen Neustart des Servers.

hinaus wird der für die Ermittlung der Leistungsfähigkeit verwendete Kriterienkatalog vorgestellt.

In Kapitel 3 folgt die Vorstellung und Leistungsanalyse der ausgewählten Ruby on Rails 3 Web Content Management Systeme Alchemy CMS, Browser CMS, Lokomotive CMS und Refinery CMS. Dabei werden die Kriterien des im Kapitel 2 vorgestellten Katalogs mit den tatsächlich gebotenen Funktionalitäten der gewählten WCMS verglichen.

Kapitel 4 beschreibt ausgewählte Implementierungsdetails der analysierten WCM-Systeme und arbeitet eventuell vorhandene Problemstellen heraus. Darauf aufbauend werden in Kapitel 5 mögliche Lösungsansätze demonstriert und die dafür notwendigen theoretischen Grundlagen geschaffen. Kapitel 6 schließt die Arbeit mit einer Zusammenfassung der herausgearbeiteten Ergebnisse ab und gibt einen Ausblick auf zukünftige Entwicklungspotenziale.

2 Grundlagen

2.1 Entwicklung mit Ruby on Rails

2004 arbeitete der dänische Programmierer David Heinemeier Hansson an der Umsetzung eines webbasierten Projektmanagement-Tools mit dem Namen Basecamp¹. Die bei der Realisierung des Projektes umgesetzten Teilkomponenten extrahierte er später und veröffentlichte sie 2005 als Framework unter dem Namen Ruby on Rails. Ruby on Rails basiert dabei auf der objektorientierten Programmiersprache Ruby und ermöglicht die schnelle Entwicklung von Webanwendungen nach dem MVC-Paradigma (Kapitel 2.1.3). 6 Jahre später (die zahlreiche Änderungen und Verbesserungen am Framework ermöglichten) beschreibt sich das Ruby on Rails-Projekt selbst mit folgenden plakativen Worten:

Ruby on Rails is an open-source web framework that's optimized for programmer happiness and sustainable productivity. It lets you write beautiful code by favoring convention over configuration. [Rai11a]

Diese subjektive Aussage der Rails-Kernentwickler bezieht sich dabei auf viele Ansätze und Entwicklungsabläufe, die innerhalb des Frameworks umgesetzt werden. Im folgenden Abschnitt sollen die mit dieser Aussage angedeuteten wichtigsten Prinzipien, Paradigmen und Programmierabläufe des Frameworks zusammengefasst werden, um auf dieser Grundlage eine Betrachtung ausgewählter Implementierungsdetails der Rails WCMS in Kapitel 4 zu ermöglichen. Auf die Themen Rest (Kapitel 2.1.4) und Middleware (Kapitel 2.1.5) wird dabei ausführlicher eingegangen, da sie für die Realisierung der in Kapitel 5 umgesetzten Lösungsvorschläge von großer Bedeutung sind.

Für eine umfassende Einführung in Rails werden [Per10] und [Sch08] empfohlen.

¹Projekt-Homepage: <http://basecampHQ.com/>

2.1.1 Don't-Repeat-Yourself (DRY)

Zur Optimierung der Entwicklungsvorgänge innerhalb des Frameworks propagiert Rails den Grundsatz des DRY (Don't-Repeat-Yourself). Dabei sollen Redundanzen, d.h. die wiederholte Angabe identischer Informationen jeglicher Art vermieden werden. So kann sichergestellt werden, dass sich Änderungen an einer zentralen Stelle im System (z.B. Quellcode) in der gesamten Anwendung auswirken und Duplikate nicht mehrfach angepasst werden müssen.

2.1.2 Convention over Configuration

Viele Web Frameworks müssen vor ihrer Nutzung erst mit Hilfe zahlreicher Konfigurationsdateien und Parametereinstellungen zu einem lauffähigen Gesamtsystem zusammengebaut werden². Abbildung 2.1 zeigt beispielhaft solch eine Konfigurationsdatei innerhalb des Java Application Frameworks Spring³:

Quelltext 2.1: Konfigurationsdatei im Java Spring Framework

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:p="http://www.springframework.org/schema/p"
5       xmlns:context="http://www.springframework.org/schema/context"
6       xmlns:mvc="http://www.springframework.org/schema/mvc"
7       xsi:schemaLocation="http://www.springframework.org/schema/beans
8                           http://www.springframework.org/schema/beans/spring-beans.xsd
9                           http://www.springframework.org/schema/context
10                          http://www.springframework.org/schema/context/spring-context
11                          .xsd
12                          http://www.springframework.org/schema/mvc
13                          http://www.springframework.org/schema/mvc/spring-mvc.xsd">
14   <mvc:annotation-driven />
15   <mvc:view-controller path="/index.html" />
16   <bean id="messageSource" class="org.springframework.context.support.
17       ResourceBundleMessageSource"
18       p:basenames="messages" />
19   <!-- Declare the Interceptor -->
20   <mvc:interceptors>
21       <bean class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor"
22           p:paramName="locale" />
23   </mvc:interceptors>
```

²Rails bezeichnet diese Art der Frameworks mit ihrem Konfigurations-Overhead oft als *enterprisy*. Dies bedeutet jedoch nicht, das Rails für Anwendungsumsetzungen im Enterprise-Bereich ungeeignet ist. [vgl. Aim10].

³Projektseite: <http://www.springsource.org/>

Um diesen zusätzlichen und zeitraubenden Aufwand vor der eigentlichen Arbeit mit einem Framework zu vermeiden, definiert Rails zahlreiche Konventionen, die es erlauben, sofort mit der Entwicklungsarbeit zu beginnen. U.a. werden folgende Festlegungen getroffen:

- Informationen zur Datenbankverbindung der Anwendung müssen in der Datei `database.yml` im Unterordner `config` hinterlegt werden
- Der Klassenname eines Domainmodells wird im Singular erwartet, der dazu korrespondierende Tabellenname in der Datenbank hingegen im Plural z.B. Domainmodell `Project` => Datenbanktabelle `projects`
- Der Primärschlüssel in einer Datenbanktabelle muss vom Typ Integer sein und den Namen `ID` besitzen
- Rails erwartet eine definierte Ordnerstruktur für Controller, Domainmodell und Views⁴

Für den produktiven Einsatz des Frameworks müssen diese daher erlernt und akzeptiert werden, was dazu führt, dass Rails häufig als *opinionated software*⁵ (starr- und eigensinnige Software) bezeichnet wird.

Ein Abweichen von den definierten Konventionen ist jederzeit möglich, erhöht jedoch den Aufwand des Entwicklers.

2.1.3 Model-View-Controller (MVC)

Für Software-Designer ist es eine gebräuchliche Technik, komplexe Software-Systeme durch Schichtenbildung in einzelne Bestandteile zu zerlegen [Fow03, Kapitel 1]. Das Ruby on Rails Framework baut ebenfalls auf einem Mehr-Schichten-Architektur-Modell auf. Zusätzlich kommt eine für das Rails-Framework spezifische Implementierung des

⁴Eine Erklärung zu Model-View-Controller folgt in Abschnitt 2.1.3

⁵Eine ausführliche Stellungnahme von Rails-Erfinder David Heinemeier Hansson zu diesem Thema gibt es unter: <http://www.linuxjournal.com/article/8686>

bereits 1979 von dem Norweger Trygve Mikkjel Heyerdahl entwickelten Model-View-Controller-Paradigmas zum Einsatz⁶. Abbildung 2.1 charakterisiert den Ablauf einer Anfrage (Request) an eine Rails-Anwendung innerhalb des Client-Server-Modells sowie die Komponenten Model, View und Controller, die eine logische Trennung der Anwendung in verschiedene Verantwortungsbereiche ermöglichen:

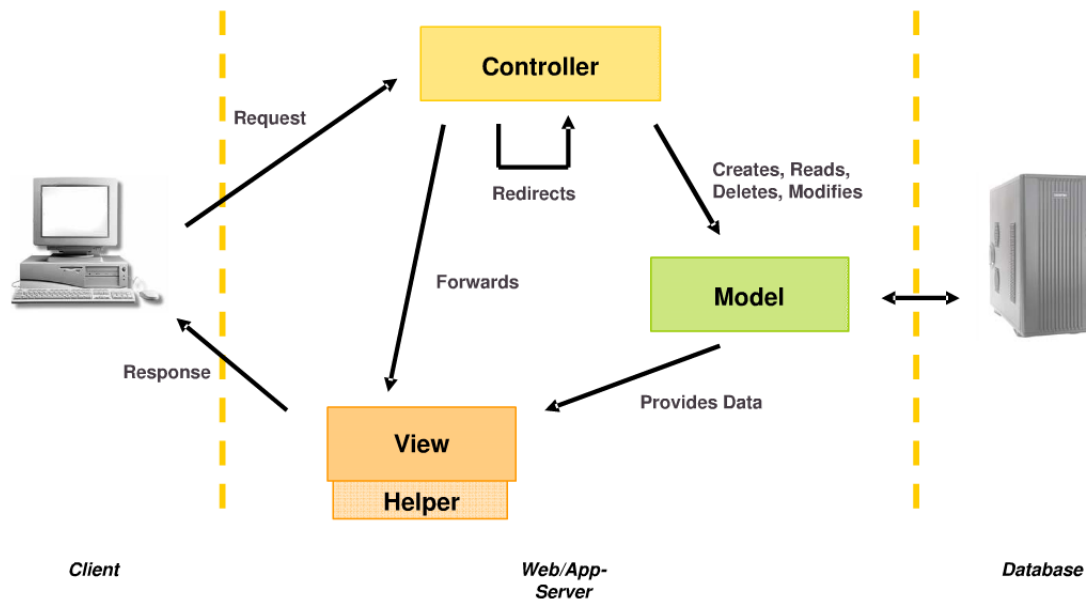


Abbildung 2.1: Verarbeitung einer Anfrage (Client-Server-Modell) innerhalb des Rails-Frameworks. Quelle: [Egg07, Seite 6]

1. Anfrage eines Clienten
2. An Hand der im Rails-Routing definierten Einträge wird die Anfrage eines Clienten an den registrierten Controller weitergeleitet
3. Der Controller steuert den Ablauf innerhalb der Anwendung. Dabei greift er über ein Model auf benötigte Daten in einem Speicher (z.B. relationale Datenbank) zu und stellt diese dem View-Layer zur Verfügung. Es ist auch möglich, dass der

⁶In der Fachliteratur wird das MVC-Paradigma häufig auf die Schichtenarchitektur einer Anwendung übertragen [BL06, S. 544 ff.]. Tatsächlich betrifft MVC in seiner ursprünglichen Form nur die Präsentationsschicht einer Webanwendung.

Controller die Anfrage an einen anderen Controller weiterleitet (Redirects). Im Gesamtkonzept von Rails enthält der Controller die Programmlogik der Anwendung.

4. Als Model werden in Rails Ruby-Klassen bezeichnet, die einen Zugriff auf relationale Datenbanken oder andere Datenspeicher ermöglichen [vgl. HM10]. Es bildet innerhalb der Anwendung die zugrundliegende Datenstruktur ab.
5. Der View-Layer (Sicht- oder Darstellungsschicht) bereitet die durch den Controller zur Verfügung gestellten Daten in der angeforderten Darstellungsform auf und gibt das Ergebnis der Anfrage aus. So wird je nach spezifiziertem Format z.B. eine HTML- oder XML-Datei erzeugt. Ein View repräsentiert somit die Darstellung einer bestimmten Datenstruktur (Model).
6. Das Ergebnis der Anfrage (die Ausgabe des View) wird vom Framework an den Server übermittelt und von dort an den Clienten ausgeliefert (Response). Der Kommunikationsprozess mit dem Rails-Framework ist damit abgeschlossen.

2.1.4 REST

Innerhalb einer Web-Applikation erfolgt der Austausch zwischen Server und Client durch die Nutzung des HTTP-Protokolls. Dabei wird eine Anfrage (Request) an einen Server geschickt, bearbeitet und eine entsprechende Antwort (Response) mit den angeforderten Inhalten zurückliefert. Ein Großteil der Webanwendungen interpretiert dabei die im HTTP-Protokoll definierten Methoden GET und POST:

GET Anforderung an den Server, eine über die Adresszeile des Browsers angegebene Ressource zurückzuliefern. Es können zusätzlich Argumente an die angeforderte URL angehängt werden.

POST Mit Hilfe dieser Methode ist es möglich, große Datenmengen aus z.B. Formularen an einen Webserver zu verschicken. Die übergebenen Informationen werden dabei im sogenannten Body der Anfrage codiert mitverschickt und somit im Vergleich zu GET nicht in der URL sichtbar⁷.

⁷Ein Post-Request findet vor allem bei der Übermittlung von Formularen innerhalb eines Browsers Verwendung.

REST, ein Akronym für **RE**presentational **S**tate **T**ransfer, erweitert die in traditionellen Webanwendungen üblichen GET und POST um die ebenfalls im HTTP-Standard enthaltenen Methoden PUT und DELETE:

PUT Die Verwendung der PUT-Methode zeigt die Neuanlage der in einer Anfrage spezifizierten Ressource an

DELETE Die Verwendung dieser Methode signalisiert dem Server, die angegebene Ressource auf dem Server zu löschen.

Da aktuelle Browser nur GET- und POST-Anfragen zuverlässig unterstützen, müssen entsprechende Anfragen zum Löschen und Verändern einer Ressource mit Hilfe von zusätzlichen Attributen in der Anfrage simuliert werden⁸. Das folgende Beispiel stellt das von Rails definierte Standard-Routing einer als *restful* angelegten Ressource Projekt dar⁹.

Tabelle 2.2 zeigt noch einmal die Trennung zwischen Ressource und Aktion innerhalb einer in Rails definierten REST-Ressource Projekt. Eine ausführliche Beschreibung von REST und dessen Realisierung liefert [Gü07].

⁸Das Rails Framework fügt innerhalb von Formularen automatisch einen versteckten Parameter `_method` in die Anfrage, die den Namen der gewünschten HTTP-Methode (DELETE oder PUT) enthält. Im Framework wird dieser Parameter ausgelesen und ein entsprechendes Routing zur geforderten Controller-Aktion eingeleitet.

⁹REST wird nicht über einen entsprechend formulierten Standard definiert. Vielmehr kann es als Programmierparadigma innerhalb von Web-Anwendungen verstanden werden, die eine Ansammlung von Best-Practices darstellen [Gü07].

Tabelle 2.1: Rails Routing der Rest-Ressource Projekt

HTTP-Methode	Anfragepfad	Ausgelöste Aktion im Controller	Wirkung
GET	/projects	index	Anzeige aller vorhandenen Projekte
GET	/projects/new	new	Anzeige eines HTML-Formulars zum erstellen eines neuen Projektes
POST	/projects	create	Erstellt ein neues Projekt mit den übermittelten Daten
GET	/projects/:id	show	Anzeige eines Projektes mit der zugeordneten ID
GET	/projects/:id/edit	edit	Anzeige eines HTML-Formulars zum Bearbeiten eines bestehenden Projektes
PUT	/projects/:id	update	Aktualisierung eines bestimmten Projektes mit den übermittelten Daten
DELETE	/projects/:id	destroy	Löschung des Projektes mit der angegebenen ID

Tabelle 2.2: Vergleich zwischen Rest-konformen und klassischen Rails-URL's

Aktion	normale URL	REST URL in Rails
show	/projects/show/12	/projects/12
delete	/projects/destroy/123	/projects/123
update	/projects/update/123	/projects/123
create	/projects/create	/projects

2.1.5 Rack und Middleware

Rails ist innerhalb der Ruby-Gemeinde nicht das einzigste existierende Webframework. Mit den Projekten Sinatra¹⁰, Merb¹¹, Camping¹² und Ramaze¹³ stehen weitere Alternativen mit unterschiedlichen Ansätzen und Funktionsumfängen zur Verfügung. Bei der Implementierung dieser Frameworks müssen Entwickler wiederholt Adapter¹⁴ (Handler) zur Ansteuerung verschiedener Webserver entwickeln. Durch Rack¹⁵, einem Ruby Webserver Interface, lässt sich die wiederholte Implementierung solcher Adapter in den einzelnen Frameworks vermeiden.

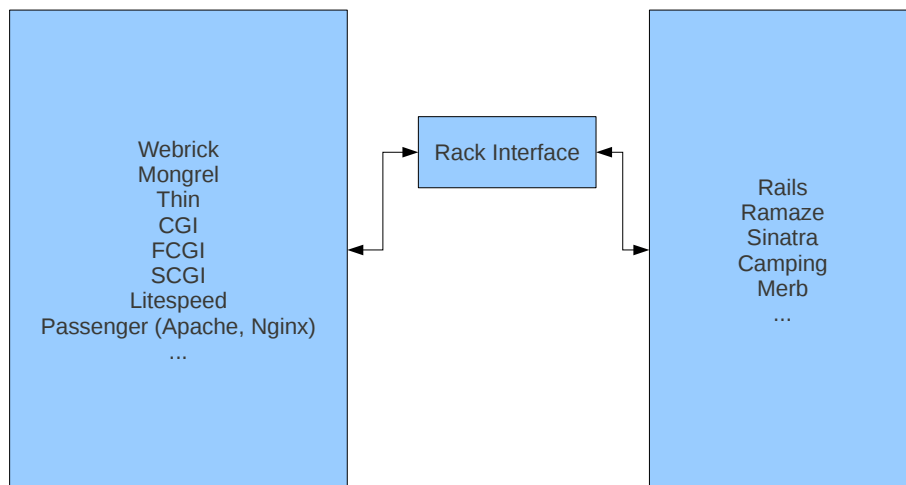


Abbildung 2.2: Rack als Vermittler zwischen Server und Ruby-Webframeworks

Damit Server und Frameworks untereinander kommunizieren können, muss eine Rack-Anwendung bestimmte Methoden implementieren und ein bestimmtes Rückgabeformat

¹⁰Projektseite: <http://www.sinatrarb.com/>

¹¹Projektseite: <http://www.merbivore.com/>

¹²Projektseite: <http://camping.rubyforge.org/>

¹³Projektseite: <http://ramaze.net/>

¹⁴

¹⁵Projektseite: <http://rack.rubyforge.org/>

einhalten. An Hand des Quelltext 2.2 sollen diese formalen Kriterien beschrieben werden:

Quelltext 2.2: Beispiel für eine einfache Rack-Anwendung

```
1 require 'rubygems'
2 require 'rack'
3
4 class MyRackApp
5
6   def initialize(name)
7     @name = name
8   end
9
10  def call(env)
11    [200, {"Content-Type" => "text/plain"}, ["Hello #{@name}!"]]
12  end
13 end
14
15 Rack::Handler::WEBrick.run MyRackApp.new("Rack"), :Port => 3001
```

Zeile 6-8

Initialisierung der Rack-Anwendung `MyRackApp` unter Angabe eines zuvor übergebenen Namen (*Rack* in Zeile 15).

Zeile 10-12

Implementierung der für eine Rack-Anwendung notwendigen Methode *call*. Diese wird vom Webserver beim Start einer Anfrage aufgerufen und muss als Antwort (Rückgabewert) einen Ruby-Hash mit den folgenden Informationen zurückliefern:

Status

Angabe des HTTP-Statuscode, *200* bedeutet in diesem Beispiel eine erfolgreiche Ausführung der Anfrage am Server¹⁶

Header

Angabe von im HTTP-Protokoll definierten Header-Informationen, im Beispiel wird ein einfaches Textdokument (*text/plain*) zurückgeliefert

Response

Die Antwort der Rack-Anwendung, im Beispiel wird ein String *Hello Rack!* an den Server übergeben und von diesem ausgeliefert.

¹⁶Die HTTP-Statuscodes sind im HTTP-Protokoll definiert und können u.a. unter folgender Quelle eingesehen werden: <http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>

Beim Aufruf der Methode *call* (Zeile 15) übergibt der Server eine Variable *env*, die alle wichtigen Informationen über die Serverumgebung und Anfrageparameter enthält.

Zeile 15

Start des Mongrel Webservers und Initialisierung der Rack-Anwendung *MyRackApp*.

Bei einem Aufruf der Adresse *localhost:3001* in einem Browser wird vom gestarteten Mongrel-Server der Rückgabewert der Methode *call* ausgegeben (Abb. 2.3).

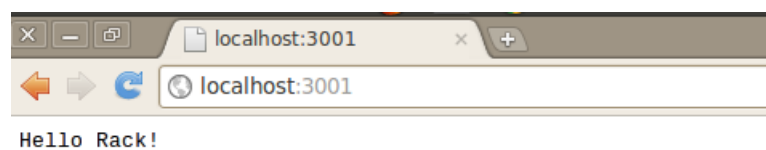


Abbildung 2.3: Ausgabe der Rack-Anwendung MyRackApp im Browser

Durch die Nutzung von Rack ergeben sich im Bereich der Ruby Webframeworks folgende Vorteile:

- Vereinfachung der Webframework-Entwicklung: Ein Framework, das Rack unterstützt, kann sofort von allen anderen rack-unterstützenden Webservern verwendet werden.
- Vereinfachung der Serverentwicklung: Ein rack-unterstützender Server kann sofort mit allen rack-basierten Webframeworks eingesetzt werden.
- Keine Codeduplizierung durch wiederholte Entwicklung von Server-Adaptern innerhalb der verschiedenen Ruby Webframeworks
- Rack beinhaltet Handler für die meist verbreitesten Webserver in Ruby (Abb. 2.2)

Darüber hinaus ist es möglich, einzelne Rack-Anwendungen hintereinander zu schalten und so ein Filtersystem für ankommende Anfragen und ausgehende Antworten vor dem eigentlichen Webframework zu realisieren. Diese Anwendungen werden Rack-Middlewares genannt und können in einer Rails-Anwendung in der Startkonfiguration

des Frameworks eingebunden werden. So kann das Ein- und Ausgabeverhalten des Railsframeworks gezielt verändert werden, da die einzelnen Rack-Anwendungen entscheiden, ob der nächste Filter (die nächste registrierte Rack-Middleware) oder eine vorzeitige Antwort an den Server geschickt werden soll.

Das Rails-Framework steht selbst am Ende der Filterliste und ist im Kontext von Rack selbst als Rack-Anwendung eingebunden (Abb. 2.4). Eine detaillierte Beschreibung von Rack und der Einrichtung eines komplexen Filtersystems¹⁷ in einer Rails-Anwendung liefern [Neu07] und [Rai11b].

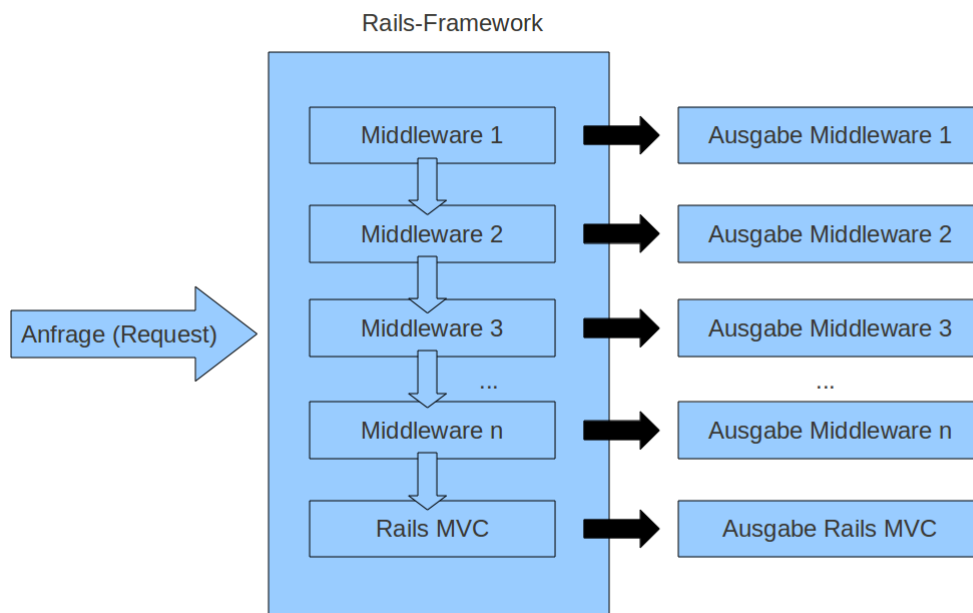


Abbildung 2.4: Möglichkeiten der unterschiedlichen Rückgabewerte durch Rack Middlewares und Rails

¹⁷Rack stellt häufig verwendete Middlewares für verschiedene Aufgabenbereiche zur Verfügung. Die Projektseite ist unter folgender Adresse verfügbar: <https://github.com/rack/rack-contrib>

2.1.6 Generatoren

Das Ruby on Rails Framework unterstützt mit Hilfe sogenannter Generatorskripte die automatische Erzeugung von funktionsfähigem Quellcode. Der folgende Scaffold-Generator¹⁸ erstellt z.B. nach dessen Initialisierung ein komplett funktionsbereites Codegerüst für die in Rails benötigten MVC-Komponenten einer Ressource *Projekt*.

Quelltext 2.3: Aufruf des Generators zur Erstellung einer MVC-Ressource Projekt mit Ausgabe der erstellten Dateien

```
1 rails g scaffold project name:string description:text
2
3 create db/migrate/20110926185944__create_projects.rb
4 create app/models/project.rb
5 create test/unit/project_test.rb
6 create test/fixtures/projects.yml
7 route resources :projects
8 create app/controllers/projects_controller.rb
9 create app/views/projects
10 create app/views/projects/index.html.erb
11 create app/views/projects/edit.html.erb
12 create app/views/projects/show.html.erb
13 create app/views/projects/new.html.erb
14 create app/views/projects/_form.html.erb
15 create test/functional/projects_controller_test.rb
16 create app/helpers/projects_helper.rb
17 create test/unit/helpers/projects_helper_test.rb
18 create app/assets/javascripts/projects.js.coffee
19 create app/assets/stylesheets/projects.css.scss
20 create app/assets/stylesheets/scaffolds.css.scss
```

2.2 Web Content Management, Content Life Cycle und Web-Publishing

Der Anstieg von digitalen Informationen und Content¹⁹ erfordert innerhalb der einzelnen Unternehmen ein immer umfangreicheres und zielgerichteteres Management. Der so entstandene Begriff des Content Management wird dabei u.a. von Brechtenbreiter mit folgenden Worten umschrieben:

¹⁸Scaffold kann in diesem Zusammenhang mit dem Wort Grundgerüst übersetzt werden.

¹⁹ „Unter dem Begriff des «Contents» werden alle Inhalte verstanden, die in einem CMS verwaltet werden und die im engeren Sinne für die Erstellung von Dokumenten und Publikationen verwendet werden. Darunter fallen alle textuellen und (audio-)visuellen Informationen [...] [PD11, S. 297]

Content Management beschreibt die Planung, Verwaltung, Steuerung und Koordination aller Aktivitäten, die auf den Content und dessen Präsentation in Unternehmen abstellen [Ber04, S. 212].

Im Bereich der internet-basierten Web-Sites und Internet-Portale hat dies zur Herausbildung des Web Content Managements geführt [Rig09, S. 3]. Die dort verwendeten Softwarelösungen (WCMS) streben dabei die Implementierung des Content Life Cycle (Inhaltslebenszyklus) ab, der als stark vereinfachtes Prozessmodell alle wichtige Phasen²⁰, die ein beliebiger Inhalt (Informationsträger) während seiner Existenz durchläuft, abbildet [PD11, S.303]. Im folgenden sollen diese Teilprozesse erläutert werden:

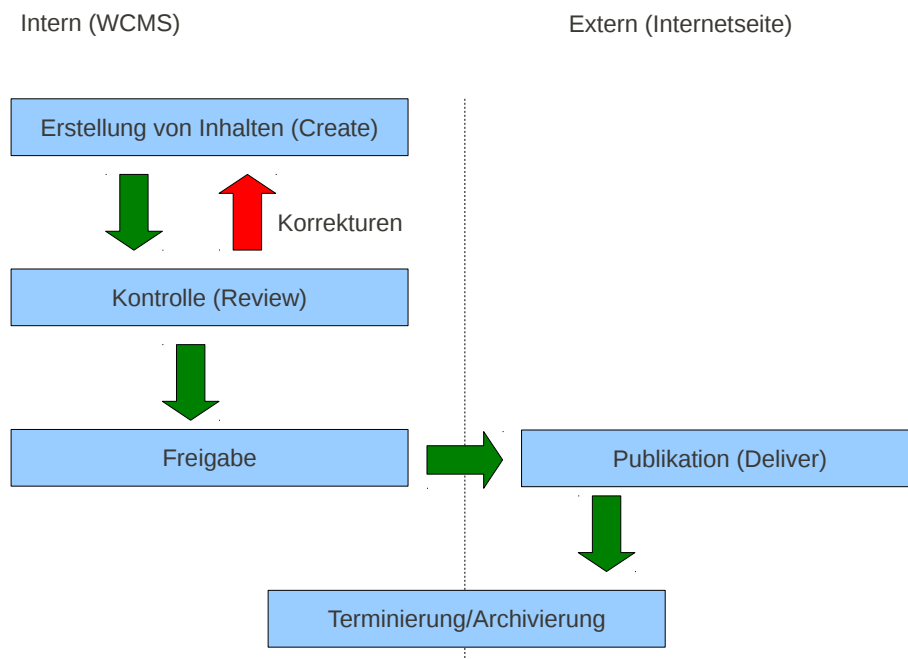


Abbildung 2.5: Prozessschritte im Content Life Cycle. Eigene Darstellung nach [Roc02, S. 81] und [Rit10, S. 10]

²⁰Die Content-Nutzung durch den Endanwender (Internetseitenutzer) findet innerhalb des Content Life Cycle keine Beachtung.

Erstellung

Autoren oder Benutzer erzeugen die Inhalte einer Internetseite an Hand der vorliegenden digitalen, (audio-)visuellen Medien oder anderer Informationsträger.

Freigabe und Kontrolle

Die Kontrolle von Content schließt sich dem Teilprozess der Content-Erstellung an und stellt durch verschiedene Redakteure und Arbeitsabläufe (Workflows) die Qualität der Inhalte, die auf der Internetseite veröffentlicht werden sollen, sicher. Die Komplexität der Kontrollinstanzen kann dabei unterschiedlich ausgeprägt sein und muss den tatsächlichen Gegebenheiten angepasst werden. Erfüllt der Content die inhaltlichen und gestalterischen Anforderungen nicht, muss dieser von den Autoren erneut überarbeitet werden.

Publikation

Nach erfolgter Freigabe des Content erfolgt im Teilprozess Publikation die Veröffentlichung auf der Internetseite. Damit werden die bis dato ausschließlich internen Informationen externen Nutzern zugänglich gemacht. Die eingesetzten WCMS unterscheiden dabei häufig zwischen einer Inter-, Intra- oder Extranetveröffentlichung, welche die Erreichbarkeit der Inhalte auf bestimmte Zielgruppen einschränkt.

Terminierung und Archivierung

Nach Verwendung des Content auf der Internetseite werden bestimmte Inhalte mit steigendem Alter uninteressant oder überflüssig (abhängig von der Art des Inhalts). Durch Aufnahme in ein internes oder öffentliches Archiv können diese für eine spätere Nutzung bzw. Wiederverwendung aufbewahrt werden.

Hauptziel dieses gesamten Redaktionsprozesses ist die Veröffentlichung des Content auf verschiedenen Internetseiten. Der Content Life Cycle bildet somit die wichtigsten Komponenten zur Umsetzung von Web-Publishing, dem Publizieren von Inhalten verschiedenster Medien im Internet, ab. Im folgenden Abschnitt wird der für die funktionale Analyse verwendete Kriterienkatalog vorgestellt. Dabei werden die formulierten Funktionsbeschreibungen nach den Teilprozessen des Content Life Cycle unterteilt.

2.3 Kriterienkatalog

Die hohe Zahl am Markt befindlicher Web Content Management Systeme führt zu einem erschwerten Auswahlverfahren. Neben kommerziellen WCMS-Lösungen stehen durch die Open Source Bewegung zusätzlich zahlreiche kostenlose Softwareprodukte zur Verfügung, die sich in ihrer Leistungsfähigkeit stark unterscheiden. So kommt es häufig vor, das klein angelegte Open Source Projekte ihre Software stolz als Web Content Management System bezeichnen, obwohl nur sehr wenige Funktionalitäten implementiert sind. Um dieser Praktik entgegenzuwirken, haben Vertreter der Content Management Branche eine Feature-Matrix (Abb. 2.6) herausgegeben, die aktuelle Anforderungen an ein Content Management System spezifizieren soll. Die dabei entstandene Übersicht zeigt dabei eine Unterscheidung in 3 Prioritätsstufen:

Must-Have

Diese Funktionalität muss in einem Web Content Management System verhanden sein.

Should-Have

Diese Funktionalität ist nicht zwingend notwendig, kann bei entsprechender Existenz aber sehr positiv wahrgenommen werden.

Nice-to-Have

Funktionalitäten, die nur in wenigen, hochwertigen Systemen zur Verfügung stehen und über die gewöhnlichen Anforderungen hinausgehen.

Die beschriebenen Funktionalitäten sind jedoch teilweise so allgemein formuliert, das dieser Ansatz nur als grobe Orientierungshilfe dienen kann. Der Wirtschaftsinformatiker Andreas Ritter hat daher im Rahmen seiner Bachelorarbeit *SWOT-Analyse zu Content-Management-Systemen* [Rit10] Bereichskriterien erarbeitet, an Hand deren die Leistungsfähigkeit von Web Content Management Systemen untersucht werden kann [Rit10, Seite 21-23]. Auf Grundlage dieser Bereichskriterien und der aufgeführten WCMS-Featurematrix wird in den Kapiteln 2.3.1 bis 2.3.5 ein Kriterienkatalog vorgestellt, der allgemeine Funktionalitätsanforderungen unterteilt nach den einzelnen Teilprozessen des Content Life Cycle (Erstellung, Kontrolle, Freigabe, Publikation, Archivierung) formuliert.



Abbildung 2.6: Feature-Matrix für Web Content Management Systeme. Quelle: Eigene Darstellung nach [jdk06].

2.3.1 Erstellung

- In einem WCMS sollen mehrere Redakteure Inhalte gleichzeitig erstellen, ändern, löschen und verwalten können.
- In einem WCMS sollen Inhalte – unabhängig von Zeit und Standort – durch mehrere Benutzer online verwaltet und erfasst werden können.
- In einem WCMS soll eine Offline-Erfassung von Inhalten unter Verwendung eines lokal auf dem Rechner installierten Programms möglich sein.
- Das WCMS verfügt über eine integrierte Mediendatenbank zur Erfassung und Verwaltung von Bildern, Multimedia, Texten, Audio, Videos, usw. Die Inhalte werden dabei in einer Datenbank gespeichert.

- Inhalte sollen in einem WCMS ohne spezielle Programmier- und HTML-Kenntnisse erfasst und verwaltet werden können.
- Die Nutzung des WCMS erfolgt über einen Internet-Browser. Dabei können alle gängigen Internet-Browser (Internet Explorer, Safari und Firefox) eingesetzt werden.
- Ein WCMS soll Inhalte mehrsprachig erfassen und verwalten können.
- Inhalte können in einem WCMS während der Erfassung über eine Preview-Funktion vorab im Design der Webseite betrachtet werden.
- Das WCMS ermöglicht eine Zuordnung von standardisierten und frei definierbaren Metadaten zu beliebigen Inhalten (z.B. Autor, Schlüsselwörter, benutzerdefinierte Felder).
- Das CMS soll über eine offene API (Programmierschnittstelle) für individuelle Erweiterungen verfügen.
- Das WCMS ermöglicht die Integration von Inhalten anderer Webseiten, Applikationen oder E-Commerce- Tools.
- In einem WCMS sollen Inhalte einfach importiert und exportiert werden können. Beim Austausch kommen Formate wie z.B. XML zum Einsatz.

2.3.2 Kontrolle

- Das WCMS verfügt über ein granulares Rechte- und Rollenkonzept für Anwender, Inhalte, Module (Plugins) und Webseiten.
- Das WCMS ermöglicht die Versionierung von Inhalten. Zusätzlich können vorhergehende Zustände/Versionen mit Hilfe einer Wiederherstellungsfunktion rekonstruiert werden.
- Das WCMS bietet einen Schutz vor gegenseitigem Überschreiben erfasster Inhalte durch z.B. Check in/ Check out- Mechanismen.

- Das WCMS ist mandantenfähig, d.h. eine Mehrfachnutzung des Systems durch verschiedene Parteien mit kompletter Trennung der Daten und Benutzer ist möglich.
- Das WCMS bietet eine Linküberprüfung, die eine korrekte Darstellung von internen und externen Links auf der Internetseite sicherstellt.

2.3.3 Freigabe

- Mit Hilfe des WCMS können *nicht technische* User den Workflowprozesse kreieren, verwalten und ändern. Es sollen dafür keine Programmierkenntnisse notwendig sein.
- Das WCMS bildet einen mehrstufigen Workflowprozess für die Freischaltung von Inhalten ab.
- Das WCMS bietet die Möglichkeit, externe Mitarbeiter in Workflowprozesse einzubinden.
- Unternehmensspezifische Bearbeitungsprozesse von Inhalten sollen über frei definierbare Workflows verwaltet werden können.

2.3.4 Publikation

- Das WCMS trennt Inhalt und Design durch die Verwendung von Templates.
- Das WCMS erlaubt die Mehrfachverwendung von Inhalten an verschiedenen Stellen (auf unterschiedlichen Seiten). Zusätzlich können angelegte Seiten kopiert werden.
- Navigationsstrukturen werden vom WCMS automatisch generiert, publiziert und verwaltet.
- Die vom WCMS erstellten Seiten können barrierefrei umgesetzt werden.
- Inhalte sollen vom WCMS auf verschiedene Medien / Technologien (Cross Media Publishing, SMS / Mobile / WAP / usw.) ausgegeben werden können.

- Das WCMS bietet Möglichkeiten, Inhalte für anderen Webseiten bereitzustellen (z.B. XML, Webservice).
- Das WCMS ermöglicht die Wahl zwischen dynamischer oder statischer Generierung der Seiten bzw. Inhalte (Caching).
- Das WCMS unterstützt die automatische Erstellung einer Druckversion für jede einzelne Seite.

2.3.5 Terminierung und Archivierung

- Das WCMS erlaubt die freie Wahl des Publikationszeitraumes (zeitgesteuertes Auf- / Abschalten / Archivieren) von Inhalten.
- Im WCMS können Inhalte und Seiten archiviert werden.
- Das WCMS ermöglicht eine Durchsuchung der archivierten Inhalte und Seiten nach wählbaren Parametern (z.B. Monat oder Jahr).

3 Analyse des Leistungsfähigkeit von Ruby on Rails Web Content Management Systemen

3.1 Vorbetrachtung

Der 2006 eingeleitete Hype¹ um das Ruby on Rails Framework hat dazu geführt, dass viele Entwickler mit der Konzeption und Umsetzung zahlreicher verschiedener Rails-Anwendungen begonnen haben. So entstanden auch im Bereich der Web Content Management Systeme zahlreiche Projekte. Ein Großteil der Vorhaben blieb jedoch in der Konzeptionsphase stecken oder die Entwicklung wurde nach wenigen Jahren eingestellt². Die Ursachen sind dabei vor allem dem Entwicklungsumfeld von Rails und dem Framework selbst geschuldet:

Schnellebigkeit

Die Entwicklung des Ruby on Rails Frameworks unterliegt einem ständigen Wandel und erfordert eine ständige Anpassung des Programmierers an neue Technologien und Konzepte.

Interessenwandlung der Entwickler

Die Programmierung mit Ruby und die vielfältigen Möglichkeiten des Ruby on Rails Frameworks erleichtern die Umsetzung verschiedenster Projektideen. Es ist daher schneller möglich, dass sich Entwickler nach einiger Zeit mit anderen Pro-

¹Durch verstärkte Vortragsreihen, zahlreiche Buchveröffentlichungen und erfolgreich eingesetzte Rails-Anwendungen gilt das Jahr 2006 als Startpunkt für eine verstärkte Wahrnehmung von Ruby on Rails.

²Im Anhang der Arbeit findet sich eine Übersicht zu ermittelten Rails 2 und 3 Web Content Management Systemen. Diese sind zum Teil seit einigen Jahren nicht mehr weiterentwickelt wurden.

jekten beschäftigen und bereits bestehende bzw. veröffentlichte Projekte vernachlässigen.

Die Realisierung eines stabilen Web Content Management Systems auf Basis von Ruby on Rails wird durch diese Faktoren erschwert und erfordert ein entsprechend tragfähiges Konzept sowie planendes Vorgehen der Projektinitiatoren.

Die hier aufgeführten ProjekteAlchemy CMS, Browser CMS, Locomotive CMS und Refinery CMS repräsentieren daher die zum Zeitpunkt der Erstellung dieser Arbeit vielversprechendsten Rails-Implementierungen eines Web Content Management Systems.

Um die Zahl der zu untersuchenden WCMS einzuschränken, wurden folgende Mindestanforderungen für existierende Rails-Anwendungen festgelegt:

Open Source Software

Die hier ermittelten WCM-Systeme sind vollständige Open Source Lösungen. Ihre Veröffentlichung unterliegt dabei den in der Open Source Bewegung üblichen Lizenzen der freien Software bzw. der Open Source Initiative (OSI).

Rails 3 Kompatibilität

Die Veröffentlichung von Rails 3 brachte vielen Verbesserungen hinsichtlich der Modularität von Rails-Anwendungen. Kern-Komponenten des Frameworks (z.B. die Persistenz-Schicht Active Record) können nun mit geringem Aufwand gegen andere Implementierungen ausgetauscht werden. Die so erreichte Flexibilität soll auch bei der Integration eines Rails WCM-Systems zur Verfügung stehen. Weiterhin sichert die Verwendung der aktuellsten Framework-Version die Unterstützung moderner Technologien und Entwicklungen innerhalb der implementierten WCMS ab.

Projektaktivität

Die kontinuierliche Entwicklung an einem Open Source Projekt ist ein Merkmal für die Akzeptanz einer Software. Zusätzlich spiegelt sie das Engagement der beteiligten Programmierer wider. Ein in diesem Umfeld entstehendes Softwareprodukt bietet daher ein entsprechend höheres Potenzial. Die hier ausgewählten Systeme erfüllen diese Forderung und zeichnen sich durch regelmäßige Verbesserungen des Quellcodes aus.

Unterscheidung in Front- und Backend

WCMS unterscheiden zwischen Frontend- und Backend-Funktionalität. Das Frontend wird durch die eigentliche Internetseite repräsentiert, die mit Hilfe des Systems erzeugt wird. Im Backend können Anwender Inhalte zentral einpflegen und verwalten. Die hier ausgewählten und untersuchten Implementierungen verfügen über eine solche konzeptionelle Trennung.

3.2 Bezugsquellen

Trotz der steigenden Bekanntheit von Ruby on Rails existiert zum Zeitpunkt der Anfertigung dieser Arbeit keine Fachliteratur, die sich mit den Möglichkeiten des Web Content Managements in Rails auseinandersetzt. Vielmehr sind folgende Schwerpunktsetzungen bei den verschiedenen Autoren festzustellen:

Grundlagenbücher

Sie dienen als Einführung in Rails und verdeutlichen an Hand einfacher Anwendungen die Arbeitsweise mit dem Ruby on Rails Framework.

Fortgeschrittene Techniken mit Ruby on Rails

Rails Kern-Entwickler stellen ihre Erfahrungen mit dem Framework dar und geben Lösungsansätze für größere Unternehmensstrukturen und Projekte. Häufig vertiefte Themen sind dabei Skalierung, Performance und Refactoring.

Zur Ermittlung existierender Ruby on Rails Open Source WCMS-Software mussten daher alternative Informationsquellen herangezogen werden, die im folgenden kurz beschrieben werden sollen:

Anfragen im offiziellen IRC³ Channel von Ruby on Rails

Der Ruby on Rails IRC Channel ermöglicht einen konstruktiven Austausch von Rails-Entwicklern zu verschiedenen Bereichen des Rails-Frameworks. Mit der Hilfe mehrerer hundert Nutzer täglich können so Probleme und Anfragen sehr umfassend beantwortet werden. Der Ruby on Rails Channel ist erreichbar unter `#rubyonrails`.

RubyGems.org

Bibliotheken können die Funktionalität von Ruby enorm erhöhen. Zur Verbreitung dieser im Internet existiert u.a. der Ruby Online Community Anbieter RubyGems.org, der über 30.000 Erweiterungspakete verschiedenster Entwickler im

Internet zum Download anbietet. Der Dienst verfügt über eine ausführliche Suchfunktion, mit der gezielt nach bestimmten Bibliotheken gesucht werden kann. Neben einer kurzen Projektbeschreibung und Informationen zum Entwickler wird jedem Projekt ein Datum der letzten Aktualisierung zugeordnet. Der Entwicklungsstand eines Pakets kann so besser eingeschätzt werden. RubyGems.org stellte daher eine wichtige Informationsquelle bei der Suche nach vorhandenen Ruby on Rails WCMS dar.

rails 3.1.1 **1**

Ruby on Rails is a full-stack web framework optimized for programmer happiness and sustainable productivity. It encourages beautiful code by favoring convention over configuration.

INSTALL > `gem install rails`

Download Subscribe Stats

Authors
David Heinemeier Hansson **2** **5,927,779** **73,635**
total downloads for this version

Owners

Links [Homepage](#) [Source Code](#) [Documentation](#) [Wiki](#) [Mailing List](#) [Bug Tracker](#)

Bundler

```
gem "rails", "~> 3.1.1"
```

Versions

3.1.1	October 7, 2011
3.1.1.rc3	October 5, 2011
3.1.1.rc2	September 29, 2011
3.1.1.rc1	September 14, 2011
3.1.0	August 31, 2011

[Show all versions \(109 total\)](#)

Runtime Dependencies

actionmailer	= 3.1.1
actionpack	= 3.1.1
activerecord	= 3.1.1
activesource	= 3.1.1
activesupport	= 3.1.1
bundler	> 1.0
railties	= 3.1.1

3

Abbildung 3.1: Ruby Gems mit Informationen zum aktuellen Rails 3.1.1. Dabei werden u.a. eine Projektbeschreibung (1), eine Autorenübersicht (2) und existierende Versionen (3) der Software angezeigt.

Github.com

Github.com ist ein Online-Netzwerk und Webhosting-Dienst für Programmierer. Nutzer können dort ihre individuellen Programme, umgesetzt in beliebigen Programmiersprachen, kostenlos⁴ veröffentlichen. Schlüsseltechnologie des Netzwerkes ist das Versionsverwaltungssystem Git⁵, welches Änderungen am Quellcode des Projektes festhält. Zusätzlich erlaubt es anderen Programmierern, bestehende Projekte zu kopieren und eigenständig weiterzuentwickeln. Anschließend können diese wieder zu einem Gesamtprojekt verschmolzen werden. Die Symbiose zwischen den Funktionalitäten von Git und den zahlreichen Interaktionsmöglichkeiten der Plattform erschaffen eine Entwicklungsumgebung, in der ein schneller und effektiver Austausch zwischen Programmierern und Projekten stattfinden kann. Innerhalb der Ruby on Rails Community hat sich Github zu einer zentralen Anlaufstelle für Rails-Programmierer entwickelt und besitzt daher zur Ermittlung bestehender Web Content Management Systeme entscheidende Relevanz.

3.3 Alchemy CMS

Unter der Leitung der Hamburger Firma *macabi* wurde 2007 die proprietäre CMS Software *WashAPP* veröffentlicht. Nach der Insolvenz der Entwickler wurde das System zu nächst weiterverkauft (dabei erfolgte die Umbenennung in *Webmate*), bevor es 2010 letztendlich als Open Source Software Alchemy CMS an die Öffentlichkeit übergeben wurde. Die Weiterentwicklung übernimmt seitdem die Hamburger Internetagentur *magiclabs*⁶ um die Entwickler Thomas von Deyen, Robin Böning und Carsten Fregin. Die aktuelle Version 1.6.0 steht als Rails 2 und 3 Umsetzung⁷ zur Verfügung. Tabelle 3.1 fasst die wichtigsten Eckdaten und Informationsquellen zu Alchemy CMS zusammen.

3.3.1 Funktionsprinzipien

Das Backend von Alchemy CMS (Abb. 3.2) verfügt u.a. über die Bereiche Seiten, Sprachen, Benutzer und Bibliothek, die im folgenden übersichtsmäßig vorgestellt werden sollen:

⁴Die Einrichtung eines kostenpflichtigen, privaten Github-Repositories ist ebenfalls möglich.

⁵Projektseite: <http://git-scm.com/>

⁶Homepage: <http://magiclabs.de/home>

⁷Der Rails 3-Entwicklungszeitraum von Alchemy befindet sich noch im Beta-Stadium.

Tabelle 3.1: Steckbrief Alchemy CMS

Aktuelle Version	1.6.0	
Lizenz	GPLv3	
Projektseite	http://alchemy-cms.com https://github.com/magiclabs/alchemy	
Quellcode	https://github.com/magiclabs/alchemy	
IRC-Channel	nicht vorhanden	
API Dokumentation	nicht verfügbar	
Forum	http://groups.google.com/group/alchemy-cms	
Demoversion	Frontend	http://demo.alchemy-cms.com
	Backend	http://demo.alchemy-cms.com/admin
	Login	demo
	Passwort	demo
Verwendete Technologien	Ruby on Rails 3.0.x, HTML, jQuery und jQueryUI, TinyMCE - JavaScript WYSIWYG Editor, SWFUpload	
Philosophie	Der Benutzer des Systems muss nur Inhalte erstellen und ändern können Formatierung von Überschriften, Bildpositionierung und -berechnung sind Aufgaben des Entwicklers, nicht die des Redakteurs	
Zielgruppe	Privatnutzer, Kleinstunternehmen	

Home

Nach einer erfolgreichen Anmeldung im Backend bildet das Home-Modul die Startseite des WCMS, in der der Nutzer über die letzten Aktivitäten des Systems informiert wird (z.B. Auflistung der zuletzt editierten Inhalte).

Seiten

Das Seiten-Modul ermöglicht die Verwaltung und Bearbeitung aller im System vorhandener Seiten und deren Inhalte. darüber hinaus kann der von Alchemy CMS verwaltete Seitenbaum angezeigt und einzelne Seiten verschoben werden. Im Seitenbearbeitungsmodus kann die Backend-Ansicht in einen Vorschaubereich, der die aktuell ausgewählte Seite mit ihren tatsächlichen Inhalten anzeigt, und einem Elemente-Bereich, der die auf der Seite verfügbaren Inhalselemente angibt und editierbar macht, aufgeteilt werden (Abb. 3.2).

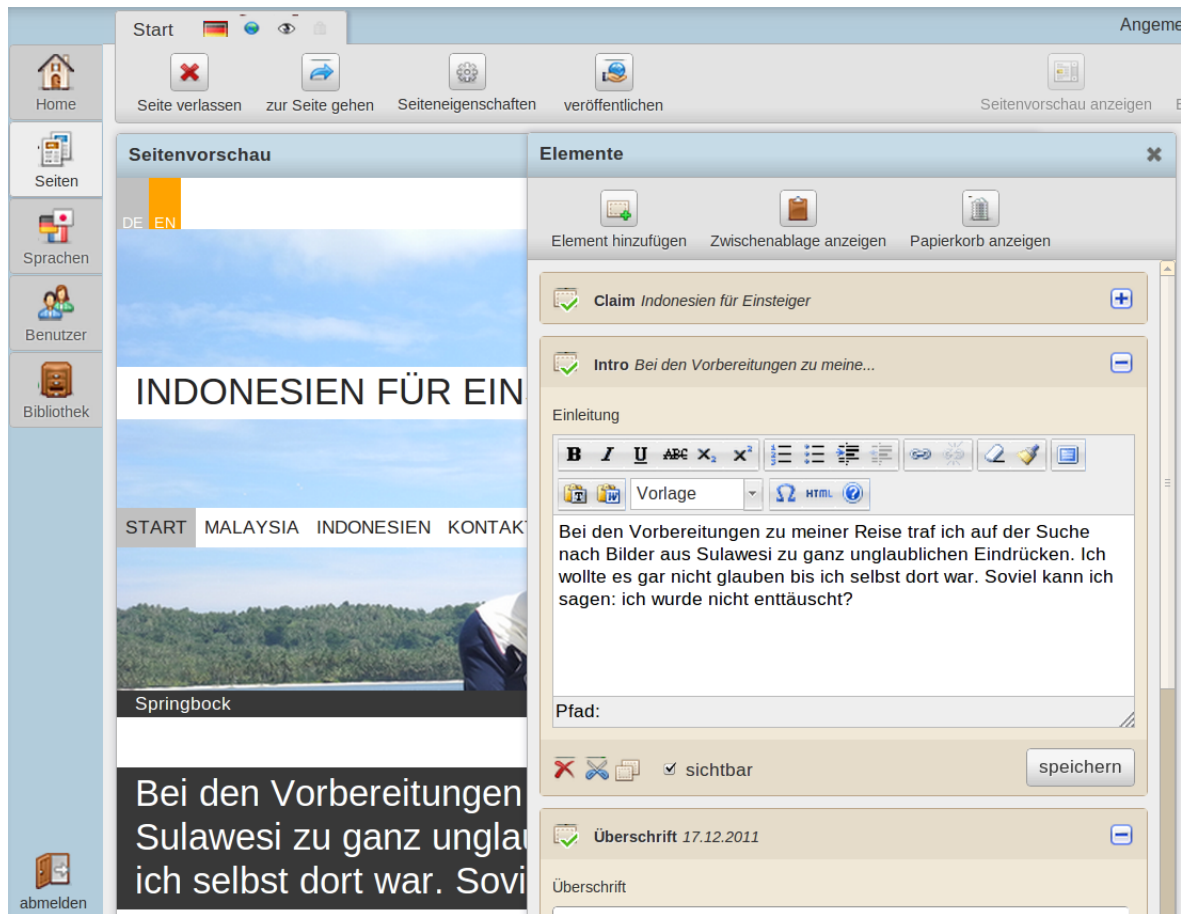


Abbildung 3.2: Backend-Ansicht von Alchemy CMS mit geöffneter Seitenvorschau und Elementebearbeitung (Seitenbearbeitungsmodus)

Sprachen

Dieses Modul ermöglicht die komfortable Verwaltung der in Alchemy CMS verfügbaren Frontend-Sprachen. Eine Alchemy Standard-Installation enthält bereits die Sprachen Deutsch und Englisch. Internetauftritte können somit jederzeit um zusätzliche Sprachversionen erweitert werden.

Benutzer

Das Benutzer-Modul ist die zentrale Anlaufstelle zur Verwaltung der am System registrierten Nutzergruppen (Administratoren, Redakteure u.a.) und ihrer jeweiligen Berechtigungen innerhalb des WCMS.

Bibliothek

Bilder und andere Dateien werden in Alchemy CMS über das Bibliotheken-Modul verwaltet. Es ermöglicht die Auflistung aller im System zur Verfügung stehenden Ressourcen. Zusätzlich stehen Funktionen zum Hochladen, Editieren und Löschen zur Verfügung.

3.3.2 Erweiterungen

Alchemy kann durch die Erstellung von Plugins in seiner Funktionalität erweitert werden. An Hand einer definierten API wird ein Grundgerüst angeboten, mit deren Hilfe Erweiterungen bequem in alle Teile des Systems integriert werden können. Ein Plugin kann so entweder als neues Backend-Modul umgesetzt werden (es erscheint in Form eines neuen Menüeintrages im linken Bereich des Backends) oder neue Inhaltselemente zur Verfügung stellen, die dann innerhalb der Seitenbearbeitung als auswählbarer Inhaltstyp zur Verfügung stehen (Abb. 3.3).

Folgende offiziellen Erweiterungen sind ebenfalls verfügbar:

- alchemy-mailings⁸: Ermöglicht die Erstellung und Verwaltung von Newsletters im Backend des Systems.
- alchemy-standard-set⁹: Enthält CSS-Formatierungen für die in Alchemy verfügbaren Standard-Inhaltselemente.

3.3.3 Verwendete Technologien

Schwerpunkttechnologien der Nutzeroberfläche von Alchemy CMS sind die an Hand von Rails erzeugten HTML-Views und darin eingebundene JavaScript-Dateien, die mit Hilfe der JavaScript-Bibliothek jQuery erstellt wurden. Das Seiten-Modul greift zusätzlich auf die jQuery-Erweiterung jQuery UI¹⁰ zurück, die vorgefertigte Elemente zur Erstellung von Dialogboxen und dem Tabulator-Menü liefert. Der bei einigen Inhaltselementen verfügbare WYSIWYG-JavaScript Editor TinyMCE ermöglicht die Formatierung einzelner

⁸Komponenten-Download: <https://github.com/magiclabs/alchemy-mailings>

⁹Komponenten-Download: <https://github.com/magiclabs/alchemy-standard-set>

¹⁰jQuery user interface

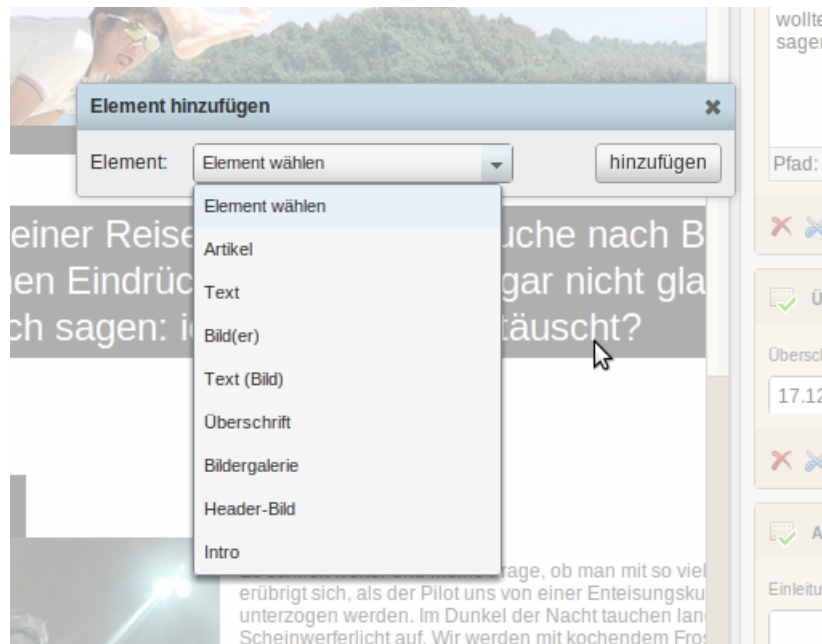


Abbildung 3.3: Standardset von Inhaltselementen in Alchemy CMS

Inhalte und das Einfügen von vorgefertigtem HTML. Ein Hochladen von Ressourcen erfolgt in Alchemy über ein herkömmliches HTML Upload-Feld oder den integrierten Adobe Flash Uploader, der somit auch das Einstellen mehrerer Medien in einem Schritt ermöglicht.

3.4 Browser CMS

Browser CMS ist ein Open Source Web Content Management System der US-amerikanischen Agentur BrowserMedia mit Sitz in Bethesda, Maryland. Im Gegensatz zum aktuellen Versionszweig 3.x des Systems wurden die Vorgängerversionen (Versionen 1.x und 2.x) mit Hilfe der Java Programmiersprache umgesetzt. Auf Grund geänderter Marktsituation entschied sich BrowserMedia jedoch 2009 dazu, alle weiteren Versionen des WCMS auf Grundlage des Rails Frameworks¹¹ zu entwickeln. Zusätzlich erfolgte der Umstieg auf die GPLv3-Lizenz, die eine freie Verwendung der Software ermöglicht. Eine Online-Testversion des Systems wird von BrowserMedia nicht angeboten. Um dennoch die Funktionalität des Systems zu demonstrieren, wurde eine Testinstallation von Browser CMS 3.3.1 eingerichtet¹².

Tabelle 3.2: Steckbrief Browser CMS

Aktuelle Version	3.3.1	
Lizenz	GPLv3	
Projektseite	http://browsercms.org	
Quellcode	https://github.com/browsermedia/browsercms	
IRC-Channel	nicht vorhanden	
API Dokumentation	http://rubydoc.info/gems/browsercms/	
Forum	http://groups.google.com/group/browsercms	
Demoversion	Frontend	http://diplomabcms.herokuapp.com/
	Backend	http://diplomabcms.herokuapp.com/admin
	Login	demo
	Passwort	demo
Verwendete Technologien	Ruby on Rails 3.0.x, HTML, jQuery und jQueryUI, diverse jQuery Plugins ,WYSIWYG-HTML-Editor CKEditor	
Zielgruppe	Privatnutzer, Kleine und mittelständige Unternehmen mit einer größeren Anzahl von Redakteuren	

¹¹Ausführliche Informationen zum Übergang von Java auf Rails können unter folgender Quelle nachvollzogen werden: <http://tinyurl.com/6eayba7>

¹²Die Upload-Funktion der Testversion wurde auf Grund zu geringer Server-Ressourcen deaktiviert.

3.4.1 Funktionsprinzipien

Innerhalb des Backend (Abb. 3.4) von Browser CMS kann der angemeldete Nutzer auf alle wichtigen Funktionsbereiche des Systems zugreifen:

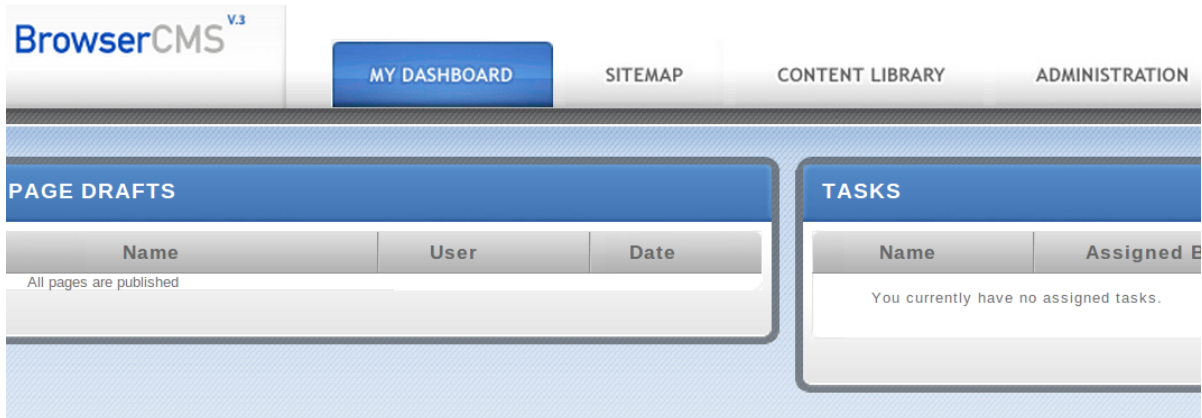


Abbildung 3.4: Backend-Ansicht von Browser CMS

Dashboard

Das Dashboard gibt eine Übersicht über die zuletzt vom Nutzer bearbeiteten Seiten sowie eine Auflistung der vom Anwender noch zu erledigenden Aufgaben.

Sitemap

Das Sitemap-Menü ermöglicht die Darstellung aller im System existierenden Seiten in einer Baumstruktur sowie die Bearbeitung aller Informationen zu einer einzelnen Seite. U.a. kann der Titel, das verwendete Template und verschiedene Meta-Informationen verändert werden.

Content Library

In der Content Library von Browser CMS können alle existierenden Inhaltselemente sortiert nach Inhaltstyp (z.B. Text, File, Image) aufgelistet und bearbeitet werden.

Administration

Im Administrator-Bereich des Systems werden alle Gruppen und Nutzer des Backend und der Internetseite verwaltet. Zusätzlich können die für die einzelnen Seiten und Inhaltselemente definierten Templates aufgelistet und bearbeitet werden.

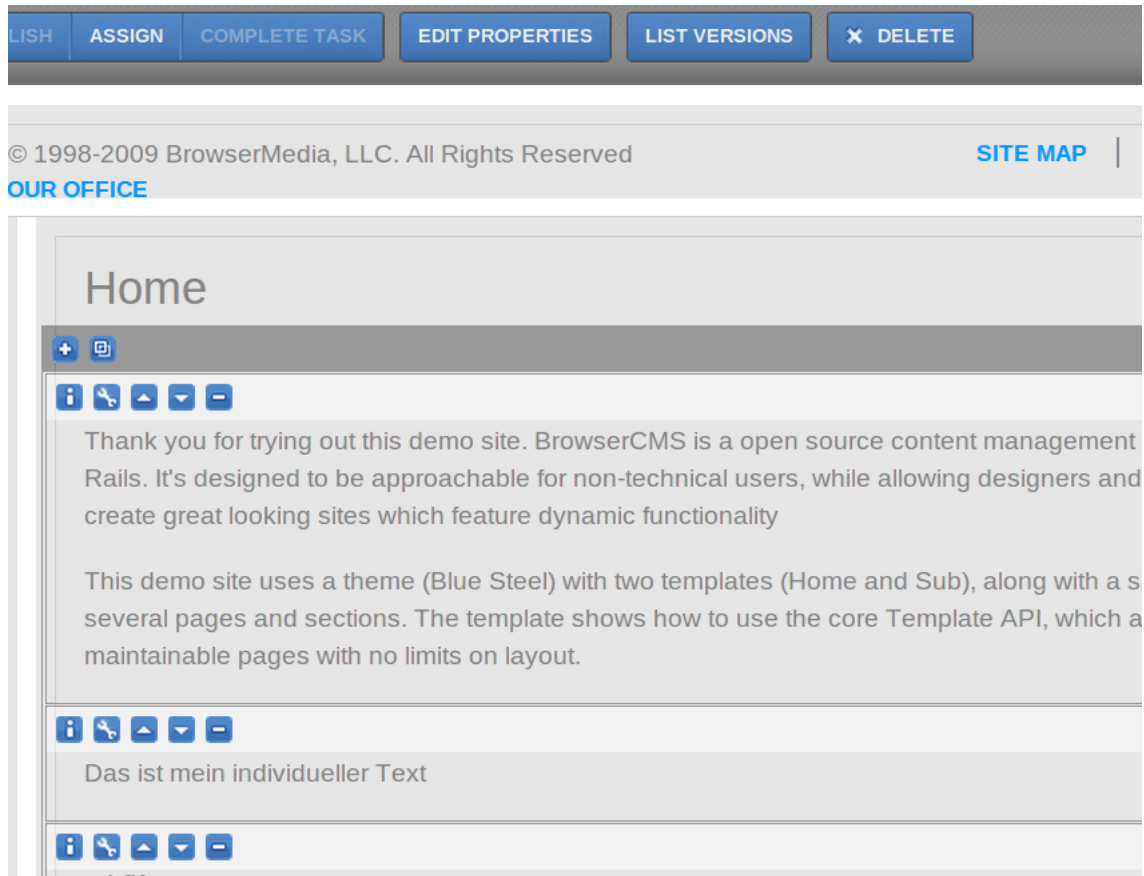


Abbildung 3.5: Ausgewählte Seite im Bearbeitungsmodus und aktiviertem Visual Editor

Die Erstellung und Anzeige von Inhalten wird in Browser CMS durch folgende 2 Prinzipien realisiert:

Content Blocks

Content Blocks sind Sammlungen von Daten mit einer definierten Anzahl an Feldern eines bestimmten Typs (z.B. Textfeld, Datumsfeld). Innerhalb der Content Library können diese verwaltet und angelegt werden.

Portlet

Portlets¹³ sind spezielle Content Blocks, die festlegen, wie bereits existierende Con-

¹³Der Begriff des Portlets stammt aus der Java-Welt und bezeichnet ursprünglich beliebig kombinierbare Komponenten unterschiedlichster Funktionalität, die an verschiedenen Positionen einer Internet- bzw. Portal-Seite dargestellt werden können.

tent Blocks (z.B. Text, News) dargestellt werden sollen. Dazu verfügt jedes Portlet über ein im Backend editierbares Template, das die zuvor ausgewählten Datensätze in dem gewünschten Layout präsentiert (Abb. 3.6).

Edit Portlet 'Tag Template Example'

PUBLISH VIEW CONTENT **EDIT CONTENT** ADD NEW CONTENT LIST VERSIONS X DELETE

Name Tag Template Example

Limit

Sizes size-0 size-1 size-2 size-3 size-4

Template

```
<div class="tag-cloud">
  <% for tag in @cloud %>
    <%= h(tag.name) %>
  <% end %>
</div>
```

SAVE

Abbildung 3.6: Tag-Portlet mit Template im Backend von Browser CMS

3.4.2 Erweiterungen

Die Funktionalität von Browser CMS kann durch das Hinzufügen von sogenannten Modulen erweitert werden. Diese beinhalten dabei neue Content-Blocks oder vordefinierte Portlets, die in der Content Library des Backends verwaltet werden können. Im Internet stehen zusätzlich folgende vorgefertigten Erweiterungen bzw. Module zur Verfügung¹⁴:

- browsercms-news¹⁵: Darstellung von kurzen Nachrichten im Frontend der Seite.

¹⁴Eine ausführliche Liste mit allen Erweiterungen findet sich unter folgender Adresse:
<http://modules.browsercms.org/modules>

¹⁵Komponenten-Download: https://github.com/browsermedia/bcms_news

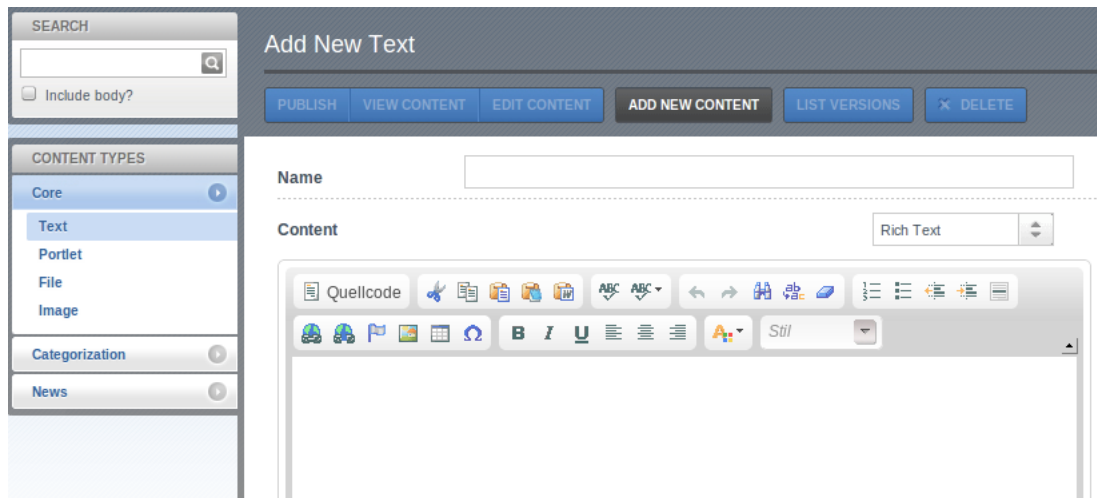


Abbildung 3.7: Erstellung eines neuen Datensatzes vom Inhaltstyp (Content Block) Text mit Hilfe des FCKEditor

- browsercms-blog¹⁶: Erstellung und Verwaltung von Blog-Einträgen inklusive Kommentar- und Tagging-Funktionalität.
- browsercms-events¹⁷: Erstellung und Verwaltung von Event-Einträgen. Diese können wahlweise alle zusammen in einer Listenansicht oder in Einzeldarstellung individuell präsentiert werden.
- browsercms-rankings¹⁸: Erlaubt dem Internetseitenbenutzer eine Bewertung der besuchten Seite abzugeben.

3.4.3 Verwendete Technologien

Das komplette Backend von Browser CMS wird mit Hilfe von in Rails gerenderten HTML-Views erzeugt. JavaScript kommt lediglich bei der Darstellung verschiedener Feldtypen (z.B. Datumsdialog und Auswahlbox) innerhalb der Content-Blöcke und beim Anlegen von Textinhalt durch die Verwendung des JavaScript-WYSIWYG-Editors FCKEditor (Abb. 3.7) zur Anwendung. Dieser ermöglicht die komfortable Formatierung eingestellter Texte sowie die Einbindung vorformatierter HTML-Inhalte.

¹⁶Komponenten-Download: https://github.com/browsermedia/bcms_blog

¹⁷Komponenten-Download: https://github.com/browsermedia/bcms_event

¹⁸Komponenten-Download: https://github.com/browsermedia/bcms_rankings

Die Erstellung der Content-Blöcke (Content Blocks) und Portlets wird durch einen in Rails umgesetzten Generatorskript ermöglicht und vereinfacht somit die Entwicklungsarbeit für den Programmierer (Abb. 3.1).

Quelltext 3.1: Aufruf und Ausgabe des Generator-Skripts zur Erstellung eines neuen Inhaltselements Produkt

```
1 rails g cms:content_block Product name:string price:integer description:html
2
3 create app/models/product.rb
4 create test/unit/models/product_test.rb
5 create app/controllers/cms/products_controller.rb
6 create app/views/cms/products/_form.html.erb
7 create app/views/cms/products/render.html.erb
8 create test/functional/cms/products_controller_test.rb
9 route namespace :cms do content_blocks :products end
10 create db/migrate/20111021055953_create_products.rb
```

3.5 Locomotive CMS

Locomotive CMS ist ein Open Source Content Management System der Ruby on Rails Entwickler Didier Lafforgue und Jacques Crocker sowie dem Designer Sacha Greif. Es wird unter der MIT-Lizenz der Open Source Initiative vertrieben und steht in den Rails-Versionen 2 und 3 als Download-Paket zur Verfügung. Neben den Möglichkeiten der eigenständigen Installation bieten die Initiatoren des Systems auch den kostenpflichtigen Dienst *Bushi.do* an, der eine automatische Komplettinstallation inklusive Hosting und Serverumgebung zur Verfügung stellt. Nach nur einem Klick kann sofort mit der Erstellung der Seite im Browser begonnen werden¹⁹.

3.5.1 Funktionsprinzipien

Locomotive CMS unterscheidet im Backend der Anwendung in folgende 2 Funktionsbereiche:

Inhalte

Im Inhalte-Bereich des Backends werden alle im System angelegten Seiten und Inhaltselemente verwaltet und bearbeitet (Abb. 3.8). Die Darstellung jeder einzelnen

¹⁹Auf der Projektseite von Locomotive CMS wird die automatische Installation mit Hilfe von Bushi.do beschrieben: <http://www.locomotivecms.com/>

Tabelle 3.3: Steckbrief Locomotive CMS

Aktuelle Version	keine Angabe von Entwicklungsversionen	
Lizenz	MIT License	
Projektseite	http://www.locomotivecms.com/	
Quellcode	https://github.com/locomotivecms/engine	
IRC-Channel	#locomotivecms	
API Dokumentation	http://rubydoc.info/github/resolve/refinerycms	
Forum	http://groups.google.com/group/refinery-cms/	
Demoversion	Frontend	http://diplom locomotive.herokuapp.com/
	Backend	http://diplom locomotive.herokuapp.com/admin
	Login	demo@demo.de
	Passwort	demo123
Verwendete Technologien	Ruby on Rails 3.0.x, HTML, jQuery und jQueryUI, diverse jQuery Plugins ,WYSIWYG-HTML-Editor Aloha und TinyMCE, MongoDB, Template Sprache Liquid	
Philosophie	Verwaltung kleiner Internetseiten Komplexe Inhaltselemente dank MongoDB selbst erstellen	
Zielgruppe	Privatnutzer, Kleinstunternehmen	

Seite und der Inhaltselemente kann darüber hinaus durch Angabe eines Templates online verändert werden (mehr dazu in Abschnitt 3.5.3), was eine Umsetzung komplexer Seitenlayouts sicherstellt.

Einstellungen

Im Einstellungs-Modul befinden sich die zentralen Funktionen zur Bearbeitung der vorhandenen Backend-Nutzer sowie globaler Locomotive CMS-Einstellungen (registrierte Domains, Import-/Export von Seiten, Bearbeitung von allgemeinen Metainformationen des Systems). Zusätzlich können in einem Template-Bereich einzelne, zentrale Dateien verwaltet werden, die innerhalb des Seitenlayouts als Gestaltungselemente dienen sollen²⁰.

²⁰Die für layoutspezifische Verwendung hochgeladenen Dateien werden in Locomotive CMS als Snippets bezeichnet

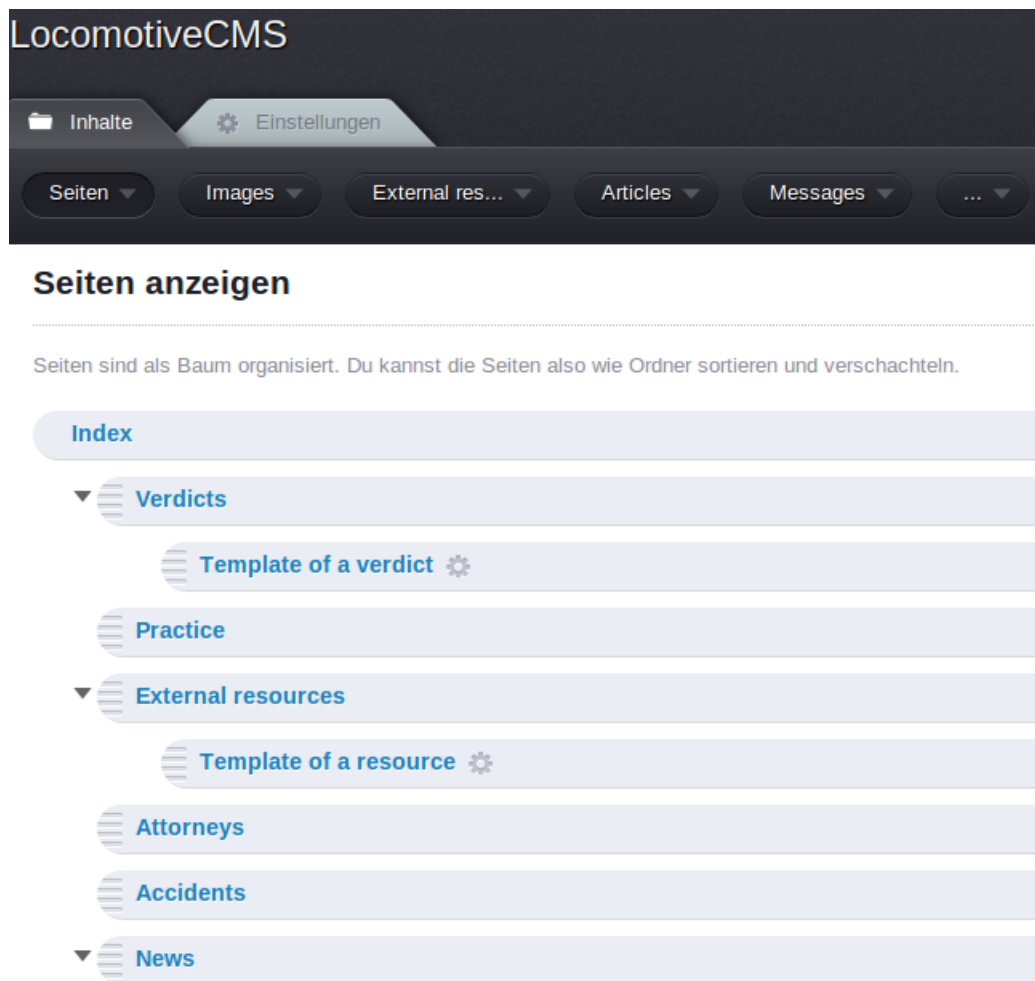


Abbildung 3.8: Backend von Locomotive CMS mit geöffnetem Seitenbaum (Ausschnitt)

3.5.2 Erweiterungen

Erweiterungen werden in Locomotive CMS als Bausteine bezeichnet und repräsentieren individuell zusammengestellte Inhaltselemente. Sie können im Backend von Locomotive CMS über einen komfortablen Bearbeitungsdialog erstellt werden (Abb. 3.9). Im Gegensatz zu den hier bereits vorgestellten Systemen werden so keine Programmierkenntnisse der Anwender benötigt. Ebenfalls entfällt der Aufwand zur Installation einer Erweiterung.

Neuer Baustein

Erstelle deine eigenen Baustein (Projekte, Leute, ...usw). Dein Baustein muss mindestens ein Feld haben. Das erste Feld ist dabei immer verpflichtend auszufüllen.

Allgemeine Informationen

Name*	Professoren
Slug*	professoren
Beschreibung	Eine Auflistung aller Professoren

Benutzerdefinierte Felder

Titel	Einfache Texteingabe	<input checked="" type="checkbox"/> Ist erforderlich	
Vorname	Einfache Texteingabe	<input checked="" type="checkbox"/> Ist erforderlich	
Nachname	Einfache Texteingabe	<input checked="" type="checkbox"/> Ist erforderlich	
Beschreibung	Text	<input checked="" type="checkbox"/> Ist erforderlich	

— ☐ Ist erforderlich

[← Ohne Speichern zurück](#)

Abbildung 3.9: Ein im Backend von Locomotive CMS zusammengestelltes Inhaltselement (Baustein)

Die Inhaltselemente können aus folgenden grundlegenden Feldtypen zusammengesetzt werden:

- Einfaches Textfeld
- Text
- Auswahlbox
- Checkbox
- Datum
- Datei

- has one (ermöglicht die Zuordnung genau eines anderen Inhaltselements des angegebenen Bausteintyps)
- has many (ermöglicht die Auswahl mehrerer anderer Inhaltselemente des angegebenen Bausteintyps)

3.5.3 Verwendete Technologien

Das Backend von Locomotive CMS besteht aus in Rails gerenderten HTML-Views mit eingebundenen JavaScript-Dateien, die einzelne Funktionalitäten bereitstellen. Das HTML wird dabei zusätzlich durch die Verwendung der JavaScript-Bibliothek Sizzle²¹ manipuliert, die es erlaubt durch Angabe eines CSS-Selektors bestimmte Elemente des HTML zu erfassen. Zur Erstellung von Templates wird die Ruby Template-Sprache Liquid eingesetzt. Sie hat ihren Ursprung in der kommerziellen Online E-Commerce-Plattform *shopify*²² und bietet Möglichkeiten der Vererbung und Überschreibung vorheriger definierter Templates. Der im Backend eingesetzte WYSIWYG-Editor TinyMCE ermöglicht die komfortable Eingabe von Text und HTML-Elementen²³. Im Frontend der Seite findet zusätzlich der WYSIWYG-JavaScript-HTML5-Editor Aloha²⁴ Verwendung. Er ermöglicht so eine Bearbeitung spezieller Inhaltselemente direkt im Frontend der Seite (Inline-Editing²⁵).

3.6 Refinery CMS

Refinery CMS – in der Kurzform oft als Refinery bezeichnet – ist ein freies Open Source Web Content Management System des neuseeländischen Entwicklerteams Resolve Digital, dessen Entwicklung 2004 durch David Jones eingeleitet wurde. Nach einer fünfjährigen, eingeschränkten Entwicklungsphase, in der nur wenige Bereiche des Systems verbessert wurden, erfolgte am 28. Mai 2009 die Veröffentlichung der ersten Open Source Software Version. In der Folgezeit wurde das CMS durch die Kernentwickler David

²¹Komponenten-Download: <http://sizzlejs.com/>

²²Liquid ist das Ergebnis einer Extrahierung dieser Funktionalität aus Shopify

²³Um den Editor innerhalb der einzelnen Seiten zu aktivieren, müssen im Liquid-Template der Seite entsprechende Befehle aufgerufen werden. Nach einem erneuten speichern der Seite steht der Editor im Backend zur Verfügung und kann mit Inhalten gefüllt werden.

²⁴Projektseite: <http://aloha-editor.org/>

²⁵Die Entwicklung dieses Features befindet sich noch im Betastadium und kann noch nicht zuverlässig verwendet werden.

Jones, Philip Arndt, Steven HeideI und Uģis Ozols auf das aktuelle Rails 3 umgestellt und eine erste stabile Version 1.0.0 veröffentlicht (28. Mai 2011).

Tabelle 3.4: Steckbrief Refinery CMS

Aktuelle Version	1.0.8	
Lizenz	MIT License	
Projektseite	http://refinerycms.com	
Quellcode	https://github.com/resolve/refinerycms	
IRC-Channel	#refinerycms	
API Dokumentation	http://rubydoc.info/github/resolve/refinerycms	
Forum	http://groups.google.com/group/refinery-cms/	
Demoversion	Frontend	http://demo.refinerycms.com
	Backend	http://demo.refinerycms.com/refinery
	Login	demo
	Passwort	demo
Verwendete Technologien	Ruby on Rails 3.0.x, HTML, jQuery und jQueryUI, WYSIWYG-HTML-Editor Wymeditor, HTML 5 Multi-Upload	
Philosophie	Realisierung einer benutzerfreundlichen, einfachen Oberfläche Einfaches Hinzufügen von Funktionalität an Hand der in Rails bekannten Entwicklungsabläufe Aktive Community durch Google Group und IRC, die eine schnelle Hilfe ermöglichen	
Zielgruppe	Privatnutzer, Kleinstunternehmen	

3.6.1 Funktionsprinzipien

Das Backend bildet in Refinery CMS die zentrale Anlaufstelle zur Erstellung und Verwaltung aller Inhalte, Einstellungen und Nutzer. Über ein zentrales Menü kann auf die Funktionsmodule des Systems zugegriffen werden, welche im folgenden vorgestellt werden:

Übersichts-Modul

Nach der Anmeldung am System bildet das Übersichts-Modul (o.a. Dashboard)

Abbildung 3.10: Backend-Ansicht von Refinery CMS

die Startseite des Systems. Dort können in einer einfachen Form die letzten Aktivitäten innerhalb des WCMS eingesehen werden. Zusätzlich werden Schnellaufrufe zu systemspezifischen Funktionen angeboten.

Seiten-Modul

Das Seiten-Modul listet alle angelegten Seiten und Unterseiten in Form einer Baumstruktur auf. Zusätzlich können Titel und Metainformationen bearbeitet werden. Darüber hinaus verfügt jede Seite über einen oder mehrere WYSIWYG-Editor-Textfelder²⁶, in die der Inhalt der Seite eingepflegt werden kann (Abb. 3.11).

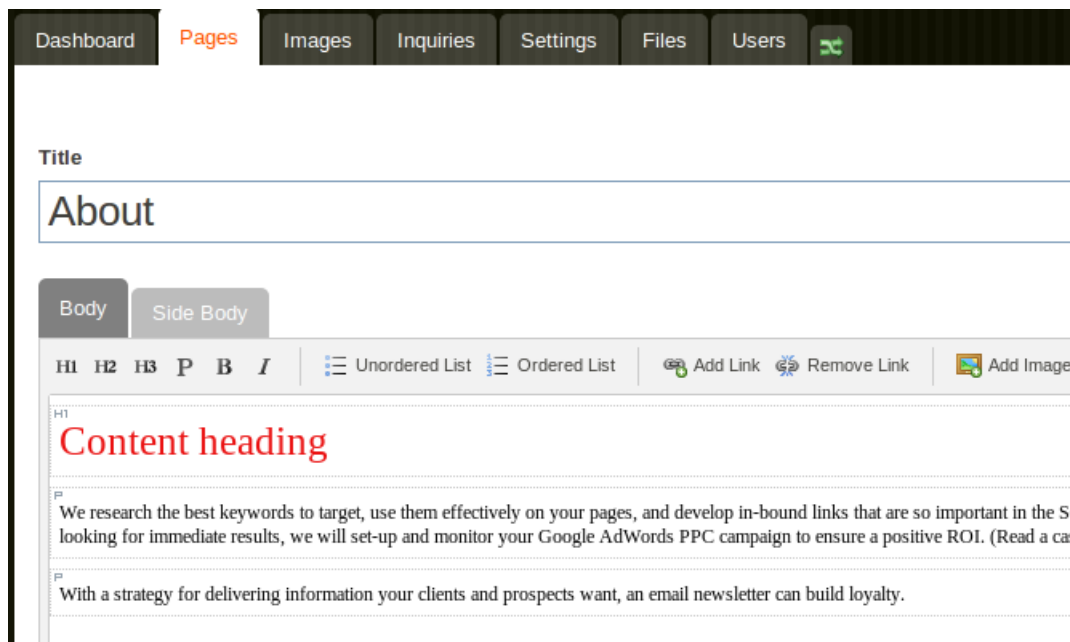


Abbildung 3.11: Seitenbearbeitung in Refinery CMS mit geöffneten Content Sections Body und Site Body

²⁶Die einzelnen Inhaltsblöcke werden als Content Sections bezeichnet. Die Anzahl der pro Seite zur Verfügung stehenden Inhaltsblöcke kann je nach Bedarf erhöht werden. Die Darstellung und Positionierung der einzelnen Blöcke wird durch ein zuvor festgelegtes HTML-Template im Rails-Quellcode festgelegt.

Bilder- und Dateien-Modul

In Refinery wird durch die Wahl der Menüpunkte Bilder und Dateien die Ressourcenverwaltung des Systems geöffnet. Dort können anschließend Mediendateien verschiedenster Formate hochgeladen, bearbeitet und durchsucht werden.

Benutzer-Modul

Das Benutzer-Modul erlaubt die Verwaltung der am System registrierten Anwender. U.a. können dort Nutzernamen, Passwörter und Berechtigungen für die Verwendung anderer Module gesetzt werden.

Einstellungs-Modul

Die einzelnen Module und Erweiterungen von Refinery können durch zuvor definierte Parameter²⁷ in ihrem Verhalten oder Aussehen beeinflusst werden. Das Einstellungs-Modul listet die in einer CMS-Installation vorhandenen Konfigurationsoptionen auf und ermöglicht eine nachträgliche Editierung und Löschung dieser.

3.6.2 Erweiterungen

Erweiterungen werden in Refinery als Engines bezeichnet. Sie werden durch Aktivierung innerhalb der Rails-Anwendung (im Quellcode) installiert und anschließend im Benutzer-Modul von Refinery den einzelnen Anwendern zugeordnet. Zum Zeitpunkt der Erstellung dieser Arbeit existieren u.a. folgende Erweiterungen:

- refinerycms-inquiries²⁸: Darstellung von Kontaktanfragen auf der Internetseite mit zusätzlicher Verwaltungsfunktion der Anfragen im Backend von Refinery CMS
- refinerycms-news²⁹: Verwaltung und Darstellung von Nachrichten im Front- und Backend-Bereich
- refinerycms-blog³⁰: Engine zur Erstellung kurzer Beiträge inklusive Kategorisierungs- und Kommentarfunktion

²⁷Die offizielle Bezeichnung der Parameter lautet *Refinery Settings*

²⁸Komponenten-Download: <https://github.com/resolve/refinerycms-inquiries>

²⁹Komponenten-Download: <https://github.com/resolve/refinerycms-news>

³⁰Komponenten-Download: <https://github.com/resolve/refinerycms-blog>

3.6.3 Verwendete Technologien

Refinery CMS ist ein technologisch sehr einfaches System. Die Realisierung der Backend-Funktionalitäten wird durch die konsequente Verwendung von HTML 5 erreicht. Zur Generierung der Dialoge, die u.a. innerhalb des Bild- und Dateien-Moduls eingesetzt werden, greift Refinery auf die jQuery UI Bibliothek zurück. Der im Seiten-Modul integrierte WYSIWYG-Editor ist ein für die Anforderungen des WCMS angepasster, XHTML konformer JavaScript WYMeditor. Durch die Unterstützung von HTML 5 innerhalb des Backend können Datenübertragungen von Medien an den Server in Form von Multi-Uploads realisiert werden. Die sonst übliche Nutzung von Flash entfällt und vereinfacht damit das System zusätzlich. Die bereits angesprochenen Erweiterungen (Engines) sind eigenständige Rails-Anwendungen, deren Grundgerüst mit Hilfe eines Refinery-Engine-Generators erzeugt wird. Die geringen technologischen Abhängigkeiten und Anforderungen des Systems erlauben es, Teile des Backends bei Bedarf anzupassen, zu verändern oder komplett auszutauschen.

3.7 Durchführung der funktionalen Analyse

Um die Leistungsfähigkeit der ausgewählten Systeme im Bereich des Webpublishing einschätzen zu können, werden die vorgestellten Kriterien des vorhergehenden Abschnitts mit den ausgewählten WCMS in Tabellenform gegenübergestellt. Es erfolgt dabei eine Festlegung der Erfüllung dieser Kriterien in folgende 3 Stufen:

Bestehende Einschränkungen und Problemen zu jedem WCMS werden in der Tabelle zusätzlich zusammenfassend aufgeführt. So wird eine genauere Einschätzung des tatsächlichen Funktionsumfangs der untersuchten Systeme möglich und eine bessere Verwertbarkeit für eventuelle Folgeuntersuchungen gewährleistet.

Tabelle 3.5: Mögliche Auswertungsstufen für die umgesetzten Funktionalitäten der WCMS

Erfüllt 100%	Das hier vorgestellte WCMS erfüllt die aus dem Kriterium definierten Funktionalitäten. Dies kann auch durch Installation einer Zusatzkomponente erreicht werden.
Erfüllt 50%	Das hier vorgestellte WCMS erfüllt die aus dem Kriterium definierten Funktionalitäten nur teilweise und unter Akzeptanz bestimmter Einschränkungen.
Erfüllt 0%	Das hier vorgestellte WCMS erfüllt die aus dem Kriterium definierten Funktionalitäten nicht. Es existieren darüber hinaus keine Erweiterungsmöglichkeiten für das System oder die Erfüllung ist nur durch eigenständige Implementierung (Programmierung) erreichbar.

3.7.1 Erstellung

In einem WCMS sollen mehrere Redakteure Inhalte gleichzeitig erstellen, ändern, löschen und verwalten können.			
Alchemy 1.6.0	Erfüllt: 100%	Refinery CMS 1.0.8	Erfüllt: 100%
Redakteure können Inhalte gleichzeitig erfassen und verwalten.		Redakteure können Inhalte gleichzeitig erfassen und verwalten.	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 100%
Redakteure können Inhalte gleichzeitig erfassen und verwalten.		Redakteure können Inhalte gleichzeitig erfassen und verwalten.	

In einem WCMS sollen Inhalte – unabhängig von Zeit und Standort – durch mehrere Benutzer online verwaltet und erfasst werden können.			
Alchemy 1.6.0	Erfüllt: 100%	Refinery CMS 1.0.8	Erfüllt: 100%
Vollständig unterstützt		Vollständig unterstützt	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 100%
Vollständig unterstützt		Vollständig unterstützt	

In einem WCMS soll eine Offline-Erfassung von Inhalten unter Verwendung eines lokal auf dem Rechner installierten Programms möglich sein.			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	

Das WCMS verfügt über eine integrierte Mediendatenbank zur Erfassung und Verwaltung von Bildern, Multimedia, Texten, Audio, Videos, usw. Die Inhalte werden dabei in einer Datenbank gespeichert.			
Alchemy 1.6.0	Erfüllt: 100%	Refinery CMS 1.0.8	Erfüllt: 100%
Alchemy bietet eine Bibliothek, in der Bilder und Dateien verwaltet werden können. Zur Speicherung verwendet Alchemy CMS das von Rails mitgelieferte Active Record als Datenbankpersistenzschicht. Durch die Verwendung von Migrationen können so eine Vielzahl relationaler Datenbanken unterstützt werden. Zusätzlich existieren einige Adapter, um auch dokumentenbasierte Datenbanken anzusteuern.		Refinery CMS bietet eine einfache Medienverwaltung. Wie Alchemy CMS greift Refinery ebenfalls auf Rails' Active Record zurück und unterstützt damit mehrere relationale und dokumentenorientierte Datenbanken.	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 100%
Browser CMS verfügt über eine <i>Content Library</i> , die eine einfache Medienverwaltung von Bildern, Dateien und definierten Inhaltselementen ermöglicht. Wie bei Alchemy und Refinery CMS wird hier auch auf Active Record zurückgegriffen. Die Entwickler garantieren auf Grund fehlender Tests jedoch nur die Unterstützung von SQLite und MySQL-Datenbanken. Tendenziell können aber alle von Active Record unterstützten Datenbanken eingesetzt werden.		Locomotive CMS bietet eine Asset-Verwaltung, in der selbst erstellte Inhaltselemente in Containern verwaltet werden können. Im Gegensatz zu den hier vertretenen Systemen basiert Locomotive CMS auf der dokumentenorientierten Datenbank MongoDB. Relationale Datenbanken werden somit nicht unterstützt. Eine Umsetzung von Locomotive CMS mit Active Record ist jedoch geplant.	

Inhalte sollen in einem WCMS ohne spezielle Programmier- und HTML-Kenntnisse erfasst und verwaltet werden können.			
Alchemy 1.6.0	Erfüllt: 100%	Refinery CMS 1.0.8	Erfüllt: 100%
Alle Inhalte können über den TinyMCE-JavaScript WYSIWYG Editor erfasst und formatiert werden.		Alle Inhalte können über den integrierten WYSIWYG-Editor Wymeditor erfasst und formatiert werden. Der Editor ist fest in das System integriert und kann nicht ausgetauscht werden. Ein Plugin, dass die Verwendung eines anderen Editors ermöglicht ist bereits in Planung.	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 100%
In Browser CMS findet der WYSIWIG-FCKEditor Verwendung. Zusätzlich stehen verschiedene Module zur Verfügung, die einen Austausch des Editors gegen andere Lösungen ermöglichen.		Alle Inhalte können über zwei integrierte WYSIWYG-Editoren erfasst und formatiert werden. Im Backend steht der JavaScript Editor TinyMCE zur Verfügung. Im Frontend findet der HTML5-WYSIWYG-Editor Aloha zur Manipulierung der Seiteninhalte Verwendung (befindet sich noch in der Entwicklung).	

Die Nutzung des WCMS erfolgt über einen Internet-Browser. Dabei können alle gängigen Internet-Browser (Internet Explorer, Safari und Firefox) eingesetzt werden.			
Alchemy 1.6.0	Erfüllt: 100%	Refinery CMS 1.0.8	Erfüllt: 50%
Vollständig unterstützt		Der Aufruf des Bildermodul-Dialogs konnte vom Internet Explorer nicht ausgeführt werden. Das zur Realisierung verwendete JavaScript bringt den Browser zum Absturz.	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 100%
Vollständig unterstützt		Vollständig unterstützt	

Ein WCMS soll Inhalte mehrsprachig erfassen und verwalten können.			
Alchemy 1.6.0	Erfüllt: 100%	Refinery CMS 1.0.8	Erfüllt: 100%
In Alchemy können Inhalte mehrsprachig angelegt werden. Durch die Auswahl einer bestimmten Sprache wird ein entsprechender Seitenbaum mit allen existierenden Inhalten zu der ausgewählten Sprache erzeugt.		Refinery CMS kann Inhalte mehrsprachig verwalten und ausgeben. Zur Aktivierung der Funktionalität müssen nur die zu unterstützenden Sprachen in einer Konfigurationsdatei angegeben werden (dies kann von Administratoren im Backend vorgenommen werden). Alle Sprachen werden dabei in einem einzigen Seitenbaum verwaltet. Vorhandene Übersetzungen zu einer bestimmten Seite werden durch Einblendung von Flaggensymbolen kenntlich gemacht.	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
In Browser CMS kann durch die Installation der Erweiterung <i>browsercmsi</i> die Unterstützung von mehrsprachigen Inhalten erreicht werden. Der Plugin-Anbieter konnte die 100% Rails 3-Kompatibilität der Erweiterung jedoch nicht garantieren. Von einem Einsatz dieser Lösung in einer Produktiv-Umgebung wird daher abgeraten. Innerhalb der Bewertung von Browser CMS werden daher 0% beim Erfüllungsgrad angegeben.		Inhalte können nur einsprachig verwaltet werden. Erweiterungen, die diese Funktionalität herstellen können, konnten nicht ermittelt werden.	

Inhalte können in einem WCMS während der Erfassung über eine Preview-Funktion vorab im Design der Webseite betrachtet werden.			
Alchemy 1.6.0	Erfüllt: 100%	Refinery CMS 1.0.8	Erfüllt: 0%
Redakteure können ihre erstellten und editierten Inhalte im Backend durch ein Preview-Fenster sichtbar machen. Änderungen an Inhaltselementen können somit sofort nachvollzogen werden.		Refinery CMS verfügt über keine Preview-Funktion der Inhalte. Ist ein Inhaltselement im Backend neu angelegt oder bearbeitet wurden, wird dies auf der Internetseite sofort sichtbar.	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 0%
Wie bei Alchemy werden Inhalte erst nach ihrer Veröffentlichung sichtbar. Bis dahin kann jedoch im Frontend durch Inline-Editing der Seite jedes Inhaltselement bearbeitet werden.		Locomotive CMS bietet wie RefineryCMS keine Preview-Funktion. Änderungen und neu angelegte Inhalte werden direkt veröffentlicht.	

Das WCMS ermöglicht eine Zuordnung von standardisierten und frei definierbaren Metadaten zu beliebigen Inhalten (z.B. Autor, Schlüsselwörter, benutzerdefinierte Felder).			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Metadaten zu Inhalten können nicht vergeben werden. Die Bibliothek von Alchemy unterstützt lediglich eine Auflistung von Ressourcen. Bilder und Dateien können damit nur in Form einer Listenansicht inspiziert werden. Eine Zuordnung zu Kategorien oder eine Anlage von Ordnerstrukturen zur Erleichterung der Orientierung ist nicht möglich. Die Verwaltung großer Datenmengen scheint daher nur schwer möglich.		Inhalte werden als einfache Datensätze betrachtet und besitzen daher keine definierten Metadaten. Ähnlich wie bei Alchemy gleicht die Ressourcenverwaltung nur einer einfachen Auflistung von Bildern und anderen Ressourcen. Eine Kategorisierung der Inhalte ist nicht möglich. Ebenfalls können keine Ordner zur sinnvollen Strukturierung der Ressourcen erstellt werden. Die Verwaltung großer Datenmengen wird dadurch schnell zu einem Geduldsakt.	
Browser CMS 3.3.1	Erfüllt: 50%	Locomotive CMS	Erfüllt: 50%
Metadaten können zu einzelnen Inhaltselementen in Form einer Tag-Liste hinzugefügt werden. Diese wird in der Datenbank als Text abgespeichert und bei ihrer Nutzung in einzelne Teil-Strings zerlegt. Das Hinzufügen zuvor definierter bzw. standardisierter Metadaten ist nicht möglich. Die <i>Content Library</i> von Browser CMS listet wie ihre Vorgänger lediglich die angelegten Bilder oder Dateien auf. Möglichkeiten zur sinnvollen Organisation (Kategorien, Ordner) großer Datenmengen sind nicht vorhanden.		Zu den verschiedenen Inhaltselementen können beliebig viele Metainformationen hinzugefügt werden. Auch die Darstellung von 1:1 und 1:n-Beziehungen ist möglich. Diese Funktionalität wird dabei vor allem durch die Verwendung der dokumentenbasierten Datenbank MongoDB möglich. Die Inhaltsverwaltung (Bausteine) kann nicht mit Hilfe von standardisierten Metadaten kategorisiert werden. Wie bei seinen Vorgängern sind die Datensätze lediglich in Listenform aufgeführt. Eine logische Strukturierung mit Hilfe von Ordnern ist nicht möglich.	

Das CMS soll über eine offene API (Programmierschnittstelle) für individuelle Erweiterungen verfügen.			
Alchemy 1.6.0	Erfüllt: 100%	Refinery CMS 1.0.8	Erfüllt: 100%
Eine flexible Plugin-DSL (Domain Specific Language) erlaubt das Hinzufügen von individuellen Erweiterungen.		Individuelle Inhaltselemente können durch die Verwendung der Refinery Engine Generatoren hinzugefügt werden. Die weitere Entwicklung Anpassung des Codegerüst folgt dann mit den innerhalb des Rails Framework üblichen Entwicklungstechniken und -abläufen.	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 100%
Ähnlich wie bei Refinery CMS können neue Module und Inhaltstypen mit Hilfe von speziellen Rails-Generatoren erzeugt werden.		Neue Inhaltstypen lassen sich im Backend durch ein einfaches User-Interface zusammenstellen. Mit wenigen Klicks sind so schnell neue Elemente erstellt. Programmierkenntnisse sind nicht notwendig.	

Das WCMS ermöglicht die Integration von Inhalten anderer Webseiten, Applikationen oder E-Commerce- Tools.			
Alchemy 1.6.0	Erfüllt: 100%	Refinery CMS 1.0.8	Erfüllt: 100%
Der verwendete WYSIWYG-Editor <i>TinyMCE</i> erlaubt in seiner HTML-Ansicht das Einbinden von Fremdinhalten anderer Seiten (z.B. IFrame). Zusätzlich ist die Erstellung von eigenen Inhaltselementen mit Hilfe der Alchemy Plugin DSL-API denkbar. So können auch die verschiedenen Ressourcen aus der Bibliothek von Alchemy Verwendung finden. Standardmäßig verfügt Alchemy bereits über die Inhaltselemente <i>Artikel</i> , <i>Text</i> , <i>Text mit Bild</i> , <i>Bilder</i> , <i>Bildergalerie</i> , <i>Überschrift</i> und <i>Intro</i> .		Refinery CMS verwaltet jede Internetseite innerhalb eines flexiblen WYSIWYG-Editors. Die Integration von vordefiniertem HTML-Code kann dabei durch Nutzung der HTML-Ansicht des Editors erreicht werden.	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 100%
Wie seine Vorgänger auch können innerhalb des WYSIWYG-Editors IFrames oder anderer HTML-Code eingebettet werden. Vordefinierte Inhaltselemente, die vorhandene Ressourcen aus der <i>Content Library</i> einbinden können, müssen eigenhändig angelegt werden.		Wie bei Alchemy und Refinery CMS können innerhalb des WYSIWYG-Editors HTML-Fragmente angegeben werden.	

In einem WCMS sollen Inhalte einfach importiert und exportiert werden können. Beim Austausch kommen Formate wie z.B. XML zum Einsatz.			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Alchemy verfügt über keine integrierten Import und Export-Funktionalitäten.		Es existieren auf Nutzerebene keine Möglichkeiten des Im- und Exports. Durch sogenannte Seed-Dateien ist jedoch ein nachträgliches Befüllen der Datenbank möglich. Der Aufruf erfordert jedoch Kenntnisse in Ruby on Rails und ist daher für Normalanwender/Redakteure nicht sinnvoll.	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 100%
In Browser CMS können Inhalte nicht importiert und exportiert werden. Entsprechende Features müssten erst eigenständig implementiert werden.		In Locomotive CMS kann ein kompletter Internetauftritt mit seinen Inhalten und Ressourcen importiert und exportiert werden. Zum Austausch der Inhalte findet eine <i>Zip</i> -Datei Verwendung, die alle benötigten Ressourcen (Bilder, Dateien, Templates usw.) sowie Inhalte der Datenbank einschließt. Ressourcen werden dabei in vordefinierten Ordnerstrukturen abgelegt. Die Datenbankeinträge aus MongoDB werden innerhalb der <i>Zip</i> -Datei im Unterordner <i>data</i> abgelegt. Die Einträge liegen dabei im <i>YAML</i> -Format vor.	

3.7.2 Kontrolle

Das WCMS verfügt über ein granulares Rechte- und Rollenkonzept für Anwender, Inhalte, Module (Plugins) und Webseiten.			
Alchemy 1.6.0	Erfüllt: 50%	Refinery CMS 1.0.8	Erfüllt: 0%
In Alchemy existieren vordefinierte Nutzerrollen (Registriert, Author, Redakteur, Administrator). Das Anlegen weiterer Rollen zur besseren Differenzierung ist jedoch nicht möglich. Die vordefinierten Rollen bestimmen den Funktionsumfang eines Anwenders im Backend. Angelegte Inhalte können einzelnen Nutzern nicht zugeordnet werden.		Refinery CMS besitzt kein Rechte- und Rollenkonzept. Dem Anwender kann lediglich der Zugang zu bestimmten Plugins erlaubt oder entzogen werden, um so den Funktionsumfang einzuschränken. Nutzer können alle Inhalte und Bereiche einer Webseite editieren, solange sie zur Nutzung des bestimmten Module (Engines) berechtigt wurden.	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 50%
In Browser CMS wird in einer Standardinstallation zwischen den Rollen Gast, CMS Administrator und Content Editor unterschieden. Zusätzlich können weitere Backend-Gruppen angelegt werden. Der Zugriff auf bestimmte Seiten (Seitenbaumzweige) kann auf bestimmte Nutzergruppen eingeschränkt werden. Zusätzlich bietet Browser CMS die Erstellung zusätzlicher Frontend-Nutzergruppen an, um so einen exklusiven Zugriff auf bestimmte Seiten des WCMS zur Verfügung zu stellen. Die Zugriffsberechtigung auf installierte Plugins kann ebenfalls für jeden Nutzer individuell festgelegt werden. Leider ist es nicht möglich, einzelne Inhaltselemente für bestimmte Nutzer unzugänglich zu machen.		Locomotive CMS besitzt ein einfaches Rechte- und Rollenkonzept. Es wird zwischen Administratoren, Designern und Autoren unterschieden. Das Anlegen weiterer Gruppen ist nicht möglich. Der Zugriff auf Seiten und Inhalte kann nicht individuell gesteuert und beeinflusst werden. Besitzt ein Anwender das Recht zum Editieren und Anlegen von Inhalten (Nutzergruppe Redakteur), können alle Inhalte im gesamten WCMS bearbeitet werden.	

Das WCMS ermöglicht die Versionierung von Inhalten. Zusätzlich können vorhergehende Zustände/Versionen mit Hilfe einer Wiederherstellungsfunktion rekonstruiert werden.			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Versionierung und Wiederherstellung von Inhalten wird nicht unterstützt.		Versionierung und Wiederherstellung von Inhalten wird nicht unterstützt.	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 0%
Wird unterstützt		Versionierung und Wiederherstellung von Inhalten wird nicht unterstützt.	

Das WCMS bietet einen Schutz vor gegenseitigem Überschreiben erfasst Inhalte durch z.B. Check in/ Check out- Mechanismen.			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
Wird nicht unterstützt. Die automatische Versionierung von Inhaltselementen erlaubt jedoch ein nachträgliches, manuelles Sichten und Zusammenfügen verschiedener Versionen.		Wird nicht unterstützt	

Das WCMS ist mandantenfähig, d.h. eine Mehrfachnutzung des Systems durch verschiedene Parteien mit kompletter Trennung der Daten und Benutzer ist möglich.			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
Wird nicht unterstützt		In Locomotive CMS können mehrere Internetauftritte gleichzeitig verwaltet werden. Eine Trennung der verschiedenen Nutzer und Daten wird jedoch nicht angeboten.	

Das WCMS bietet eine Linküberprüfung, die eine korrekte Darstellung von internen und externen Links auf der Internetseite sicherstellt.			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	

3.7.3 Freigabe

Das WCMS bildet einen mehrstufigen Workflowprozess für die Freischaltung von Inhalten ab.			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Alchemy bildet einen einfachen Workflowprozess bei der Freischaltung von Inhalt ab. dabei wird Redakteuren erlaubt, die von Autoren durchgeführten Änderungen zu kontrollieren und anschließend zu veröffentlichen. Ein Austausch zwischen beiden Nutzergruppen (wie im Modell des Content Life Cycles vorgesehen) ist nicht möglich. Redakteure müssen so Änderungen der Seiteninhalte selbst erkennen.		In Refinery CMS sind alle erstellten Inhalte sofort auf der Internetseite sichtbar. Lediglich die Sichtbarkeit der erstellten Internetseite kann beeinflusst werden. Die Implementierung des Freigabeprozesses innerhalb des Content Life Cycle wird somit nicht unterstützt.	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
Ein komplexer Workflowprozess kann in Browser CMS nicht abgebildet oder erstellt werden. Browser CMS verfügt wie Alchemy CMS über einen einfachen Workflowprozess. Autoren und Redakteure können jedoch dabei Nachrichten an die zuständigen Kontroll- bzw. Erstellungsinstanzen (Redakteure und Autoren) senden. Eine Kommunikation der Parteien ist somit gewährleistet.		Wie in Refinery CMS werden Inhalte nach ihrer Erstellung sofort sichtbar. Seiten können so bei ihrer Erstellung im Backend von Locomotive deaktiviert werden.	

Mit Hilfe des WCMS können <i>nicht technische</i> User den Workflowprozesse kreieren, verwalten und ändern. Es sollen dafür keine Programmierkenntnisse notwendig sein.			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	

Das WCMS bietet die Möglichkeit, externe Mitarbeiter in Workflowprozesse einzubinden.			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Diese Funktionalität wird nicht unterstützt. Ein externer Mitarbeiter muss erst als Backend-Nutzer im WCMS angelegt werden.		Diese Funktionalität wird nicht unterstützt.	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
Diese Funktionalität wird nicht unterstützt. Ein externer Mitarbeiter muss erst als Backend-Nutzer im WCMS angelegt werden. Es existieren jedoch bereits einige experimentelle Erweiterungen, die Nutzer aus externen Datenbanken in das WCMS integrieren (ohne vorherige Erstellung eines Nutzers im Backend von Browser CMS).		Diese Funktionalität wird nicht unterstützt.	

Unternehmensspezifische Bearbeitungsprozesse von Inhalten sollen über frei definierbare Workflows verwaltet werden können.			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	

3.7.4 Publikation

Das WCMS trennt Inhalt und Design durch die Verwendung von Templates.			
Alchemy 1.6.0	Erfüllt: 100%	Refinery CMS 1.0.8	Erfüllt: 100%
Inhalt und Design werden in Alchemy durch die Verwendung von <i>erb</i> -Templates getrennt. Das Haupt-Template der Seite wird zu Beginn der Entwicklung von einem Designer festgelegt und anschließend in der Anwendung verankert (als fixe Resource im Rails Quellcode). So können Redakteure das Aussehen der Internetseite nicht beeinflussen.		Wie Alchemy verwendet Refinery CMS <i>erb</i> als Template-Sprache. Trotz der so erreichten Trennung zwischen Inhalten und Design erschwert die fehlende Möglichkeit der Anpassung im Backend den Umgang mit dem gesamten WCMS. Dies gilt für das Haupttemplate der Seite sowie für alle Erweiterungen.	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 100%
Browser CMS unterstützt die Verwendung verschiedener Template-Sprachen. In einer Standard-Installation werden u.a. <i>erb</i> , <i>rjs</i> und <i>rxml</i> angeboten.		In Locomotive CMS kann für jede Seite ein Template angegeben werden. Die dabei verwendete Templatesprache ist <i>Liquid</i> .	

Das WCMS erlaubt die Mehrfachverwendung von Inhalten an verschiedenen Stellen (auf unterschiedlichen Seiten). Zusätzlich können angelegte Seiten kopiert und wiederverwendet werden.			
Alchemy 1.6.0	Erfüllt: 100%	Refinery CMS 1.0.8	Erfüllt: 0%
Inhaltselemente und Seiten können in Alchemy kopiert und wiederverwendet werden. Die Zuordnung eines neuen Templates muss durch den Administrator erfolgen (Änderung am Rails-Quellcode).		Inhalte und Seiten können nicht kopiert und mehrfach verwendet werden. Bei der Seitenerstellung ist es nur möglich, eine neu angelegte Seite auf eine bestehende Seite umzuleiten.	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
Es können nur Inhalte mehrfach auf verschiedenen Seiten verwendet werden. Seiten können nicht kopiert werden.		In Locomotive CMS kann nur für jede Seite ein neues Template angegeben werden. Inhalte sind den einzelnen Seiten zugeordnet und nur dort verwendbar (und verfügbar).	

Navigationsstrukturen werden vom WCMS automatisch generiert, publiziert und verwaltet.			
Alchemy 1.6.0	Erfüllt: 100%	Refinery CMS 1.0.8	Erfüllt: 100%
Wird unterstützt		Wird unterstützt	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 100%
Wird unterstützt		Wird unterstützt	

Die vom WCMS erstellten Seiten können barrierefrei umgesetzt werden.			
Alchemy 1.6.0	Erfüllt: 100%	Refinery CMS 1.0.8	Erfüllt: 100%
Barrierefreiheit kann bei entsprechender Umsetzung der Verwendeten Templates und CSS-Dateien erreicht werden.		Barrierefreiheit kann bei entsprechender Umsetzung der Verwendeten Templates und CSS-Dateien erreicht werden.	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 100%
Barrierefreiheit kann bei entsprechender Umsetzung der Verwendeten Templates und CSS-Dateien erreicht werden.		Barrierefreiheit kann bei entsprechender Umsetzung der Verwendeten Templates und CSS-Dateien erreicht werden.	

Inhalte sollen vom WCMS auf verschiedene Medien / Technologien (Cross Media Publishing, SMS / Mobile / WAP / usw.) ausgegeben werden können.			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	

Möglichkeit Inhalte für anderen Webseiten bereitzustellen (XML, Webservice)			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 100%
Wird nicht unterstützt		Es gibt Plugins mit optionaler Ausgabe als RSS-Feed oder XML. Eine Bereitstellung ausgewählter Inhalte ist damit möglich. Zusätzlich generiert Refinery eine XML-Sitemap, die von Suchmaschinen verwendet werden kann (Aufruf über die URL /sitemap.xml).	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 0%
Die Darstellung von Inhalten als RSS-Feeds ist möglich. Es muss jedoch selbst eingerichtet werden. Zusätzlich kann mit Hilfe des Plugins bcms_seo_sitemap eine XML-Sitemap der gesamten Seiten von Browser CMS erstellt werden.		Wird nicht unterstützt	

Das WCMS ermöglicht die Wahl zwischen dynamischer oder statischer Generierung der Seiten bzw. Inhalte (Caching).			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Diese Funktionalität wird nicht unterstützt. Alchemy CMS erzeugt nach dem ersten Aufruf einer Seite oder Inhaltes einen internen Cache, womit die Seite beim nächsten Zugriff schneller geladen werden kann. Der Nutzer kann dieses Cachingverhalten nicht beeinflussen. Er hat nur die Möglichkeit, den Cache für alle in Alchemy CMS verwalteten Seiten zu löschen.		Wird nicht unterstützt	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 100%
Browser CMS ermöglicht das Caching von Internetseiten. In den Seiteneinstellungen kann für jede Seite festgelegt werden, ob die Seite gecacht oder bei jedem Aufruf neu generiert werden soll.		In Alchemy CMS kann für jede Seite eine individuelle Cachedauer angegeben werden. Bei Bedarf kann das Caching für jede Seite individuell deaktiviert werden.	

Das WCMS unterstützt die automatische Erstellung einer Druckversion für jede einzelne Seite.			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	

3.7.5 Terminierung/Archivierung

Das WCMS erlaubt die freie Wahl des Publikationszeitraumes (zeitgesteuertes Auf- / Abschalten / Archivieren) von Inhalten.			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	

Im WCMS können Inhalte und Seiten archiviert werden.			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	
Browser CMS 3.3.1	Erfüllt: 50%	Locomotive CMS	Erfüllt: 0%
Browser CMS erlaubt die Archivierung von kompletten Seiten. Es ist jedoch nicht möglich, bestimmte Inhalte zu Archivieren. Die Seite bleibt nach der Archivierung jedoch in der Seitenverwaltung von Browser CMS verfügbar und wird nicht in einen Archivbereich verschoben.		Wird nicht unterstützt	

Das WCMS ermöglicht eine Durchsuchung der archivierten Inhalte und Seiten nach wählbaren Parametern (z.B. Monat oder Jahr).			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
Eine als archiviert gekennzeichnete Seite bleibt in der Seitenverwaltung (Sitemap) von Browser CMS verfügbar und wird nicht in einen gesonderten Archivbereich verschoben. Darüber hinaus existiert keine Möglichkeit, die archivierten Seiten nach bestimmten Parametern zu durchsuchen.		Wird nicht unterstützt	

3.8 Auswertung der Ergebnisse

Der Erfüllungsgrad der einzelnen Kriterien hat für die untersuchten WCMS folgendes Gesamtergebnis herausgestellt:

Bereich	Maximum	Web Content Management System			
		Alchemy CMS	Browser CMS	Locomotive CMS	Refinery CMS
Erstellung	1200	900	850	850	750
Kontrolle	500	50	200	50	0
Freigabe	400	0	0	0	0
Publikation	800	400	500	400	400
Terminierung/Archivierung	300	0	50	0	0
Gesamtpunkte	3200	1350	1600	1300	1150

Die untersuchten Systeme zeigen bei der Erfüllung der für Webpublishing relevanten Kriterien sehr unterschiedliche Resultate. Im Bereich der Inhaltserstellung verfügen die Systeme über solide Funktionalitäten. Die Webpublishingbereiche Freigabe, Kontrolle und Terminierung/Archivierung decken dagegen bei allen untersuchten Systemen funktionale Defizite auf. In der Gesamtbetrachtung liefert Browser CMS den größten Funktionsumfang. Vor allem durch die umfassendere Nutzer- und Rechteverwaltung kann sich das System von Alchemy CMS und Locomotive CMS absetzen, die funktional gesehen im Mittelfeld angesiedelt sind. Refinery CMS bietet auf Grund der nur minimal implementierten Rechteverwaltung die geringste Gesamtfunktionalität aller untersuchten Systeme.

4 Beschreibung ausgewählter Implementierungsdetails

In diesem Abschnitt der Arbeit werden die ausgewählten Systeme hinsichtlich ihrer technischen Umsetzung mit dem Rails-Framework untersucht. Bei der Betrachtung wird auf folgende Aspekte eingegangen:

1. Wartbarkeit der Erweiterungen
2. Nutzeroberfläche

4.1 Erweiterungen

Innerhalb des Rails Frameworks ist die Verwendung von Generatoren ein häufiges Mittel zur schnellen Entwicklung von funktionsfähigem Code (siehe Kapitel 2.1.6). Drei der vier hier vorgestellten Systeme¹ bedienen sich zur Realisierung neuer Inhaltslemente ebenfalls eines Generatorskripts, der ein für das jeweilige CMS funktionsbereites Grundgerüst erzeugt. Exemplarisch soll hier das Ergebnis eines Generatorkaufes in Refinery CMS aufgezeigt werden (Abb. 4.1). Das erstellte Inhaltslement Projekt verfügt dabei über die Felder Titel und Beschreibung:

Quelltext 4.1: Aufruf des Refinery Engine Generators mit Ausgabe des erzeugten Codegerüsts

```
1 rails generate refinery_engineproject name:string description:text
2 create vendor/engines/projects/app/controllers/admin/projects_controller.rb
3 create vendor/engines/projects/app/controllers/projects_controller.rb
4 create vendor/engines/projects/app/models/project.rb
5 create vendor/engines/projects/app/views/admin/projects/_actions.html.erb
6 create vendor/engines/projects/app/views/admin/projects/_form.html.erb
```

¹Locomotive CMS benötigt durch die Verwendung von MongoDB und der Möglichkeit, individuelle Inhaltslemente im Backend zu erstellen keine installierbaren Erweiterungsmodule

```
7 create vendor/engines/projects/app/views/admin/projects/_projects.html.erb
8 create vendor/engines/projects/app/views/admin/projects/_records.html.erb
9 create vendor/engines/projects/app/views/admin/projects/_project.html.erb
10 create vendor/engines/projects/app/views/admin/projects/_sortable_list.html.erb
11 create vendor/engines/projects/app/views/admin/projects/edit.html.erb
12 create vendor/engines/projects/app/views/admin/projects/index.html.erb
13 create vendor/engines/projects/app/views/admin/projects/new.html.erb
14 create vendor/engines/projects/app/views/projects/index.html.erb
15 create vendor/engines/projects/app/views/projects/show.html.erb
16 create vendor/engines/projects/config/locales/en.yml
17 create vendor/engines/projects/config/locales/fr.yml
18 create vendor/engines/projects/config/locales/lolcat.yml
19 create vendor/engines/projects/config/locales/nb.yml
20 create vendor/engines/projects/config/locales/nl.yml
21 create vendor/engines/projects/config/routes.rb
22 create vendor/engines/projects/db/migrate/create_projects.rb
23 create vendor/engines/projects/db/seeds/projects.rb
24 create vendor/engines/projects/features/manage_projects.feature
25 create vendor/engines/projects/features/step_definitions/project_steps.rb
26 create vendor/engines/projects/features/support/paths.rb
27 create vendor/engines/projects/lib/generators/refinerycms_projects_generator.rb
28 create vendor/engines/projects/lib/refinerycms-projects.rb
29 create vendor/engines/projects/lib/tasks/projects.rake
30 create vendor/engines/projects/readme.md
31 create vendor/engines/projects/refinerycms-projects.gemspec
32 create vendor/engines/projects/spec/models/project_spec.rb
```

Die durch den Aufruf erzeugten Dateien erfüllen dabei folgende wesentliche Funktionen:

Zeile 1 Aufruf des Generatorbefehls

Zeile 2-3 Der Generator erzeugt zwei Rails-Controller zur Steuerung der Logik im Frontend und Backend des Content Management Systems.

Zeile 5-15 Automatische Generierung von HTML-Views zur Darstellung aller im Frontend und Backend benötigten Komponenten. U.a. werden ein HTML-Formular zum Anlegen neuer Projekte im Backend (Zeile 6 *_form.html.erb*) und eine Auflistung von zur Verfügung stehenden Aktionen im Backend erstellt (Zeile 5 *_actions.html.erb*)

Zeile 4 und 22 Erstellung des in Rails benötigten Modells Project (Zeile 4 *project.rb*) und der zur Speicherung in der Datenbank benötigten Migration (Zeile 22 *create_projects.rb*)

Zeile 21 Erstellung einer Routing-Datei, welche die für das Frontend und Backend benötigten URL's bzw. Routen in der Rails-Anwendung registriert.

Zeile 16-20 Erstellung der für die Unterstützung von Mehrsprachigkeit notwendigen Konfigurationsdateien im YAML-

Zeile 24-32 Erzeugung der benötigten Dateien zur Unterstützung der Entwicklung von Tests sowie der Registrierung der Engine innerhalb der Rails-Anwendung (Zeile 28)

Bei Bedarf an weiteren Inhaltselementen ergibt sich schnell die Erkenntnis, dass die Realisierung der Inhaltselemente mit Hilfe eines Generators redundanten Rails-Code erzeugt. Damit verstößt diese Art der Erweiterungsentwicklung gegen den in Rails propagierten Ansatz des DRY (Don't repeat yourself).

Diese Aussage soll im folgenden näher spezifiziert werden:

1. Jedes Inhaltselement besitzt seine eigene Darstellungsrepräsentierung in Form von HTML-Views. Änderungen am Backend-Design des WCMS erfordern so eine Anpassung sämtlicher verwendeter Erweiterungen. Die Erweiterung und das WCMS werden so sehr stark von einander abhängig.
2. Werden Änderungen am Quellcode des WCMS vorgenommen (z.B. der Generatorskript erzeugt neuen HTML-Markup in den einzelnen Views), bedeutet dies für die Entwickler der Plugins ebenfalls eine Anpassung des Plugins².
3. Jedes Inhaltselement wird in einer eigenständigen Datenbanktabelle gespeichert. Ein Inhaltselement Projekt benötigt so z.B. die Datenbanktabelle Projekt mit den Tabellenfeldern Name und Beschreibung. Bei häufiger Verwendung zusätzlicher Inhaltselemente entsteht somit schnell eine beachtliche Anzahl an zusätzlichen Tabellen³.
4. Die für die Inhaltselemente benötigten Datenbankschemas müssen in Form von in Rails üblichen Migrationen verwaltet werden. Dies erfordert einen entsprechenden Mehraufwand bei der Pflege der Erweiterungen.

²Diese Aussage bezieht sich auf geringfügige Änderungen

³Die Zahl der Datenbanktabellen kann in Rails mit Hilfe von Single Table Inheritance (STI) minimiert werden.

5. Jedes Plugin bestimmt durch die Bereitstellung seiner eigenen HTML-Views die Integration und das Aussehen der Erweiterung im Backend des WCMS. Ein einheitliches Erscheinungsbild aller Plugins ist somit nur schwer umsetzbar.
6. Ein Import und Export von Inhaltselementen in das WCMS ist nur möglich, wenn die notwendigen Datenbanktabellen zuvor erzeugt wurden.
7. Inhaltselemente müssen für ihre Erreichbarkeit im Frontend und Backend des WCMS im Routing der Rails-Anwendung registriert werden. Bei der Nutzung vieler Erweiterungen entstehen so zahlreiche zusätzliche Routingeinträge.
8. Durch die Installation/Verwendung der umfassenden Erweiterungen wird die Gesamtgröße der Rails-Anwendung unnötig vergrößert (geringfügige Bedeutung).
9. Durch das Laden zusätzlicher Rails-Controller und Models benötigt die Rails-Anwendung zusätzlichen Arbeitsspeicher und weist einen verlängerten Boot-Prozess auf (geringfügige Bedeutung).
10. Durch das Laden zusätzlicher Rails-Controller und Models benötigt die Rails-Anwendung zusätzlichen Arbeitsspeicher und weist einen verlängerten Boot-Prozess auf (geringfügige Bedeutung).

Der Verstoss gegen das Dry-Prinzip wird vor allem bei der Umsetzung der einzelnen Erweiterungs-Controller innerhalb von Refinery CMS ersichtlich. Dort wird mit Hilfe der im Controller verfügbaren Klassenmethode *crudify* das gesamte Grundgerüst des Controllers dynamisch erzeugt⁴. Die Methode kann dabei verschiedene Parameter aufnehmen, an Hand deren die Ausgabe gesteuert werden kann. Alle neu erzeugten Inhaltselemente sind in ihrer Grundstruktur als Rest-basierte Ressourcen anzusehen, die mit Hilfe eines angepassten Rails-Generators in das jeweilige Backend des WCMS eingebunden werden. Durch diese gegenseitigen Abhängigkeiten ist die Wartbarkeit der Erweiterungen sehr von der Entwicklung des Kernsystems abhängig. Vorallem Änderungen am Generatorskript des jeweiligen WCMS ziehen häufig Anpassungen bei den erzeugten Plugins nach sich. Durch die Reduzierung der Abhängigkeiten zwischen WCMS und

⁴Die Methode *crudify* erstellt mit Hilfe von Ruby-Metaprogrammierung die gesamte Logik des Controllers. Alle Aktionen, die der Controller verwaltet, werden so erst zur Laufzeit der Rails-Anwendung generiert. Nähere Informationen zu *Crudify* im Anhang der Arbeit unter Abschnitt 7.3.

Erweiterung (vor allem auf Darstellungsebene durch die verwendeten Views) kann eine stabilere Entwicklungsbasis geliefert werden.

Quelltext 4.2: Projects-Controller mit verwendeter crudify-Methode und optionalen Parametern

```
1 module Admin
2   class ProjectsController < Admin::BaseController
3
4     crudify :project ,
5             :title_attribute => 'name', :xhr_paging => true
6
7   end
8 end
```

4.2 Nutzeroberfläche

Die Nutzeroberflächen der vorgestellten Web Content Management Systemen sind durch die Kombination individueller HTML, CSS und JavaScript-Dateien zusammengestellt wurden. Bei der Nutzung der Systeme und der Durchführung von Erweiterungsentwicklungen ergeben sich somit folgende Eindrücke:

1. Das Backend der Systeme wird in Rails mit Hilfe verschiedener HTML-Views und Partialen (wiederverwendbare HTML-Views) erzeugt. Dies resultiert in einer verstärkten Antwortzeit⁵, da das Rails-Framework die gesamte Backend-Oberfläche erst neu erzeugen muss. Vor allem bei der Nutzung von Browser CMS und Refinery CMS macht sich die Art der Backend-Generierung bemerkbar.
2. Die Nutzeroberfläche der Medienverwaltung (Bilder und Dateien) erlaubt nur eine Auflistung aller im System vorhandenen Ressourcen (Abb. 4.2). Bei größeren Datenmengen kommt diese Art der Darstellung schnell an ihre Leistungsgrenzen. Diese Beschränkung der Nutzeroberfläche besteht bei allen untersuchten WCMS.
3. Das verwendete JavaScript zur Realisierung von Dialogen (z.B. Auswahl eines Bildes) und Backend-Funktionalitäten (z.B. Einbindung eines WYSIWYG-Editors) wird in Form einfacher JavaScript-Funktionen in das von Rails erzeugte Template

⁵Die Antwortzeit (response time) ist die Zeitdauer, die eine Anwendung bzw. ein System benötigt, um eine Anfrage von außen zu verarbeiten (z.B. UI-Aktion) [Fow03, S. 21].

eingebunden (Abb. /refrefineryimagedialog). Die Skripte sind dabei nicht modular aufgebaut und machen so eine Integration individueller Funktionalitäten nur schwer möglich.

Quelltext 4.3: Beispiel für eine JavaScript-Funktion zur Realisierung des Bildauswahldialogs in Refinery CMS. Das Codebeispiel zeigt ebenfalls die Abhängigkeit zu dem verwendeten HTML-Markup (z.B. Zeile 18)

```
1
2 var image_dialog = {
3   initialised: false
4   , callback: null
5   , init: function(callback){
6     if (!this.initialised) {
7       this.callback = callback;
8       this.init_tabs();
9       this.init_select();
10      this.init_actions();
11      this.initialised = true;
12    }
13    return this;
14  }
15  //...
16  , set_image: function(img){
17    if ($(img).length > 0) {
18      $('#existing_image_area_content ul li.selected').removeClass('selected');
19      $(img).parent().addClass('selected');
20      var imageId = $(img).attr('data-id');
21      var geometry = $('#existing_image_size_area li.selected a').attr('data-geometry')
22      ;
23      var size = $('#existing_image_size_area li.selected a').attr('data-size');
24      var resize = $('#wants_to_resize_image').is(':checked');
25      image_url = resize ? $(img).attr('data-' + size) : $(img).attr('data-original');
26      //...
27    }
28  }
29  //...
30 };
```

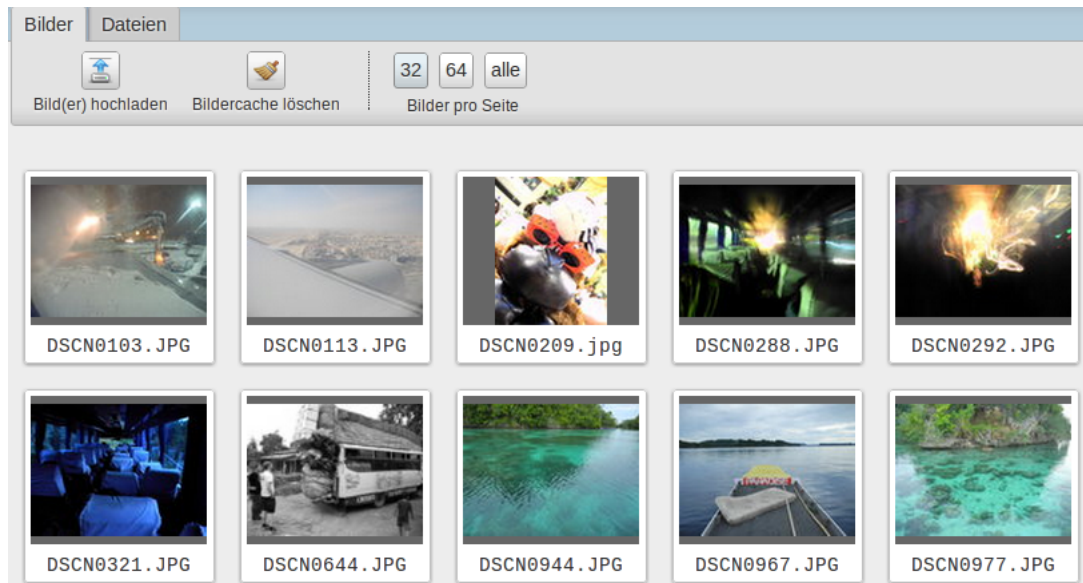


Abbildung 4.1: Ressourcen-Auflistung ohne Möglichkeiten der Strukturierung in Alchemy CMS. Für die anderen Systeme ergibt sich ein ähnliches Gesamtbild.

5 Lösungsvorschläge

In diesem Kapitel werden Lösungsvorschläge zur Beseitigung der in Kapitel 4 beschriebenen Probleme formuliert.

5.1 Implementierung eines Ruby on Rails Content Repository

Die in den analysierten WCMS umgesetzten Implementierungen der Erweiterungsentwicklung verstoßen zum Teil gegen das in Rails gebotene Prinzip des DRY (vgl. Kapitel 4.1). Eine konzeptionelle Änderung innerhalb dieses Systembereiches hätte somit Auswirkungen auf die gesamte WCMS-Infrastruktur der Inhaltsspeicherung. Dennoch soll hier der mögliche Lösungsansatz eines Content Repository konzeptionell vorgestellt werden.

5.1.1 Idee und Konzept

Heutige Webanwendungen (u.a. auch Web Content Management Systeme) benötigen neben der klassischen Speicherung von Daten zahlreiche zusätzliche Daten-Management-Funktionalitäten. Ein Content Repository (CR) soll dieser Entwicklung Rechnung tragen und definiert daher ein abstraktes Datenmodell zur Datenspeicherung und zusätzliche Servicefunktionalitäten, die häufig von content-orientierten Anwendungen verwendet werden. Ein Content Repository stellt somit u.a. folgendes Leistungsspektrum bereit:

- Speicherung strukturierter und unstrukturierter Daten (z.B. Binär- und Textformate oder Metadaten),
- Möglichkeiten der Zugangskontrolle
- Möglichkeiten der Sperrung (Locking)

- Durchführung von Transaktionen
- Versionierungsmechanismen
- Überwachung von Daten
- Volltextsuche

Der Zugriff auf das Content Repository wird dabei durch eine API genau definiert und garantiert somit einen definierten Zugriff auf die zusätzlichen Funktionalitäten. Andere Entwickler können auf das Content Repository und seine Zusatzfunktionen kontrolliert zugreifen, ohne etwas über die Infrastruktur und seine Implementierungsdetails zu wissen. Intern greift das Content Repository auf andere Technologien und Infrastrukturen zurück, um die geforderten Funktionalitäten abzubilden¹.

Im Bereich der Java Enterprise Content Management Systeme werden Content Repositories bereits erfolgreich eingesetzt. Namhafte Beispiele sind dabei das kommerzielle ECMS CQ5 und CRX² von Adobe und das u.a. als Open Source Software verfügbare Alfresco ECMS³. Innerhalb der PHP-Entwicklergemeinschaft wird ebenfalls gerade an der Umsetzung eines auf PHP basierten Content Repositories gearbeitet. Initiatoren sind dabei vor allem die Typo3 Association mit einer Implementierung innerhalb des neuen Web Content Management Systems Typo3 5.0⁴.

5.1.2 Das Java Content Repository (JCR)

Unter der Leitung von David Nüscheler der Firma Day Software wurde 2005 die erste Version einer Implementierung und API Spezifikation eines Java Content Repository veröffentlicht⁵. Die umgesetzte Referenzimplementierung wurde später unter der Lei-

¹Das im folgenden beschriebene Java Content Repository Jackrabbit verwendet in seiner Standardkonfiguration z.B. WebDAV zur Speicherung von Daten sowie Apache Lucene zur Realisierung der Suchindexierung.

²Nach der Übernahme von Day Inc. durch Adobe im Sommer 2010 wird das angebotene Content Management System CRX als Adobe Projekt weitervertrieben.

³Informationen zu Alfresco und dem Content Repository: <http://www.alfresco.com/products/platform/>

⁴Das TYPO3CR ist als eigenständiges Paket im PHP-FLOW3-Framework verfügbar und ein zentraler Bestandteil von Typo3 5.0

⁵Die Entwicklung ging im Java Specification Request (JSR) 170 als offizieller Standard in die Programmiersprache Java ein. Momentan wird an der Veröffentlichung der Version 2.0 des Standards gearbeitet(JSR-283).

tung der Apache Foundation als Open Source Projekt Jackrabbit⁶ der Öffentlichkeit zur Verfügung gestellt.

Zur Speicherung der Inhalte definiert das Java Content Repository ein einfaches hierarchisches Datenmodell, das folgende Objektstruktur aufweist:

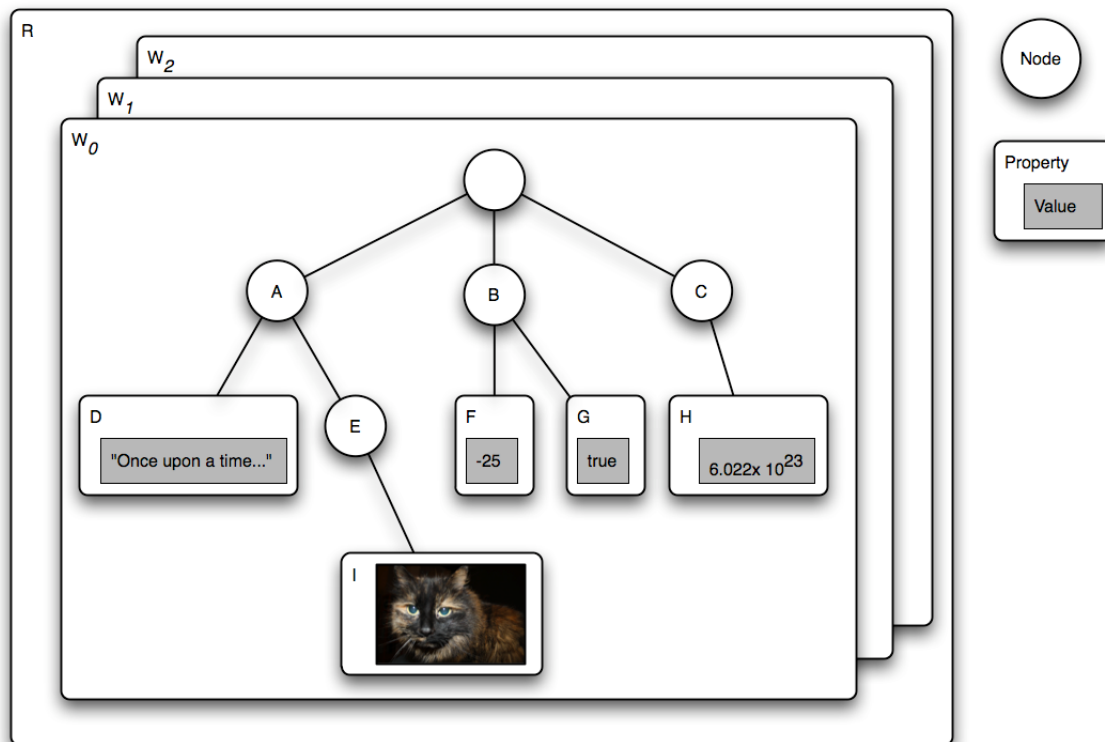


Abbildung 5.1: JCR-Datenmodell

Workspace

Ein JCR-Repository besteht aus einem oder mehreren Workspaces (Arbeitsbereichen), die jeweils mehrere Items in einer Baumstruktur beliebiger Tiefe verwalten können. Jeder Workspace lässt sich durch einen Namen eindeutig identifizieren (W_0 , W_1 , W_2) und enthält mindestens ein Item (root node).

Item

Ein Item kann entweder ein Node (Knoten) oder ein Property (Eigenschaft) sein.

⁶Projektseite: <http://jackrabbit.apache.org/>

Node

Knoten (Nodes) eines Workspaces bilden die Struktur der zu speichernden Daten ab. Ein Knoten kann daher keine oder weitere andere Kinderknoten (child notes) enthalten.

Property

Eine Property kann keine anderen Items beinhalten, aber dafür den Inhalt in Form von sogenannten Values abspeichern. Dabei kann ein Property-Node keine oder mehrere Values beinhalten.

5.1.3 Umsetzungsvarianten innerhalb von Ruby on Rails

Durch die Verfügbarkeit eines Ruby-Interpreter in der Programmiersprache Java (jruby⁷) ergeben sich für die Umsetzung in Ruby on Rails folgende 2 Möglichkeiten:

- Verwendung der Open Source Java Implementierung Jackrabbit und Spezifikation einer API zur Nutzung dieser innerhalb von Rails. Erste Implementierungsversuche und Demonstrationsanwendungen wurden bereits erstellt⁸.
- Erstellung einer eigenständigen, komplett auf Ruby und dem Rails Framework basierenden Referenzimplementierung und API nach dem Vorbild des Java Content Repository

5.1.4 Vorteile für die gewählten Ruby on Rails WCMS

Die Umsetzung eines Content Repository in Ruby kann für die bestehenden Web Content Management Systeme folgende Vorzüge bringen:

- Beseitigung des DRY-Verstosses bei den untersuchten Rails-WCMS, da die Inhalte in dem durch das Content Repository zur Verfügung gestellten hierarchischen Datenmodell gespeichert werden können.

⁷Projektseite: <http://jruby.org/>

⁸Eine mit Rails 2 umgesetzte Demo-Anwendung ist unter folgender Internetseite verfügbar: <https://github.com/wpc/jcr-rails-demo>. Sie zeigt dabei lediglich das Grundgerüst zur Umsetzung eines Content Repository auf.

- Wegfall der bisher notwendigen Datenbankmigrationen und zusätzlichen Datenbanktabellen, da das Content Repository die gesamte Infrastruktur bereitstellt und die Speicherung der Daten übernimmt.
- Vereinfachung des Zugriffs auf Inhalte innerhalb verschiedener Systeme durch Verwendung einer definierten API
- Fehlende Webpublishing-Funktionalitäten der existierenden Rails WCMS können durch eine Implementierung eines Content Repository als Infrastruktur allgemein zur Verfügung gestellt werden (z.B. Versionierung, Suchfunktionen).
- Durch die im JCR-Standard festgelegten Import- und Exportfunktionen kann ein Austausch der Inhalte zwischen verschiedenen Web Content Management Systemen ermöglicht werden.
- Ein Content Repository kann als Erweiterung auch von anderen Rails-Anwendungen verwendet werden, die mit verschiedenartig strukturierten Inhalten umgehen müssen.

5.2 Übertragung des Typo3 5.0 Phoenix User-Interfaces in Rails 3.1

Content Management Systeme erfordern bei steigender Funktionalität ein entsprechend komplexeres Nutzer-Interface. Die sinnvolle Realisierung entsprechender Oberflächen ist mit der u.a. in Refinery CMS gewählten Generierung von HTML-Views und einzelnen JavaScript-Dateien nur noch schwer möglich. Zur Umsetzung solcher Projekte empfiehlt sich daher der Einsatz alternativer Technologien. Die neue Version 5.0 des PHP basierten Web Content Management Systems Typo3 greift bei der Generierung der gesamten Backend-Oberfläche auf das Java Script Frameworks Ext JS 4⁹ zurück. Es ermöglicht durch Kombination vorgefertigter Elemente eine schnelle Erstellung komplexer Oberflächen¹⁰. Die Veröffentlichung des Typo3-Projektes unter der GPLv3-Lizenz macht eine Weiternutzung der in der Version 5.0 geplanten Oberfläche generell möglich. Im folgen-

⁹Informationen und Download: <http://www.sencha.com/>

¹⁰Beispielanwendungen: <http://dev.sencha.com/deploy/ext-4.0.0/examples/>

den sollen daher die notwendigen Schritte zur Integrierung des Typo3 5.0 User-Interfaces in eine Rails 3.1 Anwendung beschrieben werden.

5.2.1 Typo3 5.0

Die Entwicklung von Typo3 5.0 befindet sich noch in einer frühen Phase. Interessenten können jedoch bereits Entwicklerversionen in Form sogenannter Sprint Releases herunterladen und testen. Neben einem Download-Paket¹¹ auf der Projektseite von Typo3 wird zusätzlich eine aktuelle Version von Typo3 als Live-Demo¹² angeboten. Die Zugangsdaten zum Backend können im Frontend der Seite mit Hilfe eines Formulars erzeugt werden.

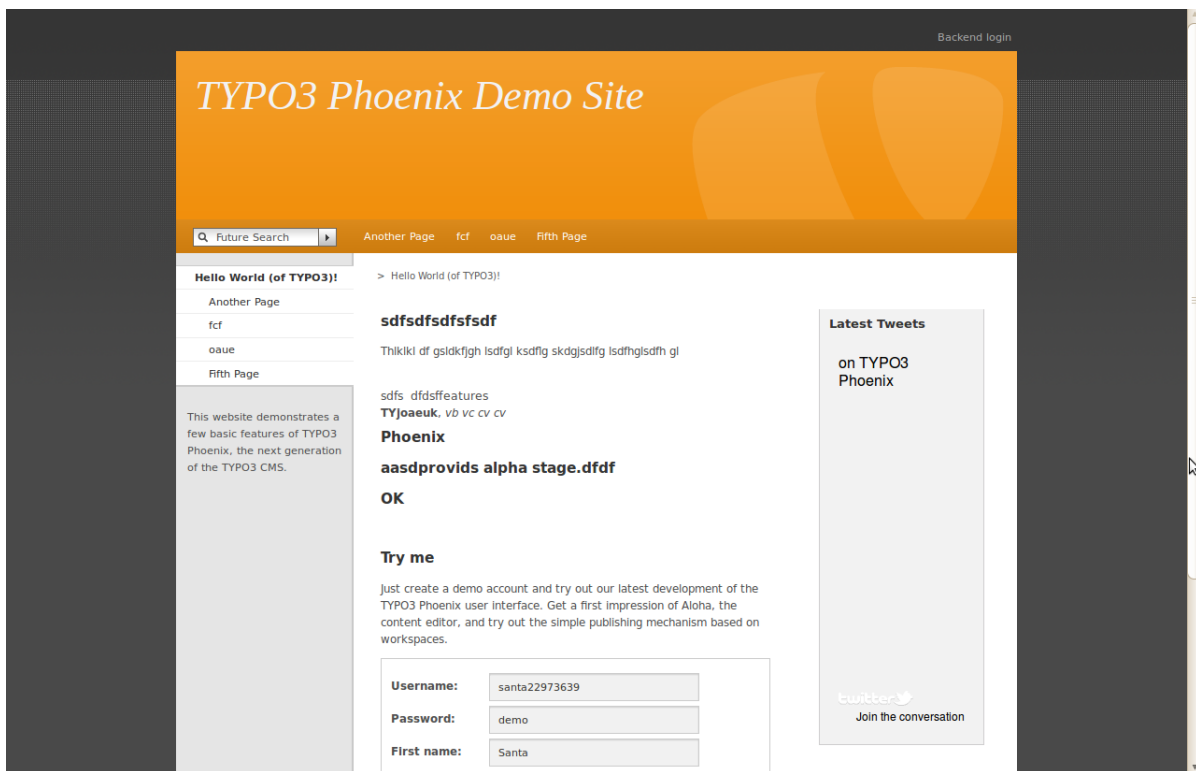


Abbildung 5.2: Frontend-Ansicht der Typo3 5.0 Sprint Release 6 Demoversion

¹¹Komponenten-Download: <http://flow3.typo3.org/typo3-phoenix/>

¹²Demoseite: <http://phoenix.demo.typo3.org/>

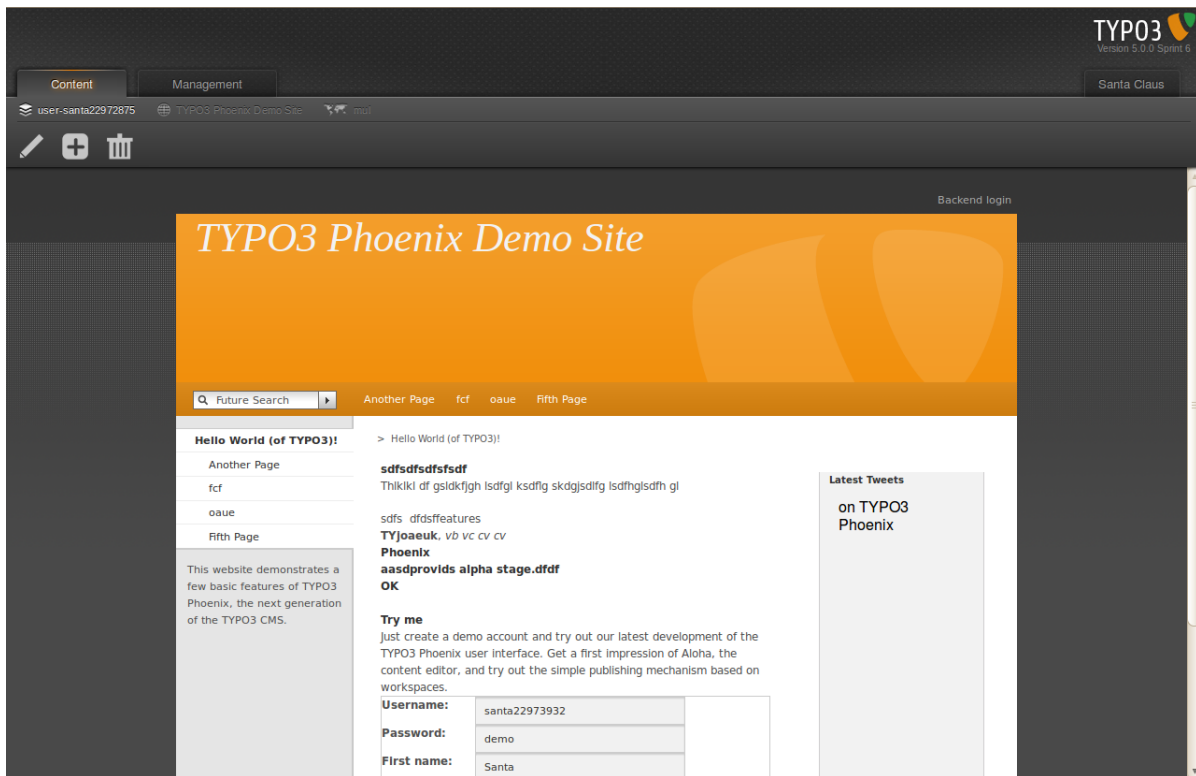


Abbildung 5.3: Backend-Ansicht der Typo3 5.0 Sprint Release 6 Demoversion

Das in der Demoversion umgesetzte Nutzer-Interface repräsentiert nur einen Teil der für Typo3 5.0 geplanten Oberfläche und Komponenten¹³. U.a. sind folgende Bestandteile bereits umgesetzt wurden:

- Login-Seite zur Anmeldung im Backend von Typo3 5.0
- Content-Modul im Backend mit integrierter Vorschau der aktuell ausgewählten Seite und beschränkten Möglichkeiten der Inhaltsbearbeitung mit Hilfe des für Typo3 5.0 geplanten Aloha-Editors
- Management-Modul zur Verwaltung der im System angelegten Seiten in Form einer Baumstruktur
- Dashboard des angemeldeten Nutzers mit Auflistung der editierten Inhalte

¹³Bilder und ausführliche Erläuterungen zur neuen Typo3 5.0 Oberfläche sind unter folgender Adresse zu finden: <http://typo3.org/teams/usability/t35ui/>

5.2.2 Ext JS und Ext Direct

Das Typo3-User-Interface setzt bei der Kommunikation zwischen den einzelnen Interface-Komponenten und dem Server auf die Nutzung von Ext Direct. Ext Direct beschreibt dabei einen in Ext JS definierten Sprachstandard, mit dessen Hilfe serverseitige Funktionen clientseitig per JavaScript aufgerufen werden können. Für ein besseres Verständnis wird in Abbildung 5.1 eine im jQuery Framework formulierte Ajax-Anfrage und der entsprechende Ext Direct Ausdruck gegenübergestellt.

Quelltext 5.1: Ajax-Anfrage an einen Server im jQuery-Framework

```
1
2 // jQuery Ajax Request
3 $.ajax({
4     url: ' /ajax?ajaxID=MyController_myMethod&parameter1=someValue ',
5     success: function( data ) {
6         alert( data );
7     }
8 }
9 });
10
11 // Ext Direct Aufruf
12 Namespace.MyController.myMethod( "some_Value", function( data , e) {
13     alert( data );
14 } );
```

Beide JavaScript-Beispiele resultieren in einer asynchronen Ajax-Anfrage, die die entsprechende Methode serverseitig aufruft. Für Ext Direct ergeben sich jedoch folgende zusätzliche Vorteile:

1. Keine wiederholte Angabe einer URL zum Aufruf der serverseitigen Funktionen.
2. Vereinfachung des JavaScript-Quellcodes, da im Vergleich zu herkömmlichen asynchronen Anfragen weniger Quellcode geschrieben werden muss.
3. Vereinfachung der clientseitigen JavaScript-Programmierung, da der Aufruf von client- und serverseitigen Methoden namentlich übereinstimmt.

Eine umfassende Beschreibung von Ext Direct ist dem Anhang der Arbeit beigelegt (Anhang 7.4). Dort werden alle Konzepte und notwendigen Komponenten für eine serverseitige Unterstützung von Ext Direct erläutert.

Um Ext Direct auch innerhalb einer Rails-Anwendung nutzbar zu machen, wurde im Rahmen dieser Diplomarbeit die Rails 3.1 kompatible Erweiterung *extr* entwickelt¹⁴. Der für Ext Direct notwendige Router (Anhang 7.4) wurde dabei mit Hilfe einer Rack Middleware realisiert, die alle Ext JS-Anfragen an die entsprechenden Serverseitigen Methoden (Controller-Methoden) weiterleitet. Schematisch ergibt sich somit folgender Anfrageablauf innerhalb der Rails-Anwendung:

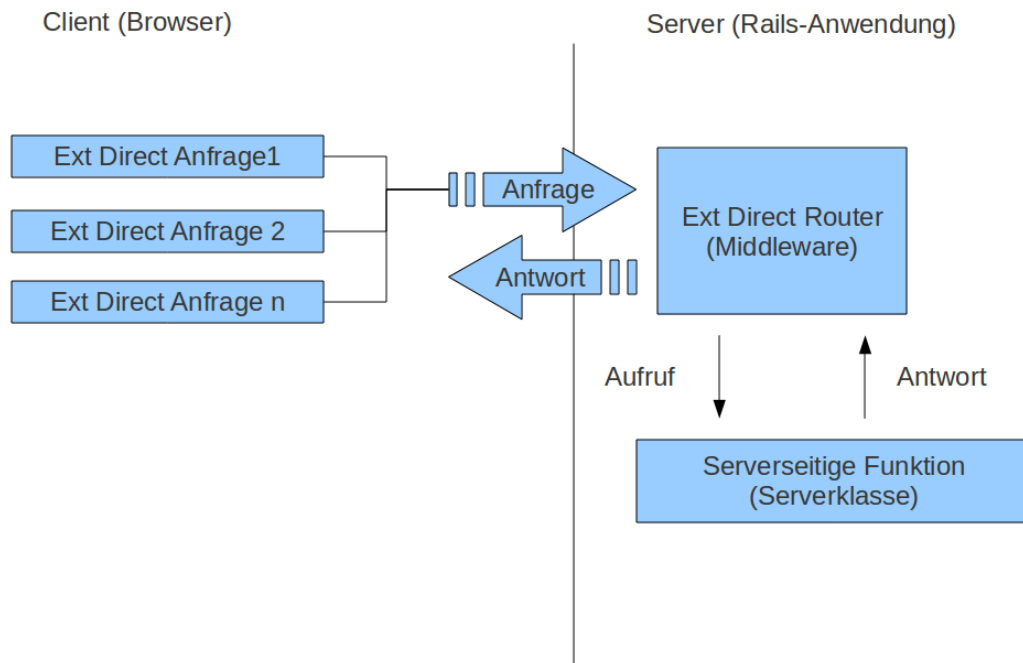


Abbildung 5.4: Schema des Ext Direct Routing mit *extr*

Die Installation und Nutzung der Erweiterung *extr* wird im Anhang dieser Arbeit demonstriert. Auf Grund des frühen Entwicklungsstatus unterstützt die Erweiterung serverseitig noch nicht alle der geforderten Ext-Direct-Funktionalitäten. Im folgenden sollen bestehende Beschränkungen und Probleme erläutert werden:

- Ext Direct-Anfragen mit Parametern aus Formularen und Datei-Uploads (Form Posts) werden von der Rails Middleware noch nicht vollständig an die entsprechende Zielmethode (controller action) weitergeleitet.

¹⁴Für Ruby und das Rails-Framework existieren bereits Implementierungen, die jedoch nicht vollständig zu Rails 3 und 3.1 kompatibel sind. Aus diesem Grund wurde die Umsetzung einer eigenen Erweiterung ins Auge gefasst.

- kein Schutz vor CSRF/XSRF-Angriffen¹⁵

5.2.3 Umgesetzte Rails-Anwendung

Durch die Realisierung der Erweiterung *extr* kann eine Übertragung des Typo3-UI in Rails 3.1 möglich gemacht werden. Die dabei vom Typo3-Projekt benötigten JavaScript-Dateien wurden dafür unverändert in eine neu erzeugte Rails 3.1-Anwendung integriert und als Backend-Bereich in der Rails-Anwendung zum Aufruf über den Browser freigeschalten. Die erzeugte Anwendung steht dabei als Download-Paket und Online-Version zur Verfügung:

Download Quellcode	https://github.com/skeller1/phoenixR
Online-Version	http://phoenix.herokuapp.com/
Nutzername	demo
Passwort	demo

5.2.4 Vorteile der Typo3-Nutzeroberfläche

Die Nutzung der Typo3-Nutzeroberfläche (und EXT JS) ergibt folgende Vorteile für Anwender und Entwickler:

- Ein Modularer Aufbau der Nutzoberfläche mit Möglichkeiten der Überschreibung und Erweiterung bestehender Komponenten der Nutzeroberfläche ist möglich.
- Die Verfügbarkeit erprobter, flexibler und vielfach genutzter UI-Komponenten (z.B. variable Listen, Dialoge, Baumstrukturen u.a.) ermöglicht die Realisierung einer konsistenten Nutzeroberfläche.
- Das Backend (Backend-View, Bilder, CSS und JavaScript) wird einmalig vom Server ausgeliefert und steht danach zur Nutzung im Browser zur Verfügung. Nur bei Bedarf werden Anfragen an den Server gestartet, um neue Daten für Nutzerinteraktionen bereit zustellen.

¹⁵Cross Site Request Forgery beschreibt einen Angriff auf eine Webanwendung durch Aufruf einer manipulierenden Aktion über eine beliebige andere Internetseite. Nähere Erläuterungen unter folgender Adresse: <http://guides.rubyonrails.org/security.html#cross-site-request-forgery-csrf>

- EXT JS garantiert eine größtmögliche Kompatibilität zu aktuellen Internet-Browsern (Internet Explorer, Firefox, Safari, Chrome u.a)

6 Zusammenfassung

6.1 Fazit

Im Rahmen dieser Arbeit konnten die ausgewählten Ruby on Rails Web Content Management Systeme Alchemy CMS, Browser CMS, Locomotive CMS und Refinery CMS auf ihre Webpublishing-Fähigkeiten untersucht werden. Dabei zeigte sich, dass der allgemein in Content Management Systemen etablierte Redaktionsprozess (Erstellung, Verwaltung, Publikation und Archivierung) sehr unterschiedlich unterstützt wird. Ein Einsatz der Systeme kann daher nur empfohlen werden, wenn in einer Voranalyse die Anforderungen an die tatsächlich umzusetzende Internetseite genau definiert und mit den gebotenen Funktionalitäten der Rails WCMS verglichen wird. Trotz der in Rails vorhandenen Möglichkeit zur individuellen Anpassung der Systeme ist der damit erforderliche Mehraufwand nicht vertretbar. Vorallem der Vergleich zu verbreiteten Lösungen wie Typo3 oder Drupal, die im Bereich des Webpublishing bereits einen Großteil der hier untersuchten Funktionalitäten erfüllen, lässt eine eigenständige Weiterentwicklung der bestehenden Rails WCMS nur beschränkt sinnvoll erscheinen. Die in Kapitel 4 aufgezeigten Implementierungsdetails zeigen darüber hinaus, dass die Systeme hinsichtlich ihres Datenbankdesigns und der User-Interface-Umsetzung Optimierungspotenzial besitzen. Vor allem der in Rails übliche Einsatz von Generatorskripten und HTML-Views bei der Erweiterungsentwicklung lässt die Arbeit mit diesen Systemen schnell unübersichtlich und aufwendig erscheinen. Zusätzlich ergeben sich starke Abhängigkeiten zwischen dem WCMS und den vorhandenen Erweiterungen.

6.2 Ausblick

Die vorgestellten WCMS sind zum Teil erst innerhalb der letzten 2 Jahre (2009) an die Öffentlichkeit übergeben wurden. Durch die Leistungen der Open Source-Bewegung

ist daher mit weiteren funktionalen und technischen Verbesserungen zu rechnen. Die Umsetzung eines rails-basierten Content Repository kann dabei einen wichtigen Entwicklungsimpuls innerhalb der Ruby on Rails Web Content Management Systeme signalisieren und eine sinnvolle Vereinfachung für Erweiterungsentwicklungen bedeuten. Die Implementierungsdetails der ausgewählten Systeme bieten tendenziell viel Freiraum für neue Ideen sowie technische und strukturelle Verbesserungen.

7 Anhang

7.1 Liste bestehender Rails 2/3 Web Content Management Systeme bzw. Blogging-Software

OpenSource Web Content Management Systeme mit Rails 2.x Unterstützung		
Projektname	Projektseite im Internet	Aktive Weiterentwicklung
adva-cms	https://github.com/svenfuchs/adva_cms	Ja
Alchemy CMS	http://magiclabs.github.com/alchemy/	Ja
Ansuz CMS	https://github.com/knewter/ansuz	eingestellt
Casein	http://www.caseincms.com/	Ja
Comatose	http://comatose.rubyforge.org/	eingestellt
Compages	http://compages.wordpress.com/	eingestellt
Geego CMS	http://gitorious.org/geego-cms#more	eingestellt
Mephisto	https://github.com/halorgium/mephisto	eingestellt
Radiant	http://radiantcms.org/	Ja
Railfrog	http://railfrog.com/	Nein
Rojo CMS	https://github.com/onomojo/rojo	unbekannt
Rubricks	http://rubricks.org/	eingestellt
Skyline CMS	http://www.skylinecms.nl/	Ja
Station	https://github.com/atd/station	Ja
Typhus	http://typus.heroku.com/	eingestellt
Vrame	https://github.com/9elements/vrame	eingestellt
Webiva	http://webiva.org/	Ja
zenacms	http://zenadmin.org/	Ja

Open Source Web Content Management Systeme mit Rails 3.x Unterstützung		
Projektname	Projektseite im Internet	Aktive Weiterentwicklung
adva-cms2	https://github.com/svenfuchs/adva-cms2	Ja
Alchemy CMS	http://magiclabs.github.com/alchemy/	Ja
Browser CMS	http://browsercms.org	Ja
Casein	https://github.com/spoiledmilk/casein3	Ja
Comfortable Mexican Sofa	https://github.com/twg/comfortable-mexican-sofa	ja
Locomotive CMS	http://www.locomotivecms.com/	Ja
Nesta CMS	https://github.com/gma/nesta	ja
Refinery CMS	http://refinerycms.com/	Ja
Surtout CMS	http://surtoutcms.com/	noch nicht veröffentlicht
typo (Blogging)	http://fdv.github.com/typo/	Ja

7.2 Funktionsumfang der untersuchten Rails WCMS

Die Ergebnisse der durchgeführten funktionalen Analyse (Kapitel 3.7) werden im Folgenden für jedes System noch einmal separat in einer Übersicht zusammengefasst:

Alchemy CMS

Erstellung

- Umfassende Möglichkeiten zur Erstellung und Verwaltung von Seiten und verschiedenen Inhaltselementen in verschiedenen Sprachen
- Template-Erstellung und -verwaltung findet im Quellcode der Rails-Anwendung statt
- Umsetzung komplexer Seitenlayouts möglich

Kontrolle

- eingeschränkte Rechteverwaltung mit festgelegten Gruppen für Front- und Backend (Registriert, Author, Redakteur, Administrator)
- kein umfassendes Linkmanagement

Freigabe

- Einfacher Freigabeprozess für Internetseiten und Inhalte
- keine umfassende Workflowfunktionalität

Publikation

- Trennung der Inhalte vom Design durch Verwendung von Templates
- umfassende Seiten- und Inhaltsverwaltung mit Möglichkeiten zum Kopieren und Verschieben der Inhalte (Zwischenablagefunktion)

Terminierung/Archivierung

- Seiten und Inhalte können nicht archiviert oder zeitraumgebunden veröffentlicht werden

Browser CMS

Erstellung

- Umfassende Möglichkeiten zur Erstellung und Verwaltung von einsprachigen Seiten und verschiedenen Inhaltselementen
- Template-Erstellung und -verwaltung im Backend der Anwendung
- Umsetzung komplexer Seitenlayouts möglich

Kontrolle

- Rechteverwaltung mit vordefinierten Gruppen für Front- und Backend sowie Möglichkeiten zum Erstellen eigener Gruppen
- kein umfassendes Linkmanagement

Freigabe

- Einfacher Freigabeprozess für Internetseiten und Inhalte
- keine umfassende Workflowfunktionalität

Publikation

- Trennung der Inhalte vom Design durch Verwendung verschiedener Template-Sprachen
- Wiederverwendung von Inhaltselementen

Terminierung/Archivierung

- Archivierung von erstellten Seiten
- keine zeitraumgebundene Veröffentlichung von Seiten und Inhalten möglich

Locomotive CMS

Erstellung

- Umfassende Möglichkeiten zur Erstellung und Verwaltung von einsprachigen Seiten und verschiedenen Inhaltselementen
- Inhaltselemente können im Backend der Seite individuell erstellt werden
- Umsetzung komplexe Layouts mit verschiedenen Seiten- und Inhaltselementen möglich

Kontrolle

- eingeschränkte Rechteverwaltung mit vordefinierten Gruppen für das Backend
- kein umfassendes Linkmanagement

Freigabe

- keine Abbildung von Freigabeprozessen für Internetseiten und Inhalte möglich
- keine Workflowfunktionalität

Publikation

- Trennung der Inhalte vom Design durch Verwendung der Template-Sprache Liquid
- Wiederverwendung von Inhaltselementen
- eingeschränkte Möglichkeiten zum Verschieben von Seiten und Inhalten

Terminierung/Archivierung

- Archivierung und Terminierung von Inhalten und Seiten ist nicht möglich

Refinery CMS

Erstellung

- Erstellung und Verwaltung von mehrsprachigen Seiten und Inhaltsböcken
- Inhaltselemente können im Backend der Seite individuell erstellt werden
- Umsetzung komplexe Layouts mit Hilfe von Rails Views möglich

Kontrolle

- keine Rechteverwaltung für Seiten, Inhalte und Nutzer
- kein umfassendes Linkmanagement

Freigabe

- keine Unterstützung von Freigabeprozessen
- Erstellte Seiten können in den Bearbeitungsmodus gesetzt werden und sind dadurch nicht mehr erreichbar
- keine Workflowfunktionalität

Publikation

- Trennung der Inhalte vom Design durch Verwendung der in Rails üblichen ERB-Views
- keine Wiederverwendung von Inhaltselementen möglich
- keine Möglichkeiten zum Verschieben und Kopieren von Inhalten

Terminierung/Archivierung

- Archivierung und Terminierung von Inhalten und Seiten ist nicht möglich

7.3 Crudify-Methode von Refinery CMS 1.0.8

Mit Hilfe der Metaprogrammierung in Ruby wird zur Laufzeit der Anwendung das gesamte Codegerüst der für Refinery-Erweiterungen benötigten Backend-Controller erzeugt. Der Aufruf der crudify-Methode (*crudify* ist abgeleitet von CRUD, das als Akronym für Create, Read, Update und Delete steht) erzeugt dabei die benötigten Controller-Aktionen zum Erstellen, Löschen und Ändern der entsprechenden Controller-Ressource(z.B. ProjectsController => Project-Ressource). Für ein besseres Verständnis wird der Quellcode der crudify-Methode aufgeführt, um den tatsächlich zur Laufzeit erzeugten Controller-Quellcode darzustellen:

Implementierung der Crudify-Methode in Refinery CMS 1.0.8

```
1 # Base methods for CRUD actions
2 # Simply override any methods in your action controller you want to be customised
3 # Don't forget to add:
4 #   resources :plural_model_name_here
5 # or for scoped:
6 #   scope(:as => 'module_module', :module => 'module_name') do
7 #     resources :plural_model_name_here
8 #   end
9 # to your routes.rb file.
10 # Full documentation about CRUD and resources go here:
11 # -> http://api.rubyonrails.org/classes/ActionDispatch/Routing/Mapper/Resources.html#method-i-resources
12 # Example (add to your controller):
13 # crudify :foo, :title_attribute => 'name' for CRUD on Foo model
14 # crudify :'foo/bar', :title_attribute => 'name' for CRUD on Foo::Bar model
15 # Note: @singular_name will result in @foo for :foo and @bar for :'foo/bar'
16
17 module Refinery
18   module Crud
19
20     def self.default_options(model_name)
21       singular_name = model_name.to_s.split('/').last
22       class_name = "::#{model_name.to_s.camelize.gsub('/', ' '::')}".gsub(':::', '::')
23       plural_name = singular_name.to_s.gsub('/', '_').pluralize
24       this_class = class_name.constantize.base_class
25       {
26         :conditions => '',
27         :include => [],
28         :order => ('position ASC' if this_class.table_exists? and this_class.
29           column_names.include?('position')),
30         :paging => true,
31         :per_page => false,
32         :redirect_to_url => "main_app.refinery_admin_#{plural_name}_path",
33         :searchable => true,
```

```
33     :search_conditions => '',
34     :sortable => true,
35     :title_attribute => "title",
36     :xhr_paging => false,
37     :class_name => class_name,
38     :singular_name => singular_name,
39     :plural_name => plural_name
40   }
41 end
42
43 def self.append_features(base)
44   super
45   base.extend(ClassMethods)
46 end
47
48 module ClassMethods
49   def crudify(model_name, options = {})
50     options = ::Refinery::Crud.default_options(model_name).merge(options)
51     class_name = options[:class_name]
52     singular_name = options[:singular_name]
53     plural_name = options[:plural_name]
54
55     module_eval %(
56       def self.crudify_options
57         #{options.inspect}
58       end
59
60       prepend_before_filter :find_#{singular_name},
61                             :only => [:update, :destroy, :edit, :show]
62
63       def new
64         @_#{singular_name} = #{class_name}.new
65       end
66
67       def create
68         # if the position field exists, set this object as last object, given the
69           conditions of this class.
70         if #{class_name}.column_names.include?("position")
71           params[:_#{singular_name}].merge!({
72             :position => ((#{class_name}.maximum(:position, :conditions => #{
73               options[:conditions].inspect})||-1) + 1)
74           })
75         end
76
77         if (@_#{singular_name} = #{class_name}.create(params[:_#{singular_name}])).
78             valid?
79           (request.xhr? ? flash.now : flash).notice = t(
80             'refinery.crudify.created',
81             :what => "'\#{@_#{singular_name}.#{options[:title_attribute]}'"
```

```
81     unless from_dialog?  
82       unless params[:continue_editing] =~ /true|on|1/  
83         redirect_back_or_default("#{options[:redirect_to_url]}")  
84       else  
85         unless request.xhr?  
86           redirect_to :back  
87         else  
88           render :partial => "/refinery/message"  
89         end  
90       end  
91     else  
92       render :text => "<script>parent.window.location = '\#{options[:  
          redirect_to_url]}';</script>"  
93     end  
94   else  
95     unless request.xhr?  
96       render :action => 'new'  
97     else  
98       render :partial => "/refinery/admin/error_messages",  
99         :locals => {  
100           :object => @#{singular_name},  
101           :include_object_name => true  
102         }  
103     end  
104   end  
105 end  
106  
107 def edit  
108   # object gets found by find_#{singular_name} function  
109 end  
110  
111 def update  
112   if @#{singular_name}.update_attributes(params[:#{singular_name}])  
113     (request.xhr? ? flash.now : flash).notice = t(  
114       'refinery.crudify.updated',  
115       :what => "'\#{@#{singular_name}.#{options[:title_attribute]}'"  
116     )  
117  
118     unless from_dialog?  
119       unless params[:continue_editing] =~ /true|on|1/  
120         redirect_back_or_default("#{options[:redirect_to_url]}")  
121       else  
122         unless request.xhr?  
123           redirect_to :back  
124         else  
125           render :partial => "/refinery/message"  
126         end  
127       end  
128     else  
129       render :text => "<script>parent.window.location = '\#{options[:  
          redirect_to_url]}';</script>"
```

```
130     end
131   else
132     unless request.xhr?
133       render :action => 'edit'
134     else
135       render :partial => "/refinery/admin/error_messages",
136             :locals => {
137               :object => @#{singular_name},
138               :include_object_name => true
139             }
140     end
141   end
142 end
143
144 def destroy
145   # object gets found by find_#{singular_name} function
146   title = @#{singular_name}.#{options[:title_attribute]}
147   if @#{singular_name}.destroy
148     flash.notice = t('destroyed', :scope => 'refinery.crudify', :what => "
149       '\#{title}')"
150   end
151
152   redirect_to #{options[:redirect_to_url]}
153 end
154
155 # Finds one single result based on the id params.
156 def find_#{singular_name}
157   @#{singular_name} = #{class_name}.find(params[:id],
158                                         :include => #{options[:include]}.map
159                                           (&:to_sym).inspect})
160 end
161
162 # Find the collection of @#{plural_name} based on the conditions specified
163 # into crudify
164 # It will be ordered based on the conditions specified into crudify
165 # And eager loading is applied as specified into crudify.
166 def find_all_#{plural_name}(conditions = #{options[:conditions]}.inspect)
167   @#{plural_name} = #{class_name}.where(conditions).includes(
168     #{options[:include]}.map(&:to_sym).inspect)
169   ).order("#{options[:order]}")
170 end
171
172 # Paginate a set of @#{plural_name} that may/may not already exist.
173 def paginate_all_#{plural_name}
174   # If we have already found a set then we don't need to again
175   find_all_#{plural_name} if @#{plural_name}.nil?
176
177   per_page = if #{options[:per_page]}.present?.inspect}
178     #{options[:per_page]}.inspect}
179   elsif #{class_name}.methods.map(&:to_sym).include?(:per_page)
180     #{class_name}.per_page
```

```
178         end
179
180         @#{plural_name} = @#{plural_name}.paginate(:page => params[:page], :
181             per_page => per_page)
182
183     end
184
185     # Returns a weighted set of results based on the query specified by the user.
186     def search_all_#{plural_name}
187         # First find normal results.
188         find_all_#{plural_name}({options[:search_conditions].inspect})
189
190         # Now get weighted results by running the query against the results already
191         found.
192         @#{plural_name} = @#{plural_name}.with_query(params[:search])
193     end
194
195     # Ensure all methods are protected so that they should only be called
196     # from within the current controller.
197     protected :find_#{singular_name},
198               :find_all_#{plural_name},
199               :paginate_all_#{plural_name},
200               :search_all_#{plural_name}
201 )
202
203 # Methods that are only included when this controller is searchable.
204 if options[:searchable]
205     if options[:paging]
206         module_eval %(
207             def index
208                 search_all_#{plural_name} if searching?
209                 paginate_all_#{plural_name}
210
211                 render :partial => '#{plural_name}' if #{options[:xhr_paging].inspect}
212                     && request.xhr?
213             end
214         )
215     else
216         module_eval %(
217             def index
218                 unless searching?
219                     find_all_#{plural_name}
220                 else
221                     search_all_#{plural_name}
222                 end
223             end
224         )
225     end
226
227 else
228     if options[:paging]
229         module_eval %(
```



```
226     def index
227       paginate_all_#{plural_name}
228
229       render :partial => '#{plural_name}' if #{options[:xhr_paging].inspect}
230         && request.xhr?
231     end
232   )
233 else
234   module_eval %(
235     def index
236       find_all_#{plural_name}
237     end
238   end
239
240 end
241
242 if options[:sortable]
243   module_eval %(
244     def reorder
245       find_all_#{plural_name}
246     end
247
248     # Based upon http://github.com/matenia/jQuery-Awesome-Nested-Set-Drag-and-Drop
249
250     def update_positions
251       previous = nil
252       params[:ul].each do |_, list|
253         list.each do |index, hash|
254           moved_item_id = hash['id'].split(/#{singular_name}\\_?/)
255           @current_#{singular_name} = #{class_name}.find_by_id(moved_item_id)
256
257           if @current_#{singular_name}.respond_to?(:move_to_root)
258             if previous.present?
259               @current_#{singular_name}.move_to_right_of(#{class_name}.
260                 find_by_id(previous))
261             else
262               @current_#{singular_name}.move_to_root
263             end
264           else
265             @current_#{singular_name}.update_attribute(:position, index)
266           end
267
268           if hash['children'].present?
269             update_child_positions(hash, @current_#{singular_name})
270           end
271
272           previous = moved_item_id
273         end
274       end
275     end
276   end
277 end
```

```

274     #{class_name}.rebuild! if #{class_name}.respond_to?(:rebuild!)
275     render :nothing => true
276   end
277
278   def update_child_positions(node, #{singular_name})
279     node['children']['0'].each do |_, child|
280       child_id = child['id'].split(/#{singular_name}\_?/)
281       child_#{singular_name} = #{class_name}.where(:id => child_id).first
282       child_#{singular_name}.move_to_child_of(#{singular_name})
283
284       if child['children'].present?
285         update_child_positions(child, child_#{singular_name})
286       end
287     end
288   end
289 )
290 end
291
292 module_eval %(
293   def self.sortable?
294     #{options[:sortable].to_s}
295   end
296
297   def self.searchable?
298     #{options[:searchable].to_s}
299   end
300 )
301
302
303   end
304 end
305 end
306 end

```

7.4 Ext-Direct Spezifikation für Ext Js 3.0

Im folgenden Abschnitt wird die offizielle PDF-Version der Ext Js 3.0-Spezifikation als PDF-Version abgebildet. Die Formatierung und Gestaltung entspricht dabei der originalen PDF-Version vom 25.Oktober 2011. Zum Zeitpunkt der Veröffentlichung dieser Arbeit existierth keine Spezifikation für die neuste Version 4.x der Ext Js-Bibliothek.

Ext.Direct

Specification Draft

Ext.Direct is a platform and language agnostic technology to remote server-side methods to the client-side. Furthermore, Ext.Direct allows for seamless communication between the client-side of an Ext application and any server-side platform. Ext 3.0 will ship with 5 server-side implementations for Ext.Direct. Each of these are called Ext.Direct stacks (need better name).

Server-side Stacks available at Ext 3.0 release:

- PHP
- Java
- .NET
- ColdFusion
- Ruby – (Merb)

There are many optional pieces of functionality in the Ext.Direct specification. The implementor of each server-side implementation can include or exclude these features at their own discretion. Some features are not required for particular languages or may lack the features required to implement the optional functionality.

Some examples of these features are:

- Remoting methods by metadata – Custom annotations and attributes
- Programmatic, JSON or XML configuration
- Type conversion
- Asynchronous method calls
- Method batching
- File uploading

This specification is being released into the open so that community members can make suggestions and improve the technology. If you do implement a router in a different language and/or server-side framework and are interested in contributing it back to Ext, please let us know.

An Ext.Direct server-side stack needs at least 3 key components in order to function.

The name and job of each of these components:

- Configuration – To specify which components should be exposed to the client-side
- API – To take the configuration and generate a client-side stub

- Router – To route requests to appropriate classes, components or functions

Configuration

There are 4 typical ways that a server-side will denote what classes need to be exposed to the client-side. These are Programmatic, JSON, XML configurations and metadata.

Languages which have the ability to do dynamic introspection at runtime may require less information about the methods to be exposed because they can dynamically determine this information at runtime.

Some considerations that are language specific:

- Ability to specify named arguments to pass an argument collection. When using an argument collection order of the arguments does not matter.
- Ability to make an argument optional
- Ability to dynamically introspect methods at runtime
- Ability to associate metadata with classes or methods
- Requirement to convert an argument to a specific class type before invoking the method on the server-side
- Requirement to specify how many arguments each method will get

As you can see different languages require different approaches to describe their server-side methods. What may work for one language will may not be enough information the other. The important outcome of the configuration is to describe the methods that need to be remotable, their arguments and how to execute them.

Programmatic Configurations:

Programmatic configuration builds the configuration by simply creating key/value pairs in the native language. (Key-Value Data Structures are known by many names: (HashMap, Dictionary, Associative Array, Structure, Object). Please use the term which feels most natural to the server-side of your choice.

PHP Example:

```
$API = array(  
    'AlbumList'=>array(  
        'methods'=>array(  
            'getAll'=>array(  
                'len'=>0  
            ),  
            'add'=>array(  
                'len'=>1  
            )  
        )  
    )  
);
```

Here we are instructing the server-side that we will exposing 2 methods of the AlbumList class getAll and add. The getAll method does not accept any arguments and the add method accepts a single argument. The PHP implementation does not need to know any additional information about the arguments, their name, their type or their order. Other server-side implementations may need to know this information and will have to store it in the configuration.

JSON Configuration:

JSON Configuration stores the exact same information in a JSON file on the file-system and then reads that in.

```
{
  AlbumList: {
    methods: {
      getAll: {
        len: 0
      },
      add: {
        len: 1
      }
    }
  }
}
```

XML Configuration:

XML Configuration stores the exact same information in an XML file on the file-system and then reads that in.

```
<AlbumList>
  <methods>
    <method name="getAll" len="0" />
    <method name="add" len="1" />
  </methods>
</AlbumList>
```

Configuration by Metadata:

PHP does not support adding custom metadata to methods.

Consider this example of a ColdFusion component which has added a custom attribute to each method (cffunction) called remotable.

ColdFusion Example:

```
<cfcomponent name="AlbumList">
  <cffunction name="getAll" remotable="true">
    <cfreturn true />
  </cffunction>
  <cffunction name="add" remotable="true">
    <cfargument name="album" />
    <cfreturn true />
  </cffunction>
</cfcomponent>
```

The ColdFusion Ext.Direct stack is able to introspect this component and determine all of the information it needs to remote the component via this custom attribute.

API

The API component of the Ext.Direct stack has an important function. It's job is to transform the configuration about the methods to be remotored into client-side stubs. By generating these proxy methods we can seamlessly call server-side methods as if they were client-side methods without worrying about the interactions between the client and server-side.

The API component will be included via a script tag in the head of the application. The server-side will dynamically generate an actual JavaScript document to be executed on the client-side.

The JS will describe the methods which have been read in via the configuration file. This example uses the variable name Ext.app.REMOTING_API. It will describe the url, the type and available actions. (Class and method).

By including the API.php file via a script tag. The server side will return the following:

```
Ext.app.REMOTING_API = {
  "url": "remote/router.php",
  "type": "remoting",
  "actions": {
    "AlbumList": [{
      "name": "getAll",
      "len": 0
    }, {
      "name": "add",
      "len": 1
    }]
  }
};
```

Router

The router accepts requests from the client-side and *routes* them to the appropriate Class, method and passes the proper arguments.

Requests can be sent in two ways, via JSON-Encoded raw HTTP Post or a form post. When uploading files the form post method is required.

JSON-Encoded raw HTTP posts look like:

```
{"action": "AlbumList", "method": "getAll", "data": [], "type": "rpc", "tid": 2}
```

The router needs to decode the raw HTTP post. If the http POST is an array, the router is to dispatch multiple requests. A single request has the following attributes:

- action – The class to use
- method – The method to execute
- data – The arguments to be passed to the method – array or hash
- type – “rpc” for all remoting requests
- tid – Transaction ID to associate with this request. If there are multiple requests in a single POST these will be different.

Form posts will be sent the following form fields.

- extAction – The class to use
- extMethod – The method to execute
- extTID – Transaction ID to associate with this request
- extUpload – (optional) field is sent if the post is a file upload
- Any additional form fields will be assumed to be arguments to be passed to the method.

Once a request has been accepted it must be dispatched by the router. The router must construct the relevant class and call the method with the appropriate arguments which it reads from *data*. For some languages this will be an array, for others it will be a hash. If it is an array then the order of the arguments will matter. If it is hash then the named arguments will be passed as an argument collection.

NOTE: At this point there is no mechanism to construct a class with arguments. It will ONLY use the default constructor.

The response of each request should have the following attributes in a key-value pair data structure.

- type – 'rpc'
- tid – The transaction id that has just been processed
- action – The class/action that has just been processed
- method – The method that has just been processed
- result – The result of the method call

This response will be JSON encoded. The router can send back multiple responses with a single request enclosed in an array.

If the request was a form post and it was an upload the response will be sent back as a valid html document with the following content:

```
<html><body><textarea></textarea></body></html>
```

The response will be encoded as JSON and be contained within the textarea.

Form Posts can only execute one method and do not support batching.

If an exception occurs on the server-side the router should also return the following error information when the router is in **debugging** mode.

- type - 'exception'
- message - Message which helps the developer identify what error occurred on the server-side
- where - Message which tells the developer where the error occurred on the server-side.

This exception handling within the router should have the ability to be turned on or off. This exception information should never be sent back to the client in a production environment because of security concerns.

Exceptions are meant for server-side exceptions. Not application level errors.

Optional Router Features

- Ability to specify before and after actions to execute allowing for aspect oriented programming. This is a powerful concept which can be used for many uses such as security.

Client Side portions of Ext.Direct

To use Ext.Direct on the client side you will need to add each provider by the variable name you used in the API.

For example:

```
Ext.Direct.addProvider(Ext.app.REMOTING_API);
```

After adding the provider all of your actions will exist in the global namespace.

The client-side will now be able to call the exposed methods as if they were on the server-side.

- AlbumList.getAll
- AlbumList.add

An additional argument will be added to the end of the arguments allowing the developer to specify a callback method. Because these methods will be executed asynchronously it is important to note that we must use the callback to process the response. We will not get the response back immediately.

For example we do NOT want to do this:

```
var albums = AlbumList.getAll();
```

This code should be written like this:

```
AlbumList.getAll(function(url, response) {  
    // process response  
});
```

7.5 Nutzung von Extr in einer Rails 3.1-Anwendung

Im folgenden Abschnitt wird die Installation und Nutzung des erstellten Plugins Extr in einer neu erstellten Rails 3-Anwendung demonstriert. Das erstellte Paket ist noch in der Entwicklungsphase und konnte nicht vollständig auf seine Sicherheit überprüft werden². Der Quellcode der Anwendung kann unter folgender Adresse heruntergeladen werden: Extr: <https://github.com/skeller1/extr> Für die folgenden Beschreibungen wird eine funktionsbereite Rails 3.1-Installation vorausgesetzt.

1. Erstellung einer neuen Rails 3-Anwendung *directtest*

```
rails new directtest
```

2. Aktivierung des Plugins/Gems in der Gemfile der erstellten Rails-Anwendung

```
# ...  
gem "extr", :git => "git://github.com/skeller1/extr.git"  
# ...
```

3. Erstellung einer Rails 3 Standard-Ressource Projekt und Erzeugung der notwendigen Datenbanktabelle.

```
rails g scaffold project name:string description:text  
rake db:migrate
```

4. Einbinden der notwendigen Ext-Direct-Bibliotheken mit Hilfe der vom Plugin zur Verfügung gestellten Helfermethoden (View Helpers). Der Namensraum des Ext-Direct-Provider ist *Rails*.

app/views/layouts/application.html.erb

```
<!DOCTYPE html>  
<html>  
<head>  
  <title>Extr</title>  
  <%= stylesheet_link_tag "extr/application" %>  
  <%= javascript_include_tag "extr/application" %>  
  <%= csrf_meta_tags %>  
  <%= ext_base_tag %>  
  <%= ext %>
```

²Sicherheitsbedenken in den Bereichen CSRF und XSS wurden bei der Vorstellung des Paktes aufgezeigt.

```
<%= ext_direct_provider "Rails" %>
</head>
<body>
  <%= yield %>
</body>
</html>
```

5. Registrierung der Ext-Direct-fähigen Controller-Methoden (Actions) im Projekte-Controller (projects controller). Die erstellten Methoden sind nur als Beispiel zu verstehen und können entsprechend angepasst werden.

app/controllers/projects_controller.rb

```
class ProjectsController < ApplicationController

  # activate extr for this controller
  include Extr::DirectController

  #disable Rails 3 authenticity_token for demonstration, not recom. in production
  skip_before_filter :verify_authenticity_token

  #register 2 ext direct controller action with different, optional controller
  name (MyDirectController)
  direct "MyDirectController",
    :getChildProject => 1,
    :someOtherMethod => 2

  def getChildProject
    # render a random project name as json response
    render :json => { :name => "Project#{Random.rand(11)}" }.to_json
  end
end
```

6. Im Index-HTML-View der Projekt-Ressource (<http://localhost:3000/projects>) kann nun durch einen JavaScript-Aufruf eine Anfrage an die registrierten Ext-Direct-Controller-Methoden gesendet werden. Abbildung 7.1 zeigt das Ergebnis eines mit Hilfe des Mozilla Firefox Plugins Firebug temporär ausgeführten JavaScript-Codes. Als JavaScript-Callback wird der Name des Projektes zurückgegeben (*Project9*).

JavaScript zum Aufruf des Projekt-Controllers in der Rails-Anwendung

```
1 Rails.MyDirectController.getChildProject("param",function(r,e){
2   alert(r.name);
3 });
```

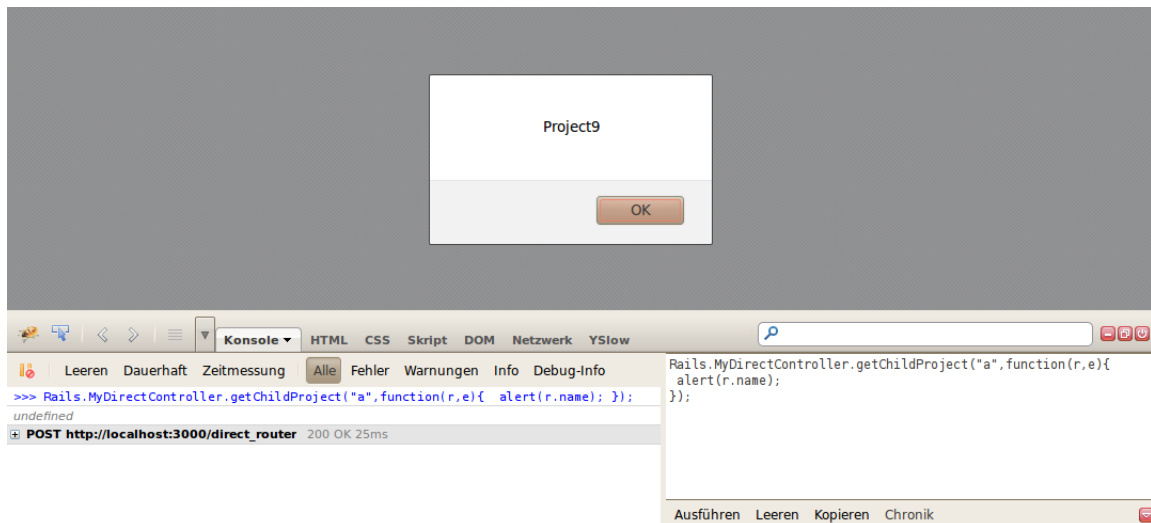


Abbildung 7.1: Mit Hilfe von Firebug formulierte JavaScript-Ext-Direct-Anfrage mit ausgegebener Antwort der Rails-Anwendung

8 Literaturverzeichnis

- [Aim10] AIMONETTI, Matt: *Rails and the Enterprise*. <http://weblog.rubyonrails.org/2010/3/24/rails-and-the-enterprise>. Version: 2010. – Zugriff: 25.10.2011
- [Ber04] BERCHTENBREITER, Ralph: *Grundlagen von Content-Management-Systemen und Ansätze ihrer Bedeutung für das CRM*. 2004
- [BL06] BERNHARD LAHRES, Gregor R.: *Praxisbuch Objektorientierung – Professionelle Entwurfsverfahren*. Galileo Computing, 2006
- [Egg07] EGGER, Frank: *Einführung in Ruby on Rails. Architektur und Konzepte*. <http://www.slideshare.net/foobar2605/einfhrung-in-ruby-on-rails>. Version: 2007. – Zugriff: 25.10.2011
- [Fow03] FOWLER, Martin: *Patterns für Enterprise Application-Architekturen*. mitp, 2003
- [Gü07] GÜNTHER, Tobias: *Theorie einer neuen Web-Architektur in Ruby On Rails: REST in Rails*. <http://t3n.de/magazin/rest-rails-teil-1-theorie-neuen-web-architektur-ruby-219794/>. Version: 2007. – Zugriff: 25.10.2011
- [HM10] HUSSEIN MORSY, Tanja O.: *Ruby on Rails 2*. Galileo Computing, 2010
- [jdk06] JDK: *Web Content Management Featurematrix*. <http://www.jdk.de/de/cms/wcm-cms-web-content-management/wcm-featurematrix.html>. Version: 2006. – Zugriff: 25.10.2011
- [Neu07] NEUKIRCHEN, Christian: *Introducing Rack*. <http://chneukirchen.org/talks/euruko-2007/neukirchen07introducingrack.pdf>. Version: 2007. – Zugriff: 25.10.2011

- [PD11] PETRA DREWER, Wolfgang Z.: *Technische Dokumentation. Übersetzungsge-
rechte Texterstellung und Content-Management*. 2011. –
<http://rubygems.org/gems/rails>
- [Per10] PERROTTA, Paolo: *Metaprogramming Ruby*. Pragmatic Programmers, 2010
- [Rai11a] RAILS: *Rails 3.1*. 2011. –
<http://rubygems.org/gems/rails>
- [Rai11b] RAILS: *Rails on Rack*. http://guides.rubyonrails.org/rails_on_rack.html. Version: 2011. – Zugriff: 25.10.2011
- [Rig09] RIGGERT, Wolfgang: *ECM - Enterprise Content Management*. Vieh-
weg+Teubner, 2009
- [Rit10] RITTER, Andreas: *SWOT-Analyse zu Content-Management-Systemen*. Grin
Verlag, 2010
- [Roc02] ROCKLEY, Ann: *Managing Enterprise Content: A Unified Content Strategy*.
New Riders Press, 2002
- [Sch08] SCHMIDT, Maik: *Enterprise Recipes with Ruby and Rails*. Pragmatic Program-
mers, 2008
- [w3t11] W3TECHS: *Usage of server-side programming languages for websites*. [http:
//w3techs.com/technologies/overview/programming_language/all](http://w3techs.com/technologies/overview/programming_language/all).
Version: 2011. – Zugriff: 25.10.2011

Tabellenverzeichnis

2.1	Rails Routing der Rest-Ressource Projekt	15
2.2	Vergleich zwischen Rest-konformen und klassischen Rails-URL's	15
3.1	Steckbrief Alchemy CMS	33
3.2	Steckbrief Browser CMS	37
3.3	Steckbrief Locomotive CMS	43
3.4	Steckbrief Refinery CMS	47
3.5	Mögliche Auswertungsstufen für die umgesetzten Funktionalitäten der WCMS	51

Abbildungsverzeichnis

1.1	Nutzung verschiedener Programmiersprachen auf Servern	5
2.1	Verarbeitung einer Anfrage innerhalb des Rails-Frameworks	12
2.2	Rack als Vermittler zwischen Server und Ruby-Webframeworks	16
2.3	Ausgabe der Rack-Anwendung MyRackApp im Browser	18
2.4	Möglichkeiten der unterschiedlichen Rückgabewerte durch Rack Middle- wares und Rails	19
2.5	Prozessschritte im Content Life Cycle	21
2.6	Feature-Matrix für Web Content Management Systeme	24
3.1	Ruby Gems mit Informationen zum aktuellen Rails 3.1.1	31
3.2	Backend-Ansicht von Alchemy CMS mit geöffneter Seitenvorschau und Elementebearbeitung (Seitenbearbeitungsmodus)	34
3.3	Standardset von Inhaltselementen in Alchemy CMS	36
3.4	Backend-Ansicht von Browser CMS	38
3.5	Ausgewählte Seite im Bearbeitungsmodus und aktiviertem Visual Editor	39
3.6	Tag-Portlet mit Template im Backend von Browser CMS	40
3.7	Erstellung eines neuen Datensatzes vom Inhaltstyp (Content Block) Text mit Hilfe des FCKEditor	41
3.8	Backend von Locomotive CMS mit geöffnetem Seitenbaum (Ausschnitt) .	44
3.9	Ein im Backend von Locomotive CMS zusammengestelltes Inhaltselement (Baustein)	45
3.10	Backend-Ansicht von Refinery CMS	48
3.11	Seitenbearbeitung in Refinery CMS mit geöffneten Content Sections Body und Site Body	48

4.1	Ressourcen-Auflistung ohne Möglichkeiten der Strukturierung in Alchemy CMS. Für die anderen Systeme ergibt sich ein ähnliches Gesamtbild. . . .	80
5.1	JCR-Datenmodell	83
5.2	Frontend-Ansicht der Typo3 5.0 Sprint Release 6 Demoversion	86
5.3	Backend-Ansicht der Typo3 5.0 Sprint Release 6 Demoversion	87
5.4	Schema des Ext Direct Routing mit <i>extr</i>	89
7.1	Mit Hilfe von Firebug formulierte JavaScript-Ext-Direct-Anfrage mit ausgegebener Antwort der Rails-Anwendung	117

Quelltext

2.1	Konfigurationsdatei im Java Spring Framework	10
2.2	Beispiel für eine einfache Rack-Anwendung	17
2.3	Aufruf des Generators zur Erstellung einer MVC-Ressource Projekt mit Ausgabe der erstellten Dateien	20
3.1	Aufruf und Ausgabe des Generator-Skripts zur Erstellung eines neuen Inhaltselements Produkt	42
4.1	Aufruf des Refinery Engine Generators mit Ausgabe des erzeugten Code- gerüsts	74
4.2	Projects-Controller mit verwendeter crudify-Methode und optionalen Pa- rametern	78
4.3	Beispiel für eine JavaScript-Funktion zur Realisierung des Bildauswahl- dialogs in Refinery CMS. Das Codebeispiel zeigt ebenfalls die Abhängig- keit zu dem verwendeten HTML-Markup (z.B. Zeile 18)	79
5.1	Ajax-Anfrage an einen Server im jQuery-Framework	88

Selbstständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig und nur unter ausschließlicher Verwendung der angegebenen Literatur, Quellen und Hilfsmittel angefertigt habe.

Ort, Datum

Stephan Keller