Infopark CMS Fiona

Rails Connector für CMS Fiona

Infopark CMS Fiona

Rails Connector für CMS Fiona

Inhalt

1

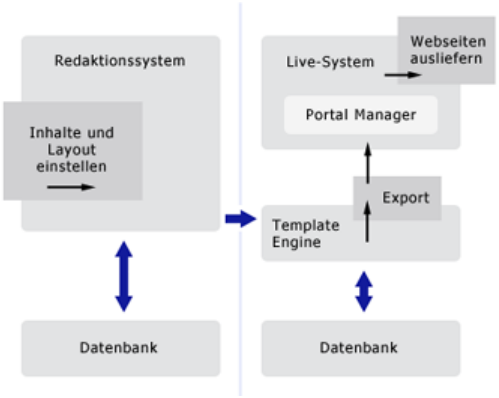
1 Konzepte

1.1 Umstellung auf Rails?

Die folgende Darstellung soll Ihnen dabei helfen, die Unterschiede zwischen einem klassisch betriebenen Infopark CMS Fiona und einem mit dem Rails Connector betriebenen Fiona zu verstehen, damit Sie entscheiden können, ob der Umstieg für Sie in Betracht kommt.

1.1.1 Redaktions- und Live-System

Infopark CMS Fiona besteht aus einem Redaktionssystem und einem Live-System. Das Redaktionssystem dient dazu, die Inhalte zu erfassen. Ferner können Administratoren hier Datenstrukturen wie Felder, Vorlagen etc. definieren und an die Erfordernisse der Website anpassen.



Das Live-System hat die Aufgabe, aus den Inhalten Webseiten zu erzeugen. Bei einem klassischen Setup werden hierzu die Layouts (Templates) verwendet, die im Redaktionssystem hinterlegt sind. In Layouts sind die Strukturen der Webseiten unabhängig von den Inhalten definiert. Mit einer speziellen Abfragesprache in den Layouts werden die Inhalte beim so genannten Export in die Seitenstrukturen eingefügt, wodurch das Endprodukt entsteht, die tatsächlichen Webseiten.

1.1.2 Gleiche Handhabung von redaktionellen Inhalten und Layouts

Technisch sind Layouts im CMS eine spezielle Art von Inhalten, nämlich solche, die dazu verwendet werden, die redaktionellen Inhalte darzustellen. Sowohl Layouts als auch die redaktionellen Inhalte werden im Redaktionssystem als Dateien gepflegt. Man kann die Layouts dort bearbeiten, an eine andere Stelle in der Dateihierarchie verschieben, mit einem Titel oder anderen Feldern versehen usw.

Dieser einheitliche Umgang mit Layouts und redaktionellen Inhalten ist ein zentrales Konzept in Infopark CMS Fiona. So gibt es Arbeitsversionen und freigegebene Versionen auch bei Layouts,

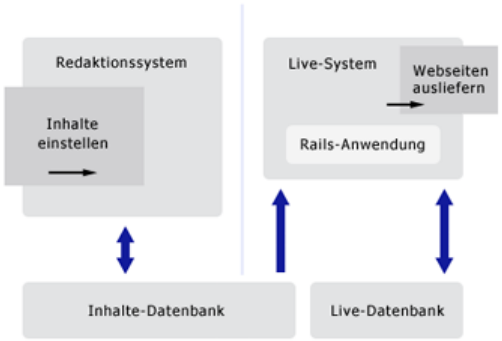
dem Benutzer stehen für beide Inhaltstypen die gleichen Werkzeuge in der gleichen Oberfläche zur Verfügung. Layouts lassen sich archivieren, unterliegen den gleichen Workflowmechanismen wie die Inhalte usw.

1.1.3 Klassisches Live-System ist heterogen

Die Layout-Sprache von CMS Fiona ermöglicht es auf einfache Art, Funktionen des Live-Servers zu nutzen. So können beispielsweise Portlets mit einer einzigen Layout-Anweisung in die Webseiten eingebunden oder Inhalte nur bestimmten Benutzergruppen zugänglich gemacht werden.

Auf der Live-Seite erfordert dieser Ansatz jedoch mehrere, z.T. disparate Technologien. Die Inhalte müssen nach Aktualisierungen nicht nur zum Live-System übertragen und dort exportiert werden. Für die Realisierung dynamischer Server mit Personalisierung, Zugriffsschutz, integrierten Portlets, möglicherweise auch unter Verwendung von JSPs, müssen die Inhalte bei der Auslieferung vom Portal Manager, von der Velocity Engine, von verschiedensten Filtern oder von der JSP Engine verarbeitet werden. Der Einsatz so unterschiedlicher Techniken lässt einen Live-Server schnell unübersichtlich werden.

1.1.4 Ruby on Rails ist ein einheitliches Framework



Im Unterschied hierzu ist Ruby on Rails ein einheitliches Framework. Durch den Infopark Rails Connector können die in der CMS-Datenbank liegenden Inhalte verarbeitet und dargestellt werden. Ein Setup, im Wesentlichen bestehend aus Ruby, Rails, dem Infopark Rails Connector und der Search Engine, ersetzt auf der Live-Seite die Template Engine, den Trifork Application Server und den darin laufenden Portal Manager.

Das Redaktionssystem dagegen bleibt unverändert bestehen. Es wird weiterhin zur komfortablen Erfassung der Inhalte benötigt, zur Versionierung, für die Steuerung der Workflows usw.

Neben der Datenbank für die redaktionellen Inhalte verwenden Rails-Anwendungen oft eine Live-Datenbank, in der die Inhalte der Website-Besucher (Kommentare, Bewertungen, persönliche Einstellungen, etc.) gespeichert sind. Der elegante und für den Entwickler transparente Zugriff auf alle Datenbankinhalte sowie die Kapselung ihrer Darstellung in so genannten Views macht Ruby on Rails zu einem idealen Werkzeug für die zügige Entwicklung moderner Web-2.0-Anwendungen.

1.1.5 Umstieg auf Ruby on Rails

Der Umstieg von einem klassischen Server-Setup mit CMS Fiona allein auf CMS Fiona plus Ruby on Rails sollte in Erwägung gezogen werden, wenn ein Relaunch der Website mit mehr Besucher-Interaktivität geplant ist. Ruby-on-Rails-Projekte werden wie Software-Projekte gehandhabt. Infopark bietet die vollständige Abwicklung und Betreuung solcher Projekte an.

1.1.6 Häufig gestellte Fragen zum Umstieg auf Rails

Wie kompliziert ist der Umstieg auf die Rails-Technologie?

Beim Einsatz des Infopark Rails Connectors müssen alle Layouts und etwa vorhandene kundenspezifische Funktionalitäten (in Form von PHP- oder Java-Code, Portlets) neu geschrieben werden. Der Aufwand hierfür hängt vom Umfang und der Komplexität Ihres Systems ab.

Die Inhalte dagegen können vollständig übernommen werden. So ist es insbesondere möglich, dass Redakteure am System weiter Inhalte einstellen, während im Hintergrund die Layouts und Funktionalität auf Rails umgestellt werden. Sobald letzteres abgeschlossen ist, kann der aktuelle Stand der Inhalte einfach mit dem neuen Rails-Server ausgeliefert werden.

Benötige ich beim Einsatz des Rails Connectors eine Template Engine?

Nein, die Template-Engine hat dann keine sinnvolle Funktion und wird nicht mehr benötigt.

Benötige ich beim Einsatz des Rails Connectors einen Trifork oder einen anderen Java-Application-Server?

Nicht auf dem Live-Server, jedoch nach wie vor auf dem Redaktionssystem, da das GUI in Java implementiert ist und zum Betrieb den Trifork oder einen vergleichbaren Application Server benötigt.

Wird denn Java dann überhaupt noch unterstützt?

Ja, der Java-PM wird weiterhin ausdrücklich unterstützt. Neukunden empfehlen wir jedoch, mit Rails zu arbeiten und nicht mehr mit Java.

Können wir unsere eigenen Portlets weiterverwenden?

Wenn Sie den Rails Connector einsetzen möchten, können Portlets derzeit nicht verwendet werden.

Können wir parallel mit der Rails-Technik und Java arbeiten?

Es ist möglich – wenn auch selten erforderlich –, die Inhalte zweier Instanzen von Fiona auf unterschiedliche Art auszuliefern. Es ist jedoch nicht möglich, bei einer einzelnen Instanz diese beiden Techniken zu kombinieren.

Beim Umstieg auf Rails braucht der Betrieb des Redaktionssystems und der Website nicht unterbrochen zu werden, da die Rails-Anwendung parallel zum laufenden Betrieb des klassischen Fiona-Systems entwickelt werden kann. Hierbei ist der Zugriff auf die Echtdaten möglich, sodass die Rails-Anwendung umfassend getestet werden kann, bevor sie das klassische Live-System vollständig ablöst.

Benötigen wir noch einen Apache Webserver zur Auslieferung der Webseiten?

Ja. Der Apache Webserver sorgt für die korrekte Weiterleitung aller eingehenden HTTP-Requests. Diese Anfragen gibt er an einen oder mehrere (auch über mehrere Maschinen verteilte) `Mongrel`-Server weiter. Mongrel ist gewissermaßen der Rails-Applikationsserver, also etwa das, was der Trifork für Java-Applikationen ist. Der Mongrel sorgt für die tatsächliche Abarbeitung der Anfrage und die Ausgabe des Ergebnisses über den Apache-Server.

Welche Datenbanken werden bei Einsatz des Rails Connectors unterstützt?

Gegenwärtig werden ausschließlich MySQL-Datenbanken unterstützt.

Welche Rolle spielt der Search Server im Zusammenhang mit dem Rails Connector?

Der Rails Connector unterstützt die Nutzung des Search Servers. Daher kann dieser wie bisher normal installiert und weiterbetrieben werden.

Benötige ich eine Versionsverwaltung für mein Rails-Projekt und wenn ja, welche wird empfohlen?

Ein Rails-Projekt lässt sich ohne eine Versionsverwaltung nicht professionell durchführen. Wie zahlreiche andere Unternehmen verwendet Infopark Git. Häufig wird als VCS (*engl.* "version control system") auch Subversion eingesetzt.

Wo finde ich Informationen über Ruby on Rails?

Die Homepage von Ruby on Rails ist http://www.rubyonrails.org/.

1.2 Was ist Ruby on Rails?

Ruby on Rails ist ein Framework, mit dem datenbankbasierte dynamische Websites in vergleichsweise kurzer Zeit entwickelt werden können. Das Framework kapselt den Zugriff auf die Datenbank und die Verarbeitung der Requests und nimmt damit dem Entwickler einen großen Teil der in anderen Umgebungen dafür erforderlichen Programmierarbeit ab.

In Ruby-on-Rails-Projekten haben die Funktionen, die beispielsweise ein Portal bietet, einen größeren Stellenwert als in klassischen Projekten, bei denen statische Inhalte im Vordergrund stehen. Damit ist Ruby on Rails ideal geeignet für Web-2.0-Anwendungen mit ihren Community-Funktionen wie Blogs, Rating usw.

Eine Ruby-on-Rails-Laufzeitumgebung ist ein geschlossenes System, das aus dem Ruby-Interpreter und zahlreichen Skripten, Hilfsprogrammen und Bibliotheken besteht. Eine Rails-Anwendung besteht aus einem Verzeichnisbaum, in dem sich die Konfiguration, die Views, die Controller, das Datenbankmodell und anderes befindet.

Durch die Model-View-Controller-Architektur, d.h. die klare Trennung zwischen den Daten, deren Aufbereitung und deren Anzeige sind die entwickelten Webanwendungen leicht zu pflegen. In dieser Architektur bedeuten:

Model

Das Datenmodell, d.h. die Struktur der Daten und ihre Beziehungen zueinander wird in Ruby on Rails auf Klassen, Methoden und Eigenschaften abgebildet. Eine Klasse entspricht üblicherweise einer Datenbanktabelle, eine Zeile darin einem Objekt, dessen Eigenschaften wiederum den Feldern. Durch diese Object-Relational Mapping genannte Abbildung der Daten auf das objektorientierte Modell der Programmiersprache gibt es bei der Handhabung der Daten keinen Unterschied zu den eingebauten und programmierten Klassen und Objekten.

View

Ein View ist eine meist dynamisch erzeugte Sicht auf Daten. Typischerweise enthält ein View HTML-Text, in den Ruby-Code mittels Processing Instructions eingebettet ist, analog zu beispielsweise PHP. So könnte der Inhalt eines Warenkorbs ein View sein, oder auch ein bestimmter Bereich einer Website.

Controller

Ein Controller ist eine Ruby-Datei, die ausgeführt wird, wenn ein Request mit einem definierten URL-Bestandteil beantwortet werden soll. Ruft beispielsweise ein Kunde auf einer Website seinen Warenkorb auf, so erkennt der Controller dies anhand des entsprechenden Bestandteils in der URL (beispielsweise `page`), ruft die entsprechenden Daten aus der Datenbank ab und bereitet sie auf. Anschließend wird der zum Controller gehörende View verarbeitet und die dadurch entstehende HTML-Seite ausgeliefert.

1.3 Die Funktionen des Infopark Rails Connectors

Der Infopark Rails Connector stellt die mit Infopark CMS Fiona redaktionell gepflegten Inhalte in einer Rails-Anwendung zur Verfügung.

- Der Rails Connector wird in Form eines Gems geliefert, das in eine bestehende Rails-Anwendung eingebunden werden kann. Er enthält Modell-Klassen für den Zugriff auf die CMS-Inhalte und Unterstützung für die Implementierung von Controllers, Helpers und Views für die Auslieferung der Inhalte.
- Ab Infopark CMS Fiona 6.6 kann die Rails-Anwendung in der Vorschau des Redaktionssystems angezeigt werden, so dass Redakteure ihre Inhalte im aktuellen Layout betrachten können. Bearbeitungselemente (zur direkten Bearbeitung von CMS-Inhalten) können ins Layout integriert werden.
- Bei der Auslieferung von Webseiten kann der Rails-Connector die im Redaktionssystem vergebenen Zugriffsrechte berücksichtigen und eine frei gestaltbare Fehlerseite ausgeben, wenn der Website-Besucher nicht angemeldet ist.
- Ferner unterstützt der Rails Connector das Deployment der Rails-Anwendung auf die verschiedenen Systeme (Redaktions-, Staging-, Live-System). Hierfür wird ein Skript für Capistrano mitgeliefert.
- Mittels Marker-Menüs können Bearbeitungselemente in der Vorschau gruppiert angeboten werden, um die redaktionelle Arbeit an den Webseiten zu vereinfachen. Marker-Menüs sind ein- und ausklappbar, damit das Layout in der Vorschau erhalten bleibt.
- Der Rails Connector ermöglicht es, Business-Funktionalität zu Ihrer Website hinzuzufügen und redaktionelle Inhalte mit besucherseitigen Inhalten (*user-generated content*) zu verbinden.

Informationen über die Website-Features des Rails Connectors finden Sie in der Beschreibung der mitgelieferten Demo-Anwendung *Playland*. Einige Features müssen explizit aktiviert werden. Details hierzu sowie zum Gebrauch der APIs finden Sie in den RDocs zum Rails Connector.

1.4 Einsatz-Szenarios für den Rails Connector

Für den Einsatz des Infopark Rails Connectors ergeben sich mehrere Szenarios, die im Folgenden kurz vorgestellt werden.

- **Vorschau im CMS**
  - Für die Vorschau im Redaktionssystem wird eine Ruby-on-Rails-Installation benötigt sowie die Rails-Anwendung, die die Vorschauseiten ausliefert. Diese Anwendung sollte einem Versionierungssystem entnommen werden.
  - Damit die Vorschau auf Benutzerseite funktioniert, wird ein Webserver benötigt, der GUI- und Vorschau-URLs zum entsprechenden Server umleitet. Diese Umleitung muss im Webserver eingerichtet werden. Ferner muss das GUI so konfiguriert werden, dass es die richtigen Vorschau-URLs erzeugt. Siehe Integration der Rails-Vorschau ins GUI.
- **Entwicklung und Layout**

Das Layout der Webseiten ist in den Views der Rails-Anwendung verankert. Normalerweise sind die folgenden Schritte erforderlich, um eine Entwicklungsumgebung für Rails-Anwendungen einzurichten:

- Installieren Sie Ruby on Rails auf dem Entwicklungsrechner. Eine Fiona-Installation wird nicht benötigt, da keine Inhalte erstellt werden.
- Legen Sie eine neue Rails-Anwendung an oder entnehmen Sie sie dem Versionierungssystem.
- Installieren Sie in dieser Anwendung den Rails Connector, falls dies noch nicht geschehen ist.
- Richten Sie auf dem Rechner eine Datenbank ein und spielen Sie in diese die Inhalte der Redaktionssystem-Datenbank ein oder konfigurieren Sie den direkten Zugriff auf die Datenbank des Redaktionssystems.
- Sofern noch nicht geschehen, konfigurieren Sie das Datenbankmodell ihrer Anwendung, damit diese auf die Daten in der Datenbank zugreifen kann.
- Erstellen oder ändern Sie in der Anwendung die Controllers und die Views, d.h. statten Sie die Anwendung mit Funktionalität (Geschäftslogik) und Layout (Präsentationslogik) aus.
- Spielen Sie die Änderungen in regelmäßigen Abständen zurück ins Versionsverwaltungssystem, damit andere Entwickler desselben Projekts Ihre Arbeit nutzen können.

- **Test der Webanwendung**

  Für professionelle Ruby-on-Rails Web-Anwendungen werden üblicherweise Testprogramme geschrieben, die sicherstellen, dass die Controller und die Views sich wie erwartet verhalten. In der Regel benötigen die Testprogramme eine eigene Datenbank, die sie mit Testdaten füllen. Die Tests gehören zum Entwicklungsprozess und laufen normalerweise auf dem jeweiligen Entwicklungssystem.

- **Live-System (Production)**

  Auf dem Livesystem werden die Ruby-on-Rails-Laufzeitumgebung, die Webanwendung mit dem Rails Connector für Fiona sowie die Datenbank(en) mit den redaktionellen Inhalten aus dem CMS benötigt. Das CMS selbst wird auf dem Live-Server nicht benötigt. Zur Aktualisierung der Inhalte auf dem Live-System werden Replikationsmechanismen verwendet, wenn Aktualität von großer Wichtigkeit ist. Es ist jedoch auch möglich, die Inhalte per Datenbank-Dump und -Restore aus dem Redaktionssystem ins Live-System zu übertragen. Für die Aktualisierung der Web-Anwendung selbst stehen Deployment-Programme wie "Capistrano" zur Verfügung. Möglicherweise benötigen Sie auf dem Live-System eine weitere Datenbank, um den von Besuchern erzeugten Content zu speichern.

Environments

Jede Rails-Anwendung hat voreingestellt drei so genannte Umgebungen (*engl.* "environments") die den Einsatz der Anwendung in unterschiedlichen Szenarios unterstützen.

Ein Environment ist ein benannter Konfigurationssatz. Beim Start des Rails-Servers kann einer der verfügbaren Namen angegeben werden. Neben der zu verwendenden Datenbank spezifiziert das Environment globale Parameter, mit denen bestimmte Aspekte des Verhaltens der Applikation gesteuert werden können, beispielsweise ob Arbeitsversionen oder freigegebene Versionen verwendet werden sollen. Die voreingestellt vorhandenen Environments sind `development` (Entwicklungssystem), `test` (Testsystem) und `production` (Live-System).

Mit dem Rails Connector kommt für den Einsatz der Rails-Anwendung als CMS-Vorschau üblicherweise ein weiteres Environment hinzu, `preview`. Über dieses Environment wird (zusätzlich zur zu verwendenden Datenbank) definiert, dass die Arbeitsversionen der CMS-Dateien maßgeblich sind

– im Unterschied zum Live-System, wo ausschließlich die freigebenen Versionen der CMS-Dateien verwendet werden.

Datenbank-Einsatz

Rails-Anwendungen können mit mehreren Datenbanken betrieben werden – im Falle des Rails Connectors ist dies eher die Regel denn die Ausnahme, weil auf der Live-Seite zum einen die redaktionellen Inhalte und zum anderen die von den Besuchern gespeicherten Daten wie Kommentare, Kundendaten und dergleichen verwendet werden. Da die redaktionellen Daten auf der Live-Seite nicht geändert werden, die Besucherdaten dagegen schon, setzt man zur klaren Trennung und aus Gründen der Stabilität und Datensicherheit unterschiedliche Datenbanken ein.

Sämtliche Datenbanken werden in der Datei `config/database.yml` der Rails-Anwendung konfiguriert. Typischerweise verwendet eine Rails-Anwendung die Datenbankverbindung, die den gleichen Namen hat wie das Environment. Der Infopark Rails Connector verwendet als Name `cms`, so dass Sie bestehende Verbindungen nicht zu ändern, sondern nur den neuen Abschnitt mit dem Namen `cms` hinzuzufügen brauchen.

1.5 Die Rails-Anwendung Playland

Zur Demonstration einiger Möglichkeiten, die die Entwicklung einer Rails-Anwendung im Zusammenhang mit Infopark CMS Fiona bietet, stellt Infopark die Rails-Applikation *Playland* zur Verfügung.

Wo nicht anders erwähnt, werden die genannten Funktionen vom Rails Connector selbst zur Verfügung gestellt, nicht von Software von Drittanbietern.

Um Playland zu installieren, gehen Sie bitte entsprechend der Anleitung in dieser Rails-Connector-Dokumentation vor.

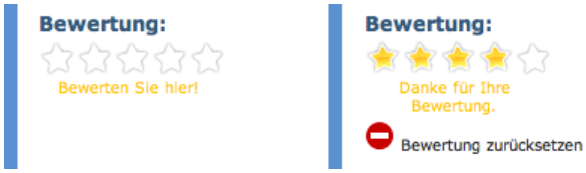
In Playland hat Infopark einige moderne Features exemplarisch implementiert: Zugriffsschutz (im Bereich *Investor Relations*), Bewertungen, Umfragen, Kommentierung und ein Wiki (beruht auf Fremdsoftware). Diese Features machen Playland lebendiger, weil sie auf Interaktion mit den Website-Besuchern ausgerichtet sind. Die Implementierung zeigt auch, wie von Besuchern erzeugter Content (etwa Kommentare) verarbeitet und dargestellt werden kann.

Website-Funktionen wie Bewertungen, Umfragen und Kommentierung zielen nicht ausschließlich darauf ab, Feedback der Besucher einzuholen und in die Entwicklung der Website oder der Produkte des Anbieters einfließen zu lassen. Es gibt für die Messung der Akzeptanz von Produkten und Websites durch die Nutzer weit präzisere Instrumente. Vielmehr haben solche Funktionen auch einen Community-Effekt. So sieht ein Besucher an den abgegebenen Bewertungen, den Stimmen bei Umfragen sowie den Kommentaren oder auch an Beiträgen in Foren (in Playland nicht enthalten), wie populär eine Website ist. Er sieht dadurch, dass sich um den Anbieter und Betreiber der Website eine Community gebildet hat. Die Existenz von “Community” macht den Anbieter interessanter, weil sie auf Akzeptanz schließen lässt. Erstbesuchern und wenig erfahrenen Besuchern verspricht eine große Community Hilfe in allen Angelegenheiten, die den Anbieter betreffen.

Werden solche Features richtig eingesetzt, können Sie neue oder zufällige Besucher dazu bewegen, sich länger mit dem Angebot zu befassen. Ganz allgemein bieten Community-Features in ihrer Gesamtheit den Besuchern eine Plattform beispielsweise für den Gedankenaustausch, für Kritik oder für die Informationsbeschaffung.

Bewertungen

Eine Bewertungsfunktion gibt dem Besucher einer Website die Möglichkeit, die Site insgesamt oder einzelne Seiten zu beurteilen. Hierzu klickt der Besucher auf ein Icon, um eine Punktzahl zu vergeben, die die Seite seiner Meinung nach “verdient”. Manchmal fokussiert der Betreiber mit Hilfe einer Frage die Bewertung auf ein Thema: “Wie gefällt Ihnen die Präsentation der Inhalte auf dieser Seite”, “Wie finden Sie diesen Artikel?”, “Wie informativ finden Sie diese Seite?”. Aus den abgegebenen Bewertungen wird der Durchschnitt berechnet und angezeigt. Die Möglichkeit, die Bewertung einer Seite zurückzusetzen, existiert nur, wenn man sich als Administrator angemeldet hat.



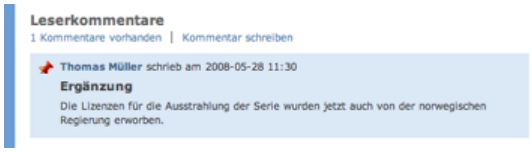
Je restriktiver der Betreiber die Bewertungsfunktion handhabt, desto glaubwürdiger wird das implizite “Bekenntnis” des Betreibers, auf das Feedback der Website-Besucher Wert zu legen. Aus diesem Grund sollten nur angemeldete Besucher eine Seite bewerten dürfen und mehrfache Bewertungen sollten die ursprüngliche Note nur korrigieren (anstatt als neue Bewertung einzufließen).

Die Bewertungen an sich sind in der Regel nicht differenziert genug, um aus ihnen allein Rückschlüsse darüber zu ziehen, wie die Qualität einer Webseite von den Besuchern eingeschätzt wird. Wenn die vergebenen Bewertungen jedoch mit den jeweiligen Besuchern verknüpft sind (dies ist mit dem OMC Connector möglich), können gezielte Maßnahmen ergriffen werden.

Bei Playland ist eine Bewertung ein von einem Benutzer erzeugter Inhalt, eine Bewertungszahl (Note), die in der Live-Datenbank zusammen mit einer Referenz auf das betreffende CMS-Objekt gespeichert wird. Das Bewertungselement kann für jedes CMS-Objekt dediziert ein- oder ausgeschaltet werden. Um die Bewertungen beispielsweise im OMC erfassen zu können, ist zusätzlicher Code erforderlich.

Kommentare

Die Kommentarfunktion gibt den Besuchern einer Webseite die Möglichkeit, eine Seite mit Anmerkungen, Kritik, Verbesserungsvorschlägen, Fragen usw. zu versehen. Die Kommentare sind normalerweise für alle Besucher sichtbar und fördern den Gedankenaustausch über die angebotenen Inhalte. Die abgegebenen Kommentare sind üblicherweise unterhalb eines Artikels aufgeführt:



Um die Wahrscheinlichkeit des inhaltlichen Missbrauchs dieser Funktion zu minimieren, sollte ein entsprechender Schutz implementiert werden, beispielsweise die explizite Freischaltung von Kommentaren durch einen Moderator. Eine Form der Benachrichtigung des Moderators bei eingehenden Kommentaren (etwa per E-Mail) ist zweckmäßig. Auch sollten Maßnahmen gegen Spam ergriffen werden. Diese Zusatzfunktionen sind in Playland nicht implementiert.

Der Websitebetreiber sollte regelmäßig die abgegebenen Kommentare auswerten, um nützliche Informationen zu extrahieren und gegebenenfalls Aktivitäten daraus abzuleiten, aber auch, um eskalierende Diskussionen zu erkennen und darauf zu reagieren (Moderation).

Die Anzahl der Kommentare, die ein Besucher abgeben kann, ist nicht beschränkt. Die Kommentare werden in der Reihenfolge ihres Eingangs in der Datenbank gespeichert, zusammen mit einer Referenz auf das betreffende CMS-Objekt und den erforderlichen Benutzerdaten.

Personalisierung mit OMC-Integration

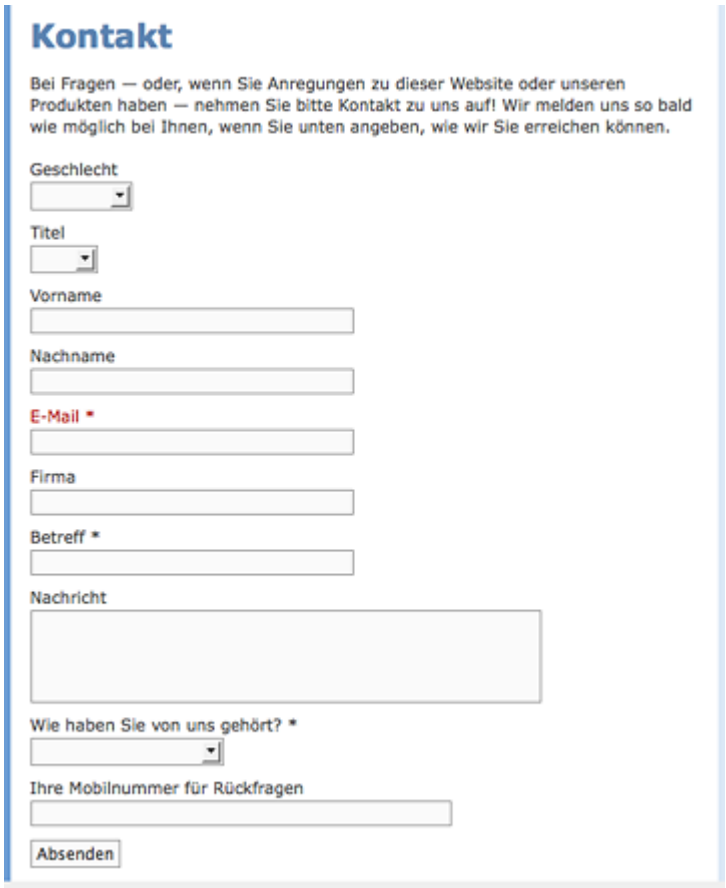
Diese Funktionalität setzt neben dem Rails Connector das Gem `infopark_omc_connector` voraus.

Immer mehr Websites bieten Personalisierung an, um beispielsweise die Kundenbindung zu verstärken. Für den Betreiber geht dies mit einigen Vorteilen einher: Premium-Kunden kann Zugriff auf weitere Inhalte gewährt werden. Daten werden durch die Benutzer selbst aktuell gehalten. Mit einem entsprechenden Backend, in dem alle Informationen zusammenfließen, werden die Daten zusätzlich konsistent gehalten.

Personalisierung erfordert eine Infrastruktur, in der Benutzer sich registrieren, an- und abmelden sowie Passwörter ändern oder zurücksetzen können.

Diese Infrastruktur wird vom Rails Connector bereitgestellt. Als Backend wird dabei das Infopark Online Marketing Cockpit (OMC) verwendet. Playland nutzt die vom Rails Connector bereitgestellten Funktionen und Views, um die Rails-Anwendung mit dem OMC zu verbinden. Benutzerrollen, die im OMC festgelegt werden, wirken sich beispielsweise auf die Sichtbarkeit von (Premium-)Inhalten und auf die Möglichkeit, Kommentare zu löschen, aus.

Basierend auf Aktivitätstypen im OMC können mit dem Rails Connector Formulare erzeugt werden. Für jedes Feld des gewünschten Aktivitätstyps (oder für eine Auswahl dieser Felder) enthält das Formular ein entsprechendes Eingabefeld:

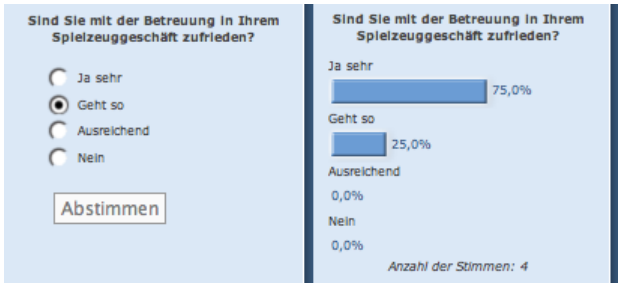


Schickt ein angemeldeter Besucher ein solches Formular ab, kann mit Hilfe des Rails Connectors im OMC eine Aktivität mit den Feldinhalten aus dem Formular angelegt werden. So lassen sich zahlreiche Vorgänge (etwa Teilnahme an Veranstaltungen, Registrierung, Support-Anfragen) über die Website abwickeln und die dazugehörenden Daten ins OMC einspeisen. Dort können sie manuell oder automatisiert weiterverarbeitet und auch beispielsweise für das Veranstaltungsmanagement, Mailings etc. genutzt werden.

Umfragen

Diese Funktionalität wird nicht vom Rails Connector bereitgestellt, sondern ist direkt in Playland implementiert.

Mit einer Umfrage kann sich ein Website-Betreiber gezielt Informationen von den Besuchern seiner Site einholen. Eine Umfrage besteht gewöhnlich aus einer Frage und mehreren Antwortmöglichkeiten, von denen der Besucher eine auswählt und per Klick abgibt.



Wurden Stimmen abgegeben, so zeigt ein Diagramm die Anteile der je Antwort abgegebenen Stimmen im Verhältnis zur Gesamtanzahl an. Der Administrator kann auf einfache Weise Umfragen anlegen, bearbeiten oder löschen.

Bei klaren Fragen kann eine Umfrage durchaus nützliche Ergebnisse hervorbringen, allerdings besteht auch hier die Gefahr, die abgegebenen Stimmen für repräsentativ zu halten oder anderweitig falsch zu interpretieren.

Technisch werden die abgegebenen Stimmen wie bei Bewertungen gehandhabt, d.h. die Auswahl eines Besuchers wird in der Live-Datenbank abgelegt und kann später ausgewertet werden.

Wikis

Diese Funktionalität wird nicht vom Rails Connector bereitgestellt, sondern ist direkt in Playland implementiert.

Ein Wiki ist eine Plattform, auf der einfach strukturierte Webdokumente von mehreren Autoren erstellt und gepflegt werden können. Der Reiz eines Wikis besteht darin, dass es leicht zu benutzen ist und für einen beliebig großen Benutzerkreis geöffnet werden kann, wodurch eine große Menge an Informationen in kurzer Zeit zusammengetragen werden kann. Das beste Beispiel hierfür ist Wikipedia. Diese Freizügigkeit bringt es mit sich, dass die Qualität der Inhalte schwankt und das Gesamtbild inkohärent werden kann. Als Medium für die schnelle Veröffentlichung von Inhalten für geschlossene Benutzerkreise (beispielsweise Intranets) oder auch für interne Dokumentationszwecke sind Wikis jedoch bestens geeignet.

Bei Playland können Wikis durch Platzhalter-Objekte in die Struktur der CMS-Inhalte eingefügt werden. Dadurch kann ein Wiki wie andere CMS-Inhalte beispielsweise in Navigationen angezeigt werden, es kann verlinkt werden, usw. In der Rails-Anwendung *Playland* wird die Wiki-Funktionalität durch ein Plugin bereitgestellt.

Weitere Website-Funktionen

Suche

Eine durchdachte Navigation kann durch eine integrierte Suche optimal ergänzt werden, so dass Besucher schnell zu den Informationen gelangen, die für sie relevant sind. Erfordert den Infopark Search Server.

PDF-Generator

Mit Hilfe des PDF Generators können Web-Inhalte dynamisch in das PDF-Format umgewandelt werden, beispielsweise um sie offline verfügbar zu machen.

Time Machine

Die Time Machine ermöglicht es – ausgehend von den Gültigkeitszeiträumen der Inhalte – zu sehen, wie die Website an einem beliebigen Datum in der Zukunft aussehen wird.

RSS-Feeds und Podcasts

RSS-Feeds und Podcasts geben Website-Besuchern die Möglichkeit, sich über Neuigkeiten auf der Site automatisch auf dem Laufenden zu halten.

Suchmaschinenoptimierung (SEO)
Webseiten können mit Metadaten versehen werden, die von Suchmaschinen ausgewertet werden. Dies verbessert das Suchmaschinen-Rating der Website und macht sie leichter auffindbar. Ferner unterstützt ein automatisch generiertes Sitemap-XML-Dokument die Indizierung Ihrer Website durch Suchmaschinen.

Tracking
Google Analytics is ein mächtiges Werkzeug, mit dem die aus Websitebesuchen entstehenden Daten gesammelt und analysiert werden können. Das mit dem Infopark Online Marketing Cockpit angebotene Tracking ermöglicht es, auch bekannte Benutzer zu tracken, wodurch die Möglichkeiten der Optimierung einer Website und die Personalisierung von Inhalten nochmals erweitert werden.

1.6 Layout einer Rails-Anwendung

Rails-Anwendungen basieren nicht auf den Layout-Dateien (Templates), die im CMS hinterlegt sind, sondern auf Views, die ein wesentlicher Bestandteil der Rails-Anwendung selbst sind. Views sind HTML-Dateien, die Programm-Code enthalten, mit dem die Inhalte dynamisch in die anzuzeigenden Webseiten aufgenommen werden.

In einem Rails-Projekt erstellt der Designer das Layout der Webseiten. Das Layout basiert auf einer Rails-Layout-Datei, die das HTML-Gerüst der auszuliefernden Webseiten enthält. In diesem HTML-Gerüst sind Platzhalter enthalten, die später vom Rails-Framework durch Inhalte ersetzt werden. Welche Inhalte eingesetzt werden, hängt von dem verwendeten View ab, der wiederum über die aufgerufene URL bestimmt wird.

Die Platzhalter im Layout umfassen einen voreingestellten Bereich (meist der Hauptinhalt) sowie optionale weitere benannte Bereiche. Für die Inhalte in sämtlichen Bereichen sorgt der jeweilige View, der hierfür ebenfalls einen Hauptabschnitt und weitere benannte Abschnitte enthält. Befindet sich im Layout ein benannter Bereich, für den es im View keinen entsprechenden Abschnitt gibt, so bleibt dieser Bereich in der erzeugten Webseite leer.

Um das Design zu erstellen oder zu ändern, passt der Designer das Layout und die Views so an, dass die erzeugten Webseiten der Spezifikation entsprechen. Typischerweise werden im Layout Style-Sheets referenziert, deren Klassen dort und in den Views verwendet werden.

Der Infopark Rails Connector unterstützt ab Version 6.7.1 zwei Template-Sprachen für Views und Layouts, ERb (Embedded Ruby) und Liquid.

Die Templatesprache ERb gehört zur Standardbibliothek von Ruby. Bei der Verwendung von ERb wird Ruby-Code direkt in die Views und in das Layout eingebettet. Mit Hilfe des Infopark Rails Connectors können CMS-Inhalte innerhalb von ERb-Templates ausgegeben werden. ERb-Templates haben die Dateiendung `.html.erb`.

Liquid ist eine in Ruby implementierte Templatesprache mit Fokus auf Sicherheit und Robustheit. Liquid-Templates haben eine einfache Syntax und sind daher im Vergleich zu ERb-Templates leichter zu schreiben und weniger fehleranfällig. Der Infopark Rails Connector erweitert Liquid um spezielle Sprachkonstrukte, mit denen CMS-Inhalte ausgegeben werden können. Liquid-Templates haben die Dateiendung `.html.liquid`.

Innerhalb einer Rails-Anwendung kann ein Teil der Views und des Layouts in ERb und der andere Teil in Liquid formuliert sein.

1.7 Verteilung einer Rails-Anwendung (Deployment)

Bei der Entwicklung und Verteilung (Deployment) von Rails-Projekten empfiehlt sich der Einsatz einer Versionierungssoftware wie Subversion oder Git sowie des Rails-Deployment-Programms Capistrano.

Die Versionierungssoftware erleichtert insbesondere bei größeren Projekten die Pflege der Rails-Anwendung. Da sämtliche Bestandteile der Anwendung, die dem Entwicklungsprozess unterliegen, an einem zentralen Ort – in der Datenbank der Versionierungsanwendung – gespeichert sind, ist zu jeder Zeit eindeutig definiert, welche Version die aktuelle ist und auf welche Weise man an diese aktuelle Version gelangt. Ein Entwicklungsprozess besteht für den einzelnen Designer oder Programmierer aus einer Abfolge identischer Zyklen: Auschecken, bearbeiten, testen, einchecken.

Capistrano verteilt Ihre Web-Anwendung auf die Zielsysteme wie den Staging- und den oder die Live-Server. Es bedient sich hierzu der im Repository der Versionierungsanwendung abgelegten aktuellen Version der Webanwendung sowie einiger Unix-Standard-Programme wie ssh.

Die auszuführenden Aktionen entnimmt Capistrano einer Konfigurationsdatei. Für ein Rails-Projekt lässt sich die initiale Konfiguration mit Capistrano selbst erzeugen. Sie ist für die klassischen Deployment-Fälle ausgelegt und kann vom Administrator leicht geändert oder erweitert werden.

1.8 CMS-Vorlagen als erweiterbare Seitentypen

Mit dem Infopark Rails Connector kann man aus Rails-Anwendungen heraus auf die CMS-Dateien zugreifen. Hierfür stellt der Rails Connector die Variable `@obj` zur Verfügung, die die auszuliefernde Datei repräsentiert. So kann man beispielsweise mit `@obj.abstract` den Wert des Feldes `abstract` ermitteln.

`@obj` gehört zur Klasse `Obj`. Bis Version 6.7.2 bildet der Rails Connector auf der Grundlage des Vorlagennamens automatisch Subklassen von `Obj`, so dass beispielsweise eine CMS-Datei mit der Vorlage `NewsArticle` automatisch eine Instanz der Klasse `NewsArticle` ist.

Ab Version 6.7.3 werden Subklassen von `Obj` nicht mehr automatisch erzeugt, sondern alle CMS-Dateien gehören immer zur Klasse `Obj`. Enthält die Applikation jedoch eine von `Obj` abgeleitete Klasse, deren Name dem Namen einer Vorlage entspricht, so sind Dateien mit dieser Vorlage automatisch Instanzen dieser Klasse.

Vorlagen-Klassen lassen sich nutzen, um CMS-Dateien, die eine bestimmte Vorlage haben, um die gewünschten Eigenschaften und Methoden zu erweitern. Sollen etwa alle CMS-Dateien mit der Vorlage `Product` bewertbar sein, so kann man hierfür das Modell für `Product` folgendermaßen definieren:

```
# This Model describes the behavior of all CMS objects of the Product class.
class Product < Obj

  has_many :ratings, :dependent => :delete_all, :foreign_key => :obj_id

  def rate(score)
    rating = ratings.find_by_score(score) || ratings.build(:score => score)
    rating.count += 1
    rating.save
  end

  def count_for_score(score)
    rating = ratings.find_by_score(score)
    rating ? rating.count : 0
  end

  def rated?
```

```
    !ratings.empty?
  end

  def average_rating
    raise TypeError unless rated?
    sum, count = ratings.inject([0, 0]) do |(sum, count), rating|
      [sum + rating.score * rating.count, count + rating.count]
    end
    sum.to_f / count.to_f
  end

  def average_rating_in_percent
    if rated?
      (100 * average_rating / Rating::MAXIMUM).to_i
    else
      0
    end
  end

  def reset_rating
    ratings.clear
  end
end
```

Ruby verlangt, dass Klassennamen der CamelCase-Notation folgen, d.h. mit einem Großbuchstaben beginnen. Wir empfehlen dringend, Vorlagen, deren Name nicht dieser Konvention entspricht, umzubenennen. Welche Vorlagen betroffen sind, kann mit dem Rake-Task `check:obj_classes` ermittelt werden. Danach können die Vorlagen mit dem Tcl-Kommando renameObjClass umbenannt werden.

1.9 Vorlagenspezifische Controller

Der Rails Connector ermöglicht es (ab Version 6.7.3), Controller zu definieren, die nur für bestimmte Vorlagen zuständig sind. Einen neuen Controller einzubinden, erfordert keinerlei Konfiguration, abgesehen davon, dass die Controller-Klasse angelegt werden muss.

Mögliche Anwendungsfälle sind:

- Sehr viele Views, die man nach Vorlage gruppieren möchte
- Umschaltung zwischen mehreren Ansichten für eine CMS-Datei
- Auswertung von Session-/Cookie-/Parameter-Daten für bestimmte Vorlagen
- Automatisches Laden des jeweils passenden Helper-Moduls (ohne `helper :all`)
- Handling von `POST`-Requests, beispielsweise für die Realisierung von Formularen über CMS-Dateien
- Ausgabe von CMS-Daten in mehreren Formaten, wie XML, HTML, JSON.

Der Rails Connector sucht für jede CMS-Datei, die angezeigt werden soll, nach einem Controller, dessen Name zur Vorlage der Datei passt. CMS-Dateien der Vorlage `Publication` würden, sobald ein Controller namens `PublicationController` existiert, der von `RailsConnector::DefaultCmsController` abgeleitet ist, von diesem ausgeliefert werden. Existiert er nicht, wird stattdessen `CmsController` verwendet.

Die Standard-Aktion zur Auslieferung einer CMS-Datei bleibt weiterhin `index`. Zusätzliche Aktionen können über benutzerdefinierte Routen angesprochen werden. Um auf einer Seite auf eine CMS-Datei und eine bestimmte Aktion des entsprechenden Controllers zu verlinken, muss eine dieser benutzerdefinierten Routen zur Generierung der Link-URL verwendet werden.

Der Rails Connector stellt eine spezielle Testmethode zur Verfügung, um vorlagenspezifische Controller zu testen. Mit der Methode `request.for_cms_object` kann im Test definiert werden, welches Cms-Objekt geladen werden soll. Hier ein Beispiel für das Testframework `rspec`:

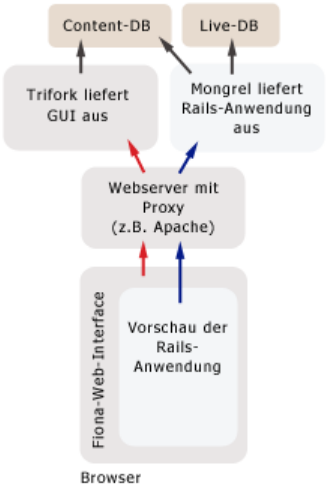
```
describe CmsController, "request for HTML document" do
  before do
    controller.stub!(:ensure_object_is_permitted).and_return(true)
    controller.stub!(:ensure_object_is_active).and_return(true)
    controller.stub!(:set_google_expire_header).and_return(true)
    @obj = mock_model(Obj, :mime_type => 'text/html', :permalink => 'dummy')
    request.for_cms_object(@obj)
  end

  it "should render the view" do
    get 'index'
    response.should be_success
  end
end
```

1.10 Die Rails-Applikation als Vorschau-Server

In der Standard-Installation von Infopark CMS Fiona wird die Vorschau vom Infopark Portal Manager ausgeliefert. Ab Version 6.6 des CMS können Sie alternativ eine Rails-Anwendung als Vorschau einbinden, so dass die erstellten Inhalte sofort im Layout der Rails-Anwendung betrachtet werden können.

Bei einem Webauftritt, der als Rails-Anwendung realisiert wurde, werden die Webseiten dynamisch ausgeliefert. Der Rails-Server erzeugt die angeforderten Seiten on-the-fly und verwendet dabei die aktuell in seiner Datenbank enthaltenen Inhalte. Bei entsprechender Synchronisation der Datenbanken des Redaktions- und des Live-Systems ist Ihr Webauftritt hochaktuell. Mit optionalen Caching-Mechanismen kann die Systemlast verringert werden.

Die Freigabe eines Dokuments im Redaktionssystem hat dessen sofortige Veröffentlichung zur Folge. Die Exportmöglichkeiten des CMS haben in diesem Falle keine Funktion. Dementsprechend werden Layoutdateien im CMS nicht berücksichtigt.



Bei einem Setup mit einer Rails-Anwendung für die Vorschau werden zwei Server verwendet, je einer für die GUI-Seiten (Trifork) und die Vorschau-Seiten (Mongrel). Die Server kommunizieren

miteinander. Sie müssen nach außen als ein Server erscheinen, weil die meisten Browser ansonsten die Kommunikation wegen XSS-Gefahr nicht zulassen.

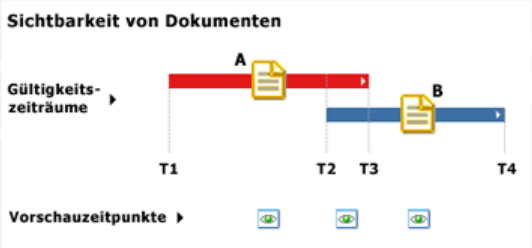
Aus diesem Grunde wird ein Apache-HTTP-Server als Reverse-Proxy vor die oben genannten Server gestellt.

Die Anleitung Integration der Rails-Vorschau ins GUI zeigt, wie die Rails-Vorschau aktiviert wird.

1.11 Vorschau in der Zukunft ("Time Machine")

Für Redakteure bietet der Infopark Rails Connector mit der "Time Machine" eine Möglichkeit, die Inhalte zu einem in der Zukunft liegenden Zeitpunkt zu betrachten. Dies ist vor allem dann interessant, wenn viele Inhalte mit zeitlich unterschiedlich begrenzter Gültigkeit zu einem bestimmten Zeitpunkt dargestellt werden sollen.

Ist beispielsweise ein Document "A" im Zeitraum von T1 bis T3 gültig und ein anderes Dokument, "B" von T2 bis T4 (die Zeiträume überschneiden sich), so ist es mit der "Time Machine" möglich, das Erscheinungsbild der Seite zu den jeweils interessanten Zeitpunkten zu betrachten: die Zeit zwischen T1 und T2 (nur "A" ist sichtbar), die Zeit zwischen T2 und T3 (beide sind sichtbar) sowie von T3 bis T4 (nur "B" ist sichtbar):



Sie können den Vorschauzeitpunkt mit Hilfe eines Links in der Vorschauseite festlegen, sofern dieser in die Vorschauseite integriert wurde. Der Link öffnet ein Fenster, das einen Schieberegler sowie einen Kalender enthält:

Mit beiden Bedienelementen kann der gewünschte Vorschauzeitpunkt ausgewählt werden. Lässt man den Schieberegler los oder klickt man im Kalender auf den gewünschten Tag, so wird der ausgewählte Vorschauzeitpunkt gesetzt und ein Refresh auslöst, sodass danach die aktuelle Vorschauseite sofort im Kontext des gewählten Zeitpunkts dargestellt wird.

1.12 Was sind Permalinks?

Permalinks sind persistente, d.h. sich nicht ändernde Bezeichnungen für CMS-Dateien. Auf einer mit dem Rails Connector ausgelieferten Website kann unter Angabe einer solchen Bezeichnung in der URL die dazugehörende Datei abgerufen werden, unabhängig von ihrem tatsächlichen Speicherort im CMS. Beispiele:

```
http://www.meinserver.de/agb
http://www.myserver.com/news/2011-07-27/current-release-is-available-for-download
```

Permalinks sind Dateifelder, gehören also nicht zum Content. Im Redaktionssystem kann nach entsprechender Konfiguration für jede Datei ein Permalink vergeben werden.

1.13 PDF-Dateien generieren

Der Infopark Rails Connector wird mit einer Komponente geliefert, dem PDF-Generator, mit der aus einer Rails-Anwendung heraus PDF-Dateien generiert werden können. Hierfür wird ein Webservice-Interface verwendet.

Der PDF-Generator nutzt den Apache FOP (Formatting Objects Processor), eine Java-Applikation, die eine XML-Datei in eine PDF-Datei übersetzt. Die Regeln zur Erzeugung des Layouts entnimmt der FOP einem XSL-Stylesheet. Ein Beispiel-Stylesheet wird mitgeliefert.

Der PDF-Generator kennt zwei Modi, den externen und den internen Modus.

Der externe Modus

Beim externen Modus kann für die XML- und die XSL-Datei jeweils eine beliebige externe Quelle angegeben werden. Dadurch lässt sich eine andere Anwendung (etwa eine PHP-Applikation) als Quelle verwenden.

Der interne Modus

Der interne Modus hingegen erlaubt es dem Programmierer, PDF-Dateien aus den CMS-Dateien zu generieren. Der hierfür verwendete Mechanismus ist flexibel und erweiterbar.

Der PDF-Generator ist initial deaktiviert und muss zunächst eingerichtet werden. Anschließend kann er ohne weiteren Aufwand sofort verwendet werden, da alle benötigten Dateien mitgeliefert werden.

1.14 Dokumente mit dem Infopark Search Server suchen

Der Infopark Rails Connector installiert einen Search-Controller, mit dem Suchanfragen an den Infopark Search Server gestellt werden können. Über einen zu diesem Controller gehörenden View werden die Suchergebnisseiten gestaltet und die Treffer verlinkt.

Damit Suchanfragen gestellt werden können, müssen der Infopark Search Server und die zu durchsuchenden Indizes für die Rails-Anwendung verfügbar gemacht werden. Hierfür gibt es im Wesentlichen die beiden folgenden Ansätze:

- Der auf dem Redaktionssystem laufende Search Server wird auch für die Live-Suche verwendet. Diese Lösung ist nur zu empfehlen, wenn die Verfügbarkeit des Redaktionssystems durch die zusätzlich entstehende Last nicht eingeschränkt wird.

Auch wenn die Inhalte-Datenbank auf dem Live-System (beispielsweise durch kurze Replikationszyklen) möglichst aktuell gehalten wird, können Diskrepanzen zwischen den Indizes auf dem Redaktionssystem und den tatsächlichen Inhalten auf dem Live-System entstehen (geänderte Inhalte sind schneller im Index vorhanden als in der Live-Datenbank). Wenn diesbezüglich eine absolute Übereinstimmung gefordert ist, kann diese Lösung nicht eingesetzt werden.

- Auf dem Live-Server wird ein dedizierter Search Server betrieben, bei hoher Last durch Suchanfagen gegebenenfalls auf einem eigenen Rechner.

  Bei dieser Lösung müssen die Indizes des Search Servers in den gleichen Intervallen auf den Live-Server repliziert werden wie die Inhalte der Datenbank des Redaktionssystems.

1.15 Integrationstests für ein Rails-Connector-Projekt schreiben

Integrationstests sind ein wichtiges Werkzeug während der Entwicklung einer Rails-Applikation. Mit Hilfe von Integrationstests kann sichergestellt werden, dass bei der Entwicklung neuer Funktionalität die bestehende Funktionalität nicht versehentlich beschädigt wird.

Ruby on Rails bietet bereits die Möglichkeit, Unit-Tests, funktionale Tests und Integrationstests für die Rails-Applikation zu schreiben. Die Integrationstests für eine Rails-Anwendung liegen normalerweise im Verzeichnis `test/integration` der Rails-Anwendung.

Bis Version 6.6.1 können Integrationstests erst nach manueller Einrichtung des Environments und der dazu gehörenden Datenbankverbindung durchgeführt werden.

Ab Version 6.7.0 werden Integrationstests im Environment `test` durchgeführt. Da in alle Environments die gleiche Datenbank eingebunden ist – die Datenbank mit den Produktivdaten –, ist es möglich, im Rahmen eines Integrationstests auf die tatsächlichen CMS-Inhalte zuzugreifen. So kann im Integrationstest geprüft werden, ob die Rails-Applikation den CMS-Content wie geplant verarbeitet.

Um zu verhindern, dass durch einen Programmierfehler in einem Integrationstest die CMS-Inhalte geändert oder korrumpiert werden, lassen Sie den Rails Connector sich nur über einen Datenbankbenutzer ohne Schreibrechte mit der Datenbank verbinden.

1.16 Empfohlene Literatur zu Ruby on Rails

Zum Thema Ruby on Rails, MySQL und verwandten Themen empfehlen wir Ihnen die folgende Literatur:

- Programming Ruby
  (ISBN-13: 978-0974514055
- Agile Web Development with Rails
  (ISBN-13: 978-0977616633)
- Rails Recipes
  (ISBN-13: 978-0977616602)
- Rails Cookbook
  (ISBN-13: 978-0596527310)
- Ruby Cookbook
  (ISBN-13: 978-0596523695)
- Ajax on Rails
  (ISBN-13: 978-0596527440)

- High Performance MySQL
  (ISBN-13: 978-0596527082)
- Building Scalable Web-Sites
  (ISBN-13: 978-0596102357)
- Ruby Pocket Reference
  (ISBN-13: 978-0596514815)
- Capistrano and the Rails Application Lifecycle (eBook / O'Reilly Shortcut)
  (ISBN-13: 978-0-596-52962-8)
- Mongrel (eBook / O'Reilly Shortcut)
  ISBN-13: 978-0-59-652854-6

2

2 Anleitungen

2.1 Konventionen

Für die in den folgenden Anleitungen gezeigten Ein-/Ausgaben der Kommandozeile gelten die folgenden Konventionen:

- `#` - Promptzeichen des Administrators. Führen Sie diesen Befehl als Benutzer mit Administrationsrechten aus.
- `$` - Promptzeichen des normalen Benutzers. Führen Sie diesen Befehl als normaler Benutzer aus.
- Benutzereingaben sind durch blaue Schriftfarbe gekennzeichnet:

```
# gem install mysql
```

- Befehlsausgaben sind durch schwarze Schrift gekennzeichnet:

```
Select which gem to install for your platform (i686-linux)
```

- Kommentare sind durch rote Schrift gekennzeichnet und in runde Klammern eingeschlossen:

```
(Bitte geben Sie die Zeilenumbrüche mit ein.)
```

- In Pfaden wird stets der Unix-Pfadtrenner `/` (slash) verwendet. Ersetzen Sie diesen unter Windows durch `\` (backslash).

2.2 Installationsvoraussetzungen

2.2.1 Hardware

Für die Entwicklung, den CMS-Server, den Live-Server sowie gegebenenfalls den Staging-Server empfehlen wir den Einsatz dedizierter Rechner. Diese sollten die gleiche Hardware-Plattform sowie einheitliche Betriebssoftware (wie Betriebssystem und Datenbank) haben. Dadurch lassen sich Probleme mit der verwendeten Hard- und Software rechtzeitig und nicht erst im Live-Betrieb feststellen.

Für die Entwicklung, das Testen und Staging ist ein handelsüblicher PC ausreichend.

Für das Live-System wird in der Regel leistungsfähigere Hardware benötigt als für die anderen genannten Systeme. Rails-Anwendungen lassen sich leicht skalieren, indem schnellere Hardware zum System hinzugefügt wird.

2.2.2 Serverseitige Software

Um den Infopark Rails Connector in der Version 6.7.3 einsetzen zu können, benötigen Sie die unten aufgeführte Software. Infopark empfiehlt, Gems nach Möglichkeit stets als derselbe Benutzer zu installieren, da sie andernfalls an unterschiedlichen Stellen im Dateisystem abgelegt werden, wodurch Bundler sie möglicherweise nicht mehr findet.

1. Für den Produktivbetrieb ist ein Standard-Linux-System (empfohlen: 64 Bit) erforderlich. Als System für die Entwicklung kann auch Mac OS X in der Version 10.6 (“Snow Leopard”) oder 10.7 (“Lion”) eingesetzt werden. Rails-Connector-Versionen bis einschließlich 6.7.1 sind jedoch nicht mit Mac OS X 10.6 oder höher kompatibel.
2. Ruby 1.8.7 Patchlevel 174 oder höher mit OpenSSL-Unterstützung und der dynamisch ladbaren Ruby-Bibliothek
3. RubyGems 1.3.7. Während der Installation des Infopark Rails Connectors wird RubyGems weitere benötigte Software von Drittanbietern installieren. Hierfür ist eine Internetverbindung erforderlich.
4. Rails 3.0.5 sowie Bundler.
5. Infopark CMS Fiona (Version 6.7.2 oder höher) mit einer MySQL-Datenbank. Bei der Datenbank muss `storeBlobsInDatabase` eingeschaltet sein. Der Content Manager (CM) muss railsifiziert worden sein.
6. Ein Apache-HTTP-Server ab Version 2.2. Es ist auch möglich, andere Webserver einzusetzen, die als Reverse-Proxy und Load-Balancer funktionieren. Infopark unterstützt aktuell nur den Betrieb mit dem Apache.

   Der eingesetzte Webserver sollte den XSendFile-Header verarbeiten. Dies ist keine absolute Notwendigkeit, jedoch führt der Verzicht auf den Einsatz von XSendFile zu Geschwindigkeitseinbußen.

   Um beim Apache-HTTP-Server die Verarbeitung von XSendFile zu ermöglichen, installieren Sie bitte das Modul `mod_xsendfile`.

   Falls erforderlich, aktivieren Sie in Ihrer Rails-Anwendung den Gebrauch von XSendFile.
7. Wenn der Rails Connector im Phusion Passenger betrieben wird, so ist `mod_passenger` mindestens in der Version 2.2.9 erforderlich.

2.2.3 Clientseitige Software

Die Bearbeitungselemente werden derzeit im Internet Explorer 6 nicht unterstützt. Die Marker-Menüs dagegen funktionieren auch in dieser Version des Internet Explorers.

2.2.4 Bekannte Einschränkungen

Bei Rails und dem Rails Connector für Infopark CMS Fiona sind derzeit die folgenden Einschränkungen bekannt:

1. Spiegeldateien werden nicht unterstützt.
2. Wenn Rails und der Infopark Rails Connector auf einem 32-Bit-Rechner eingesetzt werden, führen Datumswerte ab dem 19. Januar 2038 zu Fehlern bei der Auslieferung der betreffenden Inhalte. Bei 64-Bit-Systemen tritt dieser Fehler nicht auf.

2.3 Die Rails-Beispielanwendung Playland installieren

Diese Anleitung setzt eine lauffähige Installation von Infopark CMS Fiona voraus. Bitte beachten Sie ferner die Installationsvoraussetzungen. Wenn Sie die mitgelieferten Tests der Anwendung ausführen möchten, benötigen Sie zusätzlich die Bibliotheken `libxml2-devel` und `libxslt-devel`.

1. Die Beispielanwendung und von ihr abhängende Komponenten installieren

   - Installieren Sie das Infopark Rails Connector Gem, das MySQL-Blob-Streaming Gem und das OMC Connector Gem folgendermaßen und in der angegebenen Reihenfolge:

     ```
     $ gem install pfad/zum/mysql_blob_streaming-x.y.z.gem
     $ gem install pfad/zum/infopark_omc_connector-x.y.z.gem
     $ gem install pfad/zum/infopark_rails_connector-x.y.z.gem
     ```

     Weitere erforderliche Software wird automatisch installiert.
   - Laden Sie die Beispielanwendung *Playland* im Downloadbereich unserer Website herunter. Diese ist ab Fiona 6.7.3 im Rails-Connector-Package enthalten
   - Entpacken Sie das Package in ein Verzeichnis Ihrer Wahl. Alle folgenden Pfadangaben beziehen sich auf dieses Verzeichnis.
   - Das Package enthält Informationen über zusätzlich zu installierende Software von Drittanbietern. Wechseln Sie ins *Playland*-Verzeichnis und installieren Sie die zusätzliche Software:

     ```
     $ cd playland-x.y.z/

     # Installation inklusive der oben genannten Bibliotheken für die Tests
     $ bundle install

     # Installation ohne die oben genannten Bibliotheken für die Tests
     $ bundle install --without=test
     ```

   - Kopieren Sie Ihre Lizenzdatei `license.xml` in das Verzeichnis `config`.
2. OMC-Integration vorbereiten

   Führen Sie die folgenden Schritte durch, wenn Sie die Playland-Rails-Anwendung mit dem Infopark Online Marketing Cockpit verbinden möchten. Bitte beachten Sie, dass die Anmelde- und Abmeldefunktionen von Playland ohne OMC nicht genutzt werden können.

   - Registrieren Sie Ihre Website bei reCaptcha. Sie erhalten ein Schlüsselpaar, um den reCaptcha-Webservice zu nutzen. Falls Sie bereits Nutzer von reCaptcha sind, kann dieser Schritt übersprungen werden.
   - Tragen Sie das reCaptcha-Schlüsselpaar in die Datei `config/initializers/rails_connector.rb` ein:

```
RCC_PUB="1331"  # öffentlicher Schlüssel
RCC_PRIV="3113" # privater Schlüssel
```

- Passen Sie die Datei `rails_connector.rb` an Ihre OMC-Konfiguration an:

```
OmcConnector.configure do |config|
  port = ENV['OMC_PORT'] || 4000          # Auf welchem Port läuft das OMC?
  config.url = "http://localhost:#{port}" # Auf welchem Host?
  config.login = "webservice"     # seit 6.7.3
  config.api_key = 'geheim'       # vor 6.7.3: config.password
  config.contact_roles_callback = lambda {|c| ['admins']}  # erlaubt Löschen von
 Kommentaren
end
```

3. Datenbank und Suchfunktion von Infopark CMS Fiona vorbereiten

   Verwenden Sie nach Möglichkeit eine neue CMS-Instanz namens `playland`.

   Damit der Rails Connector auf die Datenbank des Content Managers zugreifen kann, muss die Datenbank des CM auf MySQL umgestellt werden.

   Danach kann die Datenbank mit dem Inhalt befüllt und für den Infopark Rails Connector optimiert werden. Wechseln Sie dazu in das Verzeichnis der entsprechenden Instanz und führen Sie folgende Befehle aus:

```
$ ./bin/CM -restore ../../share/demoContentDump
$ ./bin/CM -railsify
```

   Um die Suchfunktionen von Playland-Rails nutzen zu können, müssen die Collections der SES wie folgt vorbereitet werden:

```
$ ./bin/CM -single < ../../share/configureInstanceForDemoContent.tcl
$ echo "indexAllObjects" | ./bin/CM -single
```

4. Zusätzliche Datenbanken für die Playland-Rails-Anwendung anlegen

   Legen Sie mit den folgenden Kommandos zusätzliche Datenbanken für Live-Daten wie Ratings etc. an. Sie benötigen je Environment (`development` und `test`) eine solche Datenbank.

```
$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 18
Server version: 5.0.41-community MySQL Community Edition (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> create database playland_development;
Query OK, 1 row affected (0.00 sec)

mysql> grant all on playland_development.* to fiona@localhost identified by 'fiona';
Query OK, 1 row affected (0.00 sec)
```

```
mysql> create database playland_test;
Query OK, 1 row affected (0.00 sec)

mysql> grant all on playland_test.* to fiona@localhost identified by 'fiona';
Query OK, 1 row affected (0.00 sec)

mysql> exit
Bye
```

5. Datenbankverbindungen anpassen

   Die Playland-Anwendung verwendet zwei Datenbanken, die in jedem Environment konfiguriert sein müssen:

   - Eine Datenbank mit dem CMS-Inhalt (`cms:`)
   - Eine Datenbank mit den Rails-spezifischen Inhalten (Seitenkommentare, -Ratings, User-Sessioninformationen etc.)
     Die Konfigurationsdatei für die Datenbankverbindungen, `database.yml`, befindet sich im Verzeichnis `config`. Kopieren Sie `config/database.yml.template` nach `database.yml` und passen Sie dann die Konfigurationen darin an Ihr Datenbank-Setup an.

   Für beide Datenbanken gibt es Einstellungen, die abhängig von der Umgebung des Rails-Servers verwendet werden. Die Datenbank mit dem CMS-Inhalt muss entspechend der Fiona Datenbank-Konfiguration (*`Instanzname`*`/config/cmdb.xml`) gesetzt werden .

6. Einstellungen der CMS-Instanz anpassen

   Sofern Sie nicht die `default`-Instanz verwenden, nehmen Sie die folgenden weiteren Einstellungen vor:

   - Tragen Sie den Instanznamen in die Datei `config/initializer/rails_connector.rb` ein:

     ```
     RailsConnector::Configuration.instance_name = Instanzname
     ```

   - Tragen Sie den HTTP-Port des SES in die Datei `config/initializer/rails_connector.rb` ein:

     ```
     RailsConnector::Configuration.search_options = {
       ...
       :port => nnn5,
       ...
     }
     ```

7. Datenbankstruktur in der Rails-Datenbank anlegen

   Wechseln Sie in das Hauptverzeichnis der Rails-Applikation und initialisieren Sie die Datenbank:

```
$ rake db:migrate
== CreateComments: migrating ==================================================
-- create_table(:comments)
   -> 0.0461s
== CreateComments: migrated (0.0467s) =========================================

== CreateRatings: migrating ===================================================
-- create_table(:ratings)
```

```
   -> 0.0064s
== CreateRatings: migrated (0.0070s) ========================================

== AddSessions: migrating ====================================================
-- create_table(:sessions)
   -> 0.0130s
-- add_index(:sessions, :session_id)
   -> 0.0140s
-- add_index(:sessions, :updated_at)
   -> 0.0156s
== AddSessions: migrated (0.0442s) ===========================================
```

Dieses `rake`-Kommando führt die Ruby-Dateien aus, die sich im Rails-Anwendungsverzeichnis `db/migrate` befinden, um die Tabellenstruktur der Datenbanken Ihrer Rails-Anwendung anzulegen oder anzupassen. Ihre CMS-Installation oder CMS-Datenbank wird dabei nicht geändert.

8. Anwendung testen

   Starten Sie den Server mit dem folgenden Befehl im Environment `development`:

   ```
   $ rails server
   => Booting WEBrick
   => Rails 3.0.5 application starting in development on http://0.0.0.0:3000
   => Call with -d to detach
   => Ctrl-C to shutdown server
   ```

   Überprüfen Sie die Funktionalität, indem Sie die Startseite der Playland-Anwendung unter der Adresse `http://localhost:3000/` aufrufen.

9. Integrieren Sie die Rails-Vorschau ins CMS-GUI.

2.4 Datenbankverbindungen konfigurieren

Zu jedem Environment einer Rails-Anwendung gehört eine Datenbankverbindung. Diese ist in der Datei `config/database.yml` konfiguriert:

```
meinProjekt_Environment:
  adapter: mysql
  database: Rails-DB
  username: Rails-DB-User
  password: Rails-DB-User-Password
  socket: /var/run/mysqld/mysqld.sock
```

Passen Sie die Einträge Ihren Wünschen entsprechend an. Verwenden Sie keinesfalls dieselbe Datenbank für mehr als ein Environment.

Eine MySQL-Datenbank können Sie folgendermaßen anlegen:

```
$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 275
Server version: 5.0.37 MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> create database Rails-DB;
Query OK, 1 row affected (0.00 sec)

mysql> grant all on Rails-DB.* to 'Rails-DB-User'@'localhost' identified by 'Rails-DB-
User-Password';
```

```
Query OK, 0 rows affected (0.00 sec)

mysql> exit
Bye
```

2.5 Den Infopark Rails Connector installieren

Installieren Sie den Infopark Rails Connector, um in Ihrem Rails-Projekt direkt auf Ihre CMS-Inhalte in der Datenbank zuzugreifen und diese für die Anzeige aufzubereiten.

Die folgende Anleitung bezieht sich auf den Infopark Rails Connector ab Version 6.7.3. Anleitungen zur Installation früherer Versionen finden Sie in den entsprechenden Handbüchern (PDF). Wenn Sie Informationen zum Update einer früheren Version des Rails Connectors auf die aktuelle Version benötigen, wenden Sie sich bitte an den Infopark Customer Service.

2.5.1 Voraussetzungen

1. Die eingesetzte Version von Infopark CMS Fiona muss mindestens 6.7.2 sein.
2. Installieren Sie Infopark CMS Fiona nach der Installationsanleitung. Der Infopark-Demo-Content (*Playland*) ist ab Version 6.7.3 nur noch in der Rails-Version verfügbar und im Rails-Connector-Package enthalten.
3. Installieren Sie die für den Rails Connector benötigte Software nach der Anleitung im Abschnitt Installationsvoraussetzungen.
4. Da die SQLite-Datenbank nicht vom Infopark Rails Connector unterstützt wird, installieren Sie bitte zuerst eine MySQL-Datenbank für Ihre CMS-Fiona-Instanz. Gehen Sie vor wie im Abschnitt Eine andere Datenbank einbinden beschrieben.
5. Legen Sie ein neues Rails-Projekt an und wechseln Sie in das Projektverzeichnis.

   ```
   $ cd meinProjekt
   ```

6. Kopieren Sie Ihre Lizenzdatei `license.xml` in das Verzeichnis `config` unterhalb des Projektverzeichnisses.

2.5.2 Installation

1. Die Datei `database.yml` muss zusätzlich zu den Datenbank-Konfigurationen für die einzelnen Environments Ihrer Rails-Anwendung die Definition für die CMS-Fiona-Datenbank enthalten:

   ```
   cms:
     adapter: mysql
     database: Fiona-DB
     username: Fiona-DB-Read-User
     password: Fiona-DB-Read-User-Password
     socket: /var/run/mysqld/mysqld.sock
   ```

   Verwenden Sie für die Verbindung des Rails Connectors zur Fiona-Datenbank einen Datenbankbenutzer, der ausschließlich das Leserecht für die Datenbank hat.
2. Laden Sie das Infopark-Rails-Connector-Package aus dem Download-Bereich herunter und installieren Sie die darin enthaltenen Gems in der folgenden Reihenfolge:

```
$ gem install infopark_omc_connector-x.y.z.gem
$ gem install mysql_blob_streaming-x.y.z.gem
$ gem install infopark_rails_connector-x.y.z.gem
```

Konfigurieren Sie zuerst Ihre Anwendung, damit sie die genannten Gems verwendet, indem Sie die Datei `Gemfile` ergänzen:

```
gem "infopark_rails_connector", "=x.y.z"
gem "infopark_rails_connector_addons", "=x.y.z"
gem "mysql_blob_streaming", "=x.y.z"
```

Alternativ können Sie die Datei `Gemfile` aus dem Package der Beispielanwendung *Playland* einfach kopieren.

3. Die Gems enthalten neben Programm-Code auch Generatoren (*engl.* "generators"), um CMS-Daten ausliefern und zusätzliche Features verwenden zu können. Führen Sie also anschließend den folgenden Befehl innerhalb Ihrer Rails-Anwendung aus:

   ```
   $ rails generate rails_connector:install
   ```

   Dieser Befehl kopiert einige Dateien in Ihre Anwendung, die von dem Rails Connector benötigt werden (z.B. für die Darstellung der Bearbeitungselemente, die Time Machine oder die Suche). Ferner wird eine Initializer-Datei unter `config/initializers/rails_connector.rb` sowie die Datei `config/local/configuration.rb` angelegt.

   Für die Bearbeitungselemente, Kommentare und Bewertungen verwendet der Rails Connector die JavaScript-Bibliothek jQuery. Falls diese Bibliothek noch nicht in der Anwendung installiert ist, weist der soeben ausgeführte Generator darauf hin. Installieren Sie in diesem Fall die Bibliothek mit dem folgenden Befehl:

   ```
   $ rails generate jquery:install
   ```

4. Mit Hilfe der Datei `rails_connector.rb` lässt sich die Funktionsweise des Rails Connectors anpassen. So lässt sich beispielsweise festlegen, welche Add-Ons aktiviert sind oder wie der Instanzname der CMS-Installation lautet.

   Sollte ihre CMS-Instanz nicht `default` heißen, so ändern Sie den Instanznamen, indem Sie die folgende Zeile des Initializers anpassen:

   ```
   RailsConnector::Configuration.instance_name = 'default'
   ```

   Mit der Datei `configuration.rb` lässt sich das Rails-Projekt an die lokale Entwicklungsumgebung anpassen.

5. Starten Sie den Rails-Application-Server mit dem folgenden Kommando:

   ```
   rails server
   ```

   Sie können den Server danach mit Ihrem Browser unter `http://localhost:3000` erreichen.

2.5.3 Nächste Schritte

1. Um detaillierte Informationen zur Verwendung der Rails-Connector-API zu erhalten, lesen sie bitte die RDoc. Starten Sie hierfür mit dem folgenden Befehl einen RDoc-Server:

   ```
   $ gem server
   ```

   Dieses Kommando startet einen Gem-Server und liefert Ihnen eine URL, unter der Sie die Dokumentation Ihrer lokal installierten Gems lesen können. Öffnen Sie diese URL in ihrem Browser (normalerweise `http://localhost:8808/`) und navigieren zu `infopark_rails_connector` oder `infopark_rails_connector_addons`. Weitere Informationen zum Gem-Server finden Sie auf der RubyGems-Website.
2. Wenn Sie die Rails-Anwendung in der Vorschau verwenden, konfigurieren Sie die Vorschau.
3. Konfigurieren Sie auch das Deployment für die verschiedenen Umgebungen.

2.6 Eine Rails-3-Anwendung auf den Rails Connector 6.7.3 umstellen

Wenn Sie eine Rails-Anwendung mit einem Rails Connector vor Version 6.7.3 betreiben, so gehen Sie bitte nach dieser Anleitung vor, um Ihre Anwendung auf den Rails Connector 6.7.3 umzustellen. Die Rails-Anwendung als solche muss bereits auf Rails 3 aktualisiert worden sein, bevor der Rails Connector auf Version 6.7.3 aktualisiert wird.

Der Rails Connector enthält auch in Version 6.7.3 einen Generator, der die benötigten Dateien in einer Rails-Anwendung installiert. Dieser Generator ist zwar darauf optimiert, den Rails Connector in eine leere Rails-Anwendung zu installieren, er kann jedoch auch verwendet werden, um eine bestehende Anwendung zu aktualisieren. Der Generator überschreibt bestehende Dateien erst, nachdem eine entsprechende Rückfrage bestätigt wurde.

Führen Sie die Aktualisierung des Rails Connectors auf Version 6.7.3 in den folgenden Schritten durch:

1. Entfernen Sie die folgenden Zeilen aus der Datei `config/initializers/rails_connector.rb`:

   ```
   config.inquiry = {
     :agent => 'name',
     :owner => 'name'
   }
   ```

2. Entfernen Sie den folgenden Aufruf aus der Datei `config/routes.rb`:

   ```
   RailsConnector::Configuration.cms_routes(map)
   ```

3. Installieren Sie jetzt den Rails Connector:

   ```
   rails generate rails_connector:install
   ```

   Der Generator überschreibt bestehende Dateien erst nach Bestätigung. Prüfen Sie die Änderungen, die der Generator vornehmen möchte, indem Sie seine Rückfrage mit ‘d’ beantworten.

Die Datei `public/javascripts/rails_connector/editmarker.js` muss überschrieben werden. Die folgenden Dateien enthalten potenziell applikationsspezifische Änderungen und sollten nicht ohne Weiteres überschrieben werden:

- `config/initializers/rails_connector.rb`
- `lib/obj_extensions.rb`
- `app/views/layouts/application.html.erb`

4. Installieren Sie jQuery:

```
rails generate jquery:install
```

Die Dateien `public/javascripts/jquery.js` und `public/javascripts/rails.js` sollten keinen applikationsspezifischen Code enthalten und müssen überschrieben werden.

5. Kommentieren Sie in der Datei `config/application.rb` die folgende Zeile ein:

```
config.action_view.javascript_expansions[:defaults] = %w(jquery rails)
```

6. Entfernen Sie die folgenden Dateien, sofern vorhanden:

- `lib/tasks/rails_connector_addons.rake`
- `lib/tasks/rspec.rake`

Rails und den Rails Connector an die lokale Entwicklungsumgebung anpassen

Bis einschließlich Version 6.7.2 konnte die Konfiguration von Rails in der Datei `config/local/configuration.rb` überschrieben werden. Die Konfiguration des Rails Connectors konnte in `config/local/initializer.rb` überschrieben werden. Der Initialisierungsprozess des Rails-Frameworks machte diese Aufteilung der Einstellungen auf zwei Dateien erforderlich.

Diese Trennung ist nun nicht mehr zwingend erforderlich. Die Dateien im Verzeichnis `config/local` können nun zusammengeführt oder nach individuellen Aspekten aufgeteilt werden. Alle Ruby-Dateien in diesem Verzeichnis werden in derselben Initialisierungsphase verarbeitet.

Lokalisierungsdateien

Ab Version 6.7.3 sind im Lieferumfang des Rails Connectors keine Lokalisierungsdateien von Ruby on Rails mehr enthalten. Für den Betrieb einer Rails-Connector-Applikation in einer anderen Sprache als Englisch werden Lokalisierungsdateien benötigt.

Benutzerattribute in der Session

Ab Version 6.7.3 werden die Attribute eines angemeldeten Benutzers, die in der Session zwischengespeichert werden, nicht mehr im `UserController` festgelegt. Die Attribute werden jetzt in der Datei `config/initializers/rails_connector.rb` und auf folgende Weise definiert:

```
RailsConnector::Configuration.store_user_attrs_in_session =
  [:login, :first_name, :last_name, :email, :id]
```

Die ID der betreffenden Person (`contact_id`) wird immer in der Session gespeichert, auch dann, wenn sie in dieser Liste nicht aufgeführt ist.

2.7 Integration der Rails-Vorschau ins GUI

Die Anleitungen in diesem Abschnitt beziehen sich auf Infopark CMS Fiona 6.7.3. Anleitungen für frühere Versionen finden Sie in den entsprechenden Handbüchern (PDF).

2.7.1 Voraussetzungen

Um in der Vorschau des Redaktionssystems die von Ihrer Rails-Anwendung erzeugten Seiten zu sehen, muss zunächst der Infopark Rails Connector installiert worden sein. Hierdurch erhält Ihre Anwendung unter anderem einen `CmsController`, der für die Anzeige der Vorschau zuständig ist. Ferner benötigen Sie einen Webserver, der als Reverse-Proxy fungieren kann (beispielsweise ein Apache-HTTP-Server).

Die folgenden Schritte zeigen exemplarisch, wie das GUI, die Rails-Applikation und der Apache-Server konfiguriert werden, um die Vorschau einzubinden.

2.7.2 Das GUI konfigurieren

Damit das GUI Vorschau-URLs erzeugt, die zu dem Routing des PageControllers der Rails-Applikation passen, muss in der Datei `WEB-INF/gui.xml` das Property `railsMapping` des Beans `systemInfo` entsprechend gesetzt werden:

```
<property name="railsMapping" value="/:id/:name" />
```

Nach Änderungen an der Konfiguration des GUI, deployen Sie es bitte folgendermaßen und starten Sie das CMS neu:

```
$ instancePath/bin/rc.npsd deploy GUI
$ instancePath/bin/rc.npsd restart
```

2.7.3 Die Rails-Anwendung konfigurieren

Erzeugen Sie das Environment `preview`, indem Sie die Datei `config/environments/production.rb` nach `config/environments/preview.rb` kopieren. Damit der Rails Connector in diesem Environment Arbeitsversionen darstellt, fügen Sie die folgende Zeile hinzu:

```
RailsConnector::Configuration.mode = "editor"
```

Ergänzen Sie ferner einen Abschnitt `preview` in ihrer Datenbankkonfiguration und tragen Sie dort die Datenbank ein, die sie für den Vorschaubetrieb verwenden möchten.

Darüber hinaus ist es erforderlich, dass der Webserver den Instanznamen dem Rails-Application-Server übergibt. Wenn Sie den Apache Webserver mit Phusion Passenger einsetzen, geben Sie den Instanznamen als `RailsBaseURI` in der Konfiguration des Webservers an:

```
RailsBaseURI /instance-name
```

2.7.4 Den Apache-Webserver konfigurieren

Damit aus Sicht des Webbrowsers die Kommunikation mit dem GUI und der Rails-Applikation nicht als XSS erscheint, benötigen Sie einen Webserver wie den Apache, der als Proxy-Server zwischen Client und GUI/Rails steht. An welchen der beiden Server ein Request vom Browser gerichtet ist, erkennt der Proxy am URL-Präfix.

Bitte ergänzen Sie die Apache-Konfigurationsdatei `httpd.conf` um die folgenden Einträge:

```
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_http_module modules/mod_proxy_http.so

# ..

ProxyRequests Off
ProxyVia On

# Anfragen an das GUI
ProxyPass /instance-name/NPS http://localhost:8080/instance-name/NPS
ProxyPassReverse /instance-name/NPS http://localhost:8080/instance-name/NPS
```

Stellen Sie bitte sicher, dass die Proxy-Module wie im obigen Beispiel eingebunden sind. Platzieren Sie die Konfigurationseinträge außerhalb eines Containers, damit die Einstellungen serverweit gelten. Damit die Änderungen wirksam werden, starten Sie bitte den Apache-Server neu. Die GUI-URL ändert sich dadurch zu http://`gui-host`:`apache-port`/`instance-name`/NPS. Insbesondere ist die Vorschau nicht mehr auf Port 8080 (Standardeinstellung) zu erreichen.

2.8 Die Rails-Anwendung deployen

2.8.1 Voraussetzungen

Um Ihre Rails-Anwendung auf einem entfernten Server zu deployen, benötigen Sie das Tool Capistrano. Installieren Sie daher das Gem `capistrano`:

```
# gem install capistrano --include-dependencies
```

In dieser Anleitung wird davon ausgegangen, dass Sie in Ihrer Rails-Anwendung den Infopark Rails Connector installiert haben.

2.8.2 Konfiguration

Richten Sie Capistrano für Ihre Rails-Anwendung ein:

```
$ capify meinProjekt
[add] writing `meinProjekt/Capfile'
[add] writing `meinProjekt/config/deploy.rb'
[done] capified!
```

Bearbeiten Sie anschließend bitte die Datei *meinProjekt*`/config/deploy.rb` und ersetzen ihren Inhalt durch Folgendes:

```
set :application, "meinProjekt"
set :repository, "Repository-URL"
```

```
set :deploy_via, :copy
set :copy_strategy, :export
set :user, "Remote-Login"
set :deploy_to, "Zielpfad"
set :use_sudo, false

# Use the command line switch -S to preselect a stage.
# Example:
#   cap -S stage=production deploy
unless fetch(:stage, nil)
  set :stage do
    Capistrano::CLI.ui.choose do |menu|
      menu.header = "The following stages are available"
      menu.choice "preview"
      menu.choice "staging"
      menu.choice "production"
      menu.prompt = "Please choose: "
    end
  end
end

set :rails_env, stage

case rails_env
when "preview"
  role :web, "Preview-Web-Server"
  role :app, "Preview-App-Server"
  role :db,  "Preview-DB-Server", :primary => true
  set :prefix, "/CMS-Instanzname"
when "staging"
  role :web, "Staging-Web-Server"
  role :app, "Staging-App-Server"
  role :db,  "Staging-DB-Server", :primary => true
when "production"
  role :web, "Production-Web-Server"
  role :app, "Production-App-Server"
  role :db,  "Production-DB-Server", :primary => true
end

namespace :deploy do

  task :start do ; end
  task :stop do ; end
  desc 'Restart Rails App using Phusion Passenger'
  task :restart, :roles => :app do
    run "touch #{File.join(current_path,'tmp','restart.txt')}"
  end

  task :symlinks do
    %w(database.yml license.xml initializers/rails_connector.rb).each do |file|
      run "ln -nfs #{shared_path}/config/#{file} #{release_path}/config/#{file}"
    end
  end
end

before "deploy:setup", :db
after "deploy:update_code", "deploy:symlinks"

set :keep_releases, 10
after "deploy:update", "deploy:cleanup"

require 'bundler/capistrano'
```

Passen Sie die Konfiguration jetzt Ihren Bedürfnissen entsprechend an:

- *meinProjekt* ist der Verzeichnisname Ihrer Rails-Anwendung.
- *Subversion-URL* ist die Subversion-Adresse Ihrer Rails-Anwendung.

- *Remote-Login* ist das Login, unter dem Ihre Rails-Anwendung auf dem Zielrechner deployed werden soll.
- *Zielpfad* ist das Verzeichnis, in dem Ihre Rails-Anwendung auf dem Zielrechner deployed werden soll.
- *CMS-Instanzname* ist der Name der CMS-Instanz, für die der Preview-Server bestimmt ist.
- *Environment-Rolle-Server* ist der Zielrechner für ein bestimmtes Environment (`preview`, `staging` oder `production`) und eine bestimmte Rolle (`:web`, `:app` oder `:db`).

In allen Werten können Sie bereits gesetzte Variablen verwenden. Wenn Sie beispielsweise den Projektnamen in der Subversion-Adresse verwenden möchten, könnte dies beispielsweise folgendermaßen aussehen: `svn://svn/meinRepository/#{application}`.

Speichern Sie die Konfiguration, nachdem Sie sie an Ihre Anforderungen angepasst haben.

2.8.3 Initiales Deployment

Bevor Sie Ihre Rails-Anwendung für ein Environment deployen können, muss es eingerichtet werden. Führen Sie hierzu bitte den folgenden Befehl aus:

```
$ cap deploy:setup
1. preview
2. staging
3. production
Please choose: 2
  * executing `deploy:setup'
  * executing "umask 02 && mkdir -p /tmp/meinProjekt /tmp/meinProjekt/releases /tmp/
meinProjekt/shared /tmp/meinProjekt/shared/system /tmp/meinProjekt/shared/log /tmp/
meinProjekt/shared/pids"
    servers: ["ip20-127"]
    [ip20-127] executing command
    command finished
```

Am Ende der Konfiguration sind die Software-Pakete aufgeführt, die auf dem Zielrechner installiert sein müssen. Nachdem Sie ein Environment eingerichtet haben, können Sie überprüfen, ob auf dem Zielrechner die benötigte Software installiert ist, indem Sie das folgende Kommando ausführen:

```
$ cd meinProjekt
$ cap deploy:check
1. preview
2. staging
3. production
Please choose: 2
  * executing `deploy:check'
[...]
You appear to have all necessary dependencies installed
```

Wenn alle Voraussetzungen erfüllt sind, können Sie Ihre Rails-Anwendung deployen:

```
$ cap deploy:cold
1. preview
2. staging
3. production
Please choose: 2
  * executing `deploy:cold'
[...]
    command finished
```

`deploy:cold` migriert vor dem Deployment die Datenbank auf den aktuellen Stand.

2.8.4 Erneutes Deployment

Wenn Ihre Rails-Anwendung bereits deployed ist, können Sie sie neu deployen, indem Sie das folgende Kommando ausführen:

```
$ cap deploy
1. preview
2. staging
3. production
Please choose: 2
  * executing `deploy'
[...]
    command finished
```

Wenn sich im Zuge eines Updates ihrer Rails-Anwendung das Datenbankschema ändert, so verwenden Sie zum Deployen `cap deploy:migrations`, um die Datenbank auf den aktuellen Stand zu migrieren.

2.8.5 Kommandos für weitere Administrationsaufgaben

Capistrano bietet neben den oben beschriebenen Kommandos (Tasks) auch solche zu weiteren Administrationsaufgaben an. Sie können die verfügbaren Kommandos folgendermaßen ausgeben lassen:

```
$ cap -T
1. preview
2. staging
3. production
Please choose: 1
cap deploy               # Deploys your project.
cap deploy:check         # Test deployment dependencies.
[...]
cap deploy:web:disable   # Present a maintenance page to visitors.
cap deploy:web:enable    # Makes the application web-accessible again.
cap invoke               # Invoke a single command on the remote servers.
cap shell                # Begin an interactive Capistrano session.
```

2.9 Datenbankinhalte übertragen (am Beispiel von MySQL)

Nachdem Sie Ihre Entwicklungsumgebung eingerichtet haben, benötigen Sie in der Datenbank die Daten, auf denen Ihre Rails-Anwendung operieren soll. Um die Datenbankinhalte des Live-Systems lokal zu duplizieren, gehen Sie bei MySQL-Datenbanken folgendermaßen vor:

- Melden Sie sich am Quellserver als ein Benutzer an, der Zugriff auf die Datenbank hat.
- Führen Sie auf der Kommandozeile das Folgende mit den entsprechenden Parametern wie Datenbank-Benutzername, -Passwort und Zielverzeichnis aus:

```
$ mysqldump -h MySQL-Host -u MySQL-Login -p DB-Name > dump.sql
Enter password: MySQL-Passwort
```

Im obigen Beispiel wird der Datenbank-Dump unkomprimiert unter dem Namen `dump.sql` im aktuellen Verzeichnis abgelegt.

- Transferieren Sie den Dump beispielsweise mit `scp` auf den Entwicklungsrechner.
- Importieren Sie den Dump in eine neue MySQL-Datenbank. Hier ein Ablaufbeispiel:

```
Entwicklungsrechner:~ $ mysql -u root -p
Enter password: MySQL-Passwort
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 275
Server version: 5.0.37 MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> create database railsdb;
Query OK, 1 row affected (0.00 sec)

mysql> grant all on railsdb.* to railsuser@localhost identified by 'railspassword';
Query OK, 0 rows affected (0.00 sec)

mysql> exit
Bye


Entwicklungsrechner:~ $ mysql -u railsuser -prailspassword railsdb < dump.sql
```

2.9.1 Hinweise

Verwenden Sie als Datenbank-User und dessen Passwort die gleichen Angaben wie in der Datenbankkonfiguration (`config/database.yml`).

2.10 Die CMS-Datenbank replizieren

Die folgenden Datenbanktabellen müssen bei der Replikation einbezogen werden. Der Tabellen-Präfix (beispielsweise `default` entspricht dem Namen der Instanz):

Ab Version 6.7.0

```
default_objs
default_blobs
default_permissions
default_news
```

Aus der `default_blobs`-Tabelle werden nur die Einträge benötigt, bei denen `blob_name` gleich `%.blobs` ist (`select * from default_blobs where blob_name like '%.blob'`).

Bis Version 6.6.1

```
default_attributes
default_blobs
default_channels
```

```
default_contents
default_links
default_news
default_obj_class_attrs
default_obj_classes
default_objects
default_permissions
```

Schritte nach der Replikation

Durch die Replikation der Tabellen `*_blobs`, `*_objs` (bzw. `*_objects`) und `*_news` entstehen Bedingungen (*constraints*), die nicht erfüllt werden können, weil sie sich auf nicht replizierte Tabellen beziehen. Daher ist es unmittelbar nach der Replikation erforderlich, diese Bedingungen zu entfernen:

```
ALTER TABLE instanzName_permissions DROP FOREIGN KEY internet_permfk_1;
ALTER TABLE instanzName_news DROP FOREIGN KEY internet_newsfk_1;
```

2.11 CMS-Inhalte mit ERb-Templates ausliefern

Der CMS-Controller des Infopark Rails Connectors liefert Inhalte von Dokumenten und Ordnern mit Hilfe eines Views aus. Durch das Routing wird dabei genau eine Datei als aktueller Kontext ausgewählt. Im View kann diese Datei – als Instanz der Klasse `Obj` – unter dem Namen @obj referenziert werden.

Bei ERb-Templates werden die dynamischen Inhalte durch Ruby-Anweisungen im HTML-Text des Views eingefügt. Um beispielsweise den Namen der aktuellen Datei auszugeben, schreiben Sie:

```
<%= @obj.name %>
```

Auf Felder der Version kann auch direkt zugegriffen werden. Es gibt zwei Varianten, mit denen beispielsweise auf den Titel der freigegebenen Version der Datei zugegriffen werden kann (wenn keine freigegebene Version existiert, tritt bei beiden Varianten ein Fehler auf):

```
<%= @obj.title %>
<%= @obj[:title] %>
```

Die Variante `@obj.title` wird empfohlen, wenn Sie im Falle eines fehlenden Feldes eine Fehlermeldung erzeugen möchten. Soll dagegen ein fehlendes Feld ignoriert werden, verwenden Sie die Schreibweise `@obj[:title]`.

Damit die Werte korrekt dargestellt werden, ist es notwendig, den Helper `display_field` zu verwenden (bis 6.6.1 wird anstelle der Helper `display_field` und `display_value` der Helper `display` verwendet). Er wird folgendermaßen eingebunden:

```
<%= display_field @obj, :title %>
```

Durch diese Schreibweise wird automatisch auch ein Bearbeitungselement in die Vorschau integriert. Bearbeitungselemente können auf mehrere Arten unterdrückt werden. Verwenden Sie eine der folgenden Schreibweisen (bis 6.6.1 wird anstelle der Helper `display_field` und `display_value` der Helper `display` verwendet):

```
<%= display_value @obj.title %>
```

```
<%= display_field @obj, :title, :marker => false %>
```

Komplexere Funktionen wie die Erzeugung einer Navigation werden aus Gründen der Übersichtlichkeit, der Wiederverwendbarkeit und der Qualitätssicherung nicht direkt in die Views aufgenommen. Hierfür werden so genannte Helpers verwendet.

Für HTML-Fragmente, die wiederverwendet werden sollen, stellt das Rails-Framework die Möglichkeit bereit, sie als sogenannte Partials auszulagern.

2.12 CMS-Inhalte mit Liquid-Templates ausliefern

2.12.1 Was ist Liquid?

Liquid ist eine in Ruby implementierte Template-Sprache mit Fokus auf Sicherheit und Robustheit. Liquid-Templates sind einfacher zu entwickeln als ERb-Templates. Daher können Templates mit Liquid häufig schneller und mit geringerer Fehlergefahr umgesetzt werden. Um fehlende kundenspezifische Felder und nil-Werte braucht man sich als Template-Autor nicht zu kümmern. Liquid stellt hier (im Gegensatz zu ERb) “graceful degradation” sicher.

Grundlagenwissen über die Verwendung von Liquid gibt es auf den folgenden Webseiten:

- http://wiki.github.com/tobi/liquid/liquid-for-designers
- http://wiki.shopify.com/UsingLiquid

Der Infopark Rails Connector erweitert die Grundsyntax von Liquid um spezielle Sprachkonstrukte, mit denen CMS-Inhalte angezeigt werden können. Diese werden im Folgenden beschrieben.

2.12.2 Inhalte ausgeben und Sub-Templates einbinden

Dieses Liquid-Template gibt den Titel und Hauptinhalt der aktuellen Seite (repräsentiert durch `obj`) aus und ruft dann ein weiteres Template auf:

```
<h1>{{ obj.title }}</h1>
{{ obj.body }}

{% template 'toclist' %}
```

Das CMS-Objekt `obj` ist in einem Drop gekapselt, das die Zugriffe auf die Methoden und Felder des Objekts regelt. Der Drop wandelt Text automatisch in dessen HTML-Darstellung um, löst Links auf und fügt Bearbeitungselemente ein.

2.12.3 Kontextlisten, Linklisten, Links und Bilder

Das folgende Template gibt eine Liste der Unterordner und -Dokumente (`toclist`) aus, falls das betreffende CMS-Objekt ein Ordner ist. Jedes Element erscheint mit einem Vorschaubild und dem verlinkten Titel.

```
<ul>
{% for object in obj.toclist %}
  <li>{{ object.thumbnail | image_tag }}
  {{ object.title | link_to: object }}</li>
{% endfor %}
```

```
</ul>
```

Die Vorschaubilder werden in diesem Beispiel über das einelementige Linklistenfeld `thumbnail` eingebunden, aus der der Filter `image_tag` ein im Browser darstellbares Bild erzeugt. Mit Hilfe des `link_to`-Filters wird ein HTML-Link erzeugt, über den man im Browser zum Zielobjekt gelangen kann.

Linklisten mit mehreren Elementen können in Liquid ebenfalls verwendet werden:

```
<ul>
{% for link in obj.related_links %}
  <li>{{ link | link_to }}</li>
{% endfor %}
</ul>
```

Enthält ein Link einen ausgefüllten Titel, verwendet der Filter `link_to` diesen als Linktext. Andernfalls setzt er bei internen Links den Titel der referenzierten Datei ein und bei externen Links die URL.

Auf einzelne Links aus einer Linkliste kann auch über den Index des Links zugegriffen werden:

```
{{ "Ein Ersatz-Linktext" | link_to: obj.related_links[2] }}
```

Der Standard-Linktext wird in diesem Beispiel durch einen benutzerdefinierten Text ersetzt.

2.12.4 Bearbeitungselemente automatisch erzeugen

Wird die Rails-Applikation als Vorschauansicht in das GUI des CMS eingebunden, so gibt es die Möglichkeit, Bearbeitungselemente für die dargestellten CMS-Inhalte zu integrieren. Bei der Verwendung von Liquid-Templates können die Bearbeitungselemente entweder automatisch oder manuell gesetzt werden.

Voreingestellt werden die Bearbeitungselemente automatisch gesetzt. Gibt man beispielsweise den Titel eines CMS-Objekts aus, so erscheint in der Vorschau automatisch ein Bearbeitungselement:

```
{{ obj.title }}
```

Um im Einzelfall kein Bearbeitungselement auszugeben, kann der Filter `editmarker` verwendet werden:

```
{{ obj.title | editmarker: false }}
```

2.12.5 Bearbeitungselemente manuell ausgeben

Um die automatische Ausgabe von Bearbeitungselementen für alle Liquid-Templates einer Rails-Anwendung global zu deaktivieren, wird folgende Zeile in die Konfiguration aufgenommen:

```
RailsConnector::Configuration.auto_liquid_editmarkers = false
```

Die folgende Liquid-Anweisung führt dann nicht mehr dazu, dass ein Bearbeitungselement erzeugt wird:

```
{{ obj.title }}
```

Der Autor eines Templates kann jedoch manuell mit folgender Syntax Bearbeitungselemente erzeugen:

```
{{ obj.title | editmarker }}
```

2.12.6 URLs für Inhalte generieren

Mit dem Filter `url` kann man sich die URL geben lassen, unter der ein CMS-Objekt erreichbar ist:

```
<object data="external:{{ object | url }}" />
```

Dies ist beispielsweise dann nützlich, wenn ein CMS-Objekt als Flash-Content oder als Java Applet in eine HTML-Seite eingebunden werden soll.

2.12.7 Felder von Objekten auf Werte prüfen

```
{% if obj.mein_feld != blank %}
  Mein Feld enthält den Wert {{ obj.mein_feld }}.
{% else %}
  Mein Feld enthält keinen Wert.
{% endif %}
```

Im obigen Code wird der `else`-Zweig in den folgenden Fällen ausgewertet:

- `obj.mein_feld` enthält einen leeren String
- `obj.mein_feld == nil`
- `obj` hat gar kein Feld `mein_feld`

Daher ist der Test `obj.mein_feld != blank` der beste Weg, um vor leeren Feldwerten zu schützen. Mit einer ähnlichen Syntax lässt sich explizit testen, ob ein String leer oder der Feldwert `nil` ist:

```
{% if obj.mein_feld == empty %}
  Mein Feld enthält einen leeren String.
{% endif %}
{% if !obj.mein_feld %}
  Mein Feld ist nicht vorhanden (== nil).
{% endif %}
```

2.12.8 Zeit oder Datum formatiert ausgeben

Zeit- und Datumswerte können mit Hilfe des `date`-Filters ausgegeben werden:

```
{{ obj.ip_eventStart | date: "%d.%m.%Y" }}
```

2.12.9 Auf benannte Objekte (Named Links) zugreifen

Der folgende Template-Code holt das CMS-Objekt, das durch den benannten Link `my_name` referenziert wird und gibt dessen Titel aus:

```
{{ named_object["my_name"].title }}
```

Benannte Links können im CMS als Links in der Linkliste `relatedLinks` einer CMS-Datei mit der Vorlage `NamedLink` gespeichert werden. Das Ziel eines Links in der Linkliste kann im Rails Connector über den Linktitel referenziert werden.

2.12.10 Action-Marker einbinden

In der integrierten Vorschau können Menübefehle und registrierte Assistenten per Action-Marker verfügbar gemacht werden. Der Template-Code, mit dem sich beispielsweise ein Bildbearbeitungsassistent einbinden lässt, könnte so aussehen:

```
{% actionmarker obj.some_image editImageWizard %}
  [Bild bearbeiten]
{% endactionmarker %}
```

2.13 Eigene Liquid-Filter definieren

2.13.1 Filter in Liquid

Um Texte formatiert auszugeben, können in Liquid-Templates sogenannte Filter verwendet werden. Liquid liefert einige Standardfilter mit, etwa den Filter `truncate`, der eine Ausgabe auf eine bestimmte Anzahl Zeichen beschneidet:

```
Die ersten 50 Zeichen des Hauptinhalts lauten:
{{ page.body | truncate: 50 }}
```

Eine Übersicht über die Standardfilter findet sich in der Syntaxbeschreibung von Liquid.

2.13.2 Eigene Filter definieren und einbinden

Der Entwickler einer Rails-Connector-Applikation kann eigene Filter definieren und diese zum Gebrauch in den Liquid-Templates der Applikation zur Verfügung stellen. Ein Filter ist im Grunde nur eine Methode, die mindestens einen Parameter hat und die einen modifizierten Wert zurückgibt.

Im folgenden Beispiel wird ein Filter definiert, der einen Text mit `h1`-Tags umschließt:

```
module MyFilters

  def headline(input)
    "<h1>#{input}</h1>"
  end

end
```

Ein Filter muss in einem Modul definiert sein, das innerhalb der Rails-Applikation im Ordner `/app/filters` liegt. Der obige Filter müsste also in der Datei `/app/filters/my_filters.rb` liegen. Der Filter könnte dann beispielsweise folgendermaßen in einem Liquid-Template verwendet werden:

```
{{ "Meine Überschrift" | headline }}
```

Das Ergebnis wäre:

```
<h1>Meine Überschrift</h1>
```

2.13.3 Weitere Informationen

Weitere Anregungen für die Erstellung von Liquid-Filtern kann der Quellcode der Standardfilter von Liquid liefern.

2.14 Views erweitern

Diese Anleitung richtet sich an Entwickler, die eine existierende Rails-Applikation, die bereits CMS-Inhalte mit Hilfe des Infopark Rails Connectors darstellt, um einen View erweitern möchten. Um dies tun zu können, müssen die folgenden Fragen geklärt werden.

- Wo und wie wird darüber entschieden, welcher View verwendet werden soll?
- Wo und wie legt man einen neuen View an?
- Wie füllt man einen View mit Inhalt?

Wo und wie wird darüber entschieden, welcher View verwendet werden soll?

In herkömmlichen Rails-Applikationen ist es selten nötig, im Controller explizit anzugeben, welcher View zur Darstellung verwendet werden soll. In der Regel folgt man der Konvention, den View genauso zu nennen wie die betreffende Aktion des Controllers. Rails würde entsprechend dieser Konvention für die Aktion `CmsController#index` einen View unter `app/views/cms/index.html.erb` erwarten und versuchen, diesen zu laden.

Solange alle Objekte mit identischem Layout dargestellt werden sollen, können Sie dieser Konvention folgen. Typischerweise existieren allerdings CMS-Objekte unterschiedlicher Vorlagen (`objClass`), in unterschiedlichen Pfaden oder mit unterschiedlichen Feldinhalten. Welche Funktion diese Unterschiede in der Web-Anwendung haben, ob sie sich beispielsweise auch auf die Anzeige auswirken, hängt vom Einzelfall ab.

Für den Fall, dass Objekte je nach Vorlage in einem anderen Layout dargestellt werden sollen, verfügt der Rails Connector über vorlagen-spezifische Controller.

Wo und wie legt man einen neuen View an?

Nutzt man vorlagen-spezifische Controller, müssen Views in einem nach dem Controller benannten Unterverzeichnis von `app/views/` angelegt werden.

Wie füllt man einen View mit Inhalt?

In Anwendungen mit dem Rails Connector unterscheidet sich der Inhalt von Views konzeptionell nicht von Views in herkömmlichen Rails-Anwendungen. Es ist jedoch wichtig zu wissen, dass in jedem View eine `@obj`-Instanz zur Verfügung steht, die das aktuell geladene CMS-Objekt repräsentiert. Je nachdem, um welches Objekt es sich dabei handelt, d.h. welche Vorlage (`objClass`) ihm zugewiesen wurde und über welche Felder es dadurch verfügt, kann auf eine bestimmte Menge von Feldern zugegriffen werden. Dazu gehören CMS-interne Felder wie `title`, `name`, `path`, `valid_from` und `valid_until`, jedoch auch kundenspezifische Felder wie zum Beispiel `abstract` und `showintoc`.

Zur Anzeige von Feldwerten stehen Hilfsmethoden (Helpers) zur Verfügung. Welche Helpers für welche Felder am besten geeignet sind, ist in der API-Dokumentation des Rails Connectors beschrieben.

2.15 Mitgelieferte Views des Rails Connectors anpassen

Die folgenden Bestandteile des Rails Connectors enthalten Views, die ausgetauscht oder angepasst werden können (hierfür sind Ruby-on-Rails-Grundkenntnisse erforderlich):

- CMS Controller
- Suche
- Time Machine
- Kommentare
- Ratings
- RSS-Feed
- SEO-Sitemap
- OMC-Anbindung

Die View-Dateien sind nicht in den Verzeichnissen Ihrer Rails-Anwendung zu finden, da direkt auf die im Gem liegenden Dateien zugegriffen wird. Die einfachste Möglichkeit, die Darstellung zu einer Controller-Aktion zu modifizieren, besteht darin, in der Applikation einen View mit entsprechendem Pfad und Namen anzulegen, und auf diese Weise den gleichnamigen mitgelieferten View zu ersetzen. Für die Aktion `SearchController#search` müsste beispielsweise die Datei `app/views/search/search.html.erb` angelegt werden.

Auch der Austausch mitgelieferter Partials ist möglich. Hierfür ist es empfehlenswert, sich vorher einen Überblick darüber zu verschaffen, welche Partials im Rails-Connector-Gem existieren und an welchen Stellen sie eingebunden sind. Der Installationsort des Rails-Connector-Gems kann mit dem Kommando

```
$ gem which infopark_rails_connector
```

ausfindig gemacht werden.

Die Views und Partials befinden sich jeweils im Unterverzeichnis `app/views/`*`controller-name`*, also beispielsweise für die Suche im Verzeichnis `app/views/search`. Legt man im entsprechenden Unterverzeichnis der Applikation eine Datei an, die schon im Gem unter gleichem Namen vorhanden ist, ersetzt diese automatisch die mitgelieferte Datei.

2.16 Fehlerseiten anpassen

Der Infopark Rails Connector bietet die Möglichkeit, das Aussehen der Fehlerseiten für die Status-Codes 403 (Zugriff verweigert) und 410 (Seite existiert nicht) sowie das Verhalten der Rails-Anwendung bei Auftreten eines dieser Fehler anzupassen.

Um das Aussehen der jeweils angezeigten Fehlerseite anzupassen, können Sie ein eigenes Template im Verzeichnis `app/views/errors/` anlegen.

Um eigenen Code auszuführen, sobald einer dieser Fehler auftritt, können Sie den entsprechenden Controller (für 403-Fehler) bzw. den `CmsController` (für 410-Fehler) wunschgemäß erweitern. Details hierzu finden Sie in der RDoc-Dokumentation zum Modul `CmsAccessible`, die mit dem Infopark Rails Connector geliefert wird.

2.17 Permalinks aktivieren

Um Permalinks im Redaktionssystem definieren zu können, ist es erforderlich, die Konfiguration in der Datei `gui.xml` im Verzeichnis `webapps/GUI/WEB-INF` der entsprechenden Instanz folgendermaßen zu ändern:

```
<bean id="inspectorRegistry" ...>
    ...
    <property name="enablePermalinks" value="true" />
</bean>
```

Anschließend muss das GUI neu deployed werden:

```
./bin/rc.npsd deploy GUI
```

In der Detailübersicht erscheinen nun die Permalinks:

Als Permalink kann eine eindeutige Zeichenkette angegeben werden, die aus Zeichen besteht, die in URLs zulässig sind. Eine solche Zeichenkette wird zum Pfad der URL der betreffenden CMS-Datei. Die Live-Server-Applikation hat ebenfalls Zugriff auf Permalinks und ist in der Lage, die Datei zusätzlich unter diesem Pfad ausliefern.

2.18 Formulare für OMC-Aktivitäten generieren

Wenn der Rails Connector an das Online Marketing Cockpit angebunden ist, d.h. der OMC Connector installiert und eingeschaltet ist, können Sie in Ihrer Rails-Anwendung Formulare auf der Basis von Aktivitäten im OMC erzeugen. Schickt ein Website-Besucher ein solches Formular ab, sorgt der Rails Connector dafür, dass im OMC eine Aktivität angelegt wird. So können beispielsweise Formulare

für Kundenanfragen, Anmeldungen zu Veranstaltungen und andere Vorgänge automatisch und wartungsfrei erzeugt werden. Diese Vorgänge werden im OMC als Aktivitäten angelegt und können dort bequem bearbeitet oder auch für den Versand von Mailings genutzt werden.

Um Seiten mit solchen generierten Formularen im Redaktionssystem anzulegen, benötigen Sie lediglich eine Vorlage namens `OmcForm` und eine auf dieser Vorlage basierende CMS-Datei. Durch den für diese Vorlage mitgelieferten View erzeugt Ihre Rails-Anwendung dann ein Formular für den Aktivitätstyp `contact form`, wenn die CMS-Datei ausgeliefert wird.

Technische Details und Erweiterungsmöglichkeiten

Die Rails-Anwendung liefert die auf der Vorlage `OmcForm` basierende CMS-Datei mit Hilfe des mitgelieferten Controllers `OmcFormController` aus. In dem dazu gehörenden View sorgt der Helper `OmcFormHelper` dafür, dass die Formularelemente erzeugt werden.

Der `OmcFormController` ist vom `DefaultOmcFormController` abgeleitet. Dieser verfügt über Methoden, die in den abgeleiteten Klassen überschrieben werden können, um die Voreinstellungen des zu erzeugenden Formulars zu ändern – beispielsweise den zu verwendenden Aktivitätstyp (`activity_kind`).

Über den Aktivitätstyp – voreingestellt `contact form` – ermittelt der `OmcFormHelper` die zu erzeugenden Formularfelder sowie den Typ der Aktivität, die im OMC angelegt werden soll, wenn das Formular abgeschickt wird. Der Helper generiert Formularfelder für alle zusätzlichen Felder des Aktivitätstyps sowie optional für den künftigen Titel der anzulegenden Aktivität. Mit Hilfe eines Callbacks (`allow_custom_attribute?`) kann bei Zusatzfeldern die Ausgabe eines Formularelements unterdrückt werden.

Um unterschiedlich gestaltete Seiten mit generierten Formularen im CMS anzulegen, können Sie individuelle Vorlagen als Alternativen oder ergänzend zu `OmcForm` erstellen. Wenn Sie beispielsweise ein Formular für die Anmeldung an einer Veranstaltung ausgeben möchten und hierfür die Vorlage `OmcEventForm` angelegt haben, so brauchen Sie in Ihrer Rails-Anwendung lediglich die Klasse `OmcEventFormController` zu definieren und von `RailsConnector::DefaultOmcFormController` abzuleiten. Um das Verhalten von `OmcEventFormController` anzupassen, können Sie die Methoden dieser Klasse überschreiben. Wie neue Controller angelegt und erweitert werden können, ist in den Abschnitten Vorlagen-spezifische Controller und Den Rails Connector anpassen beschrieben.

In dem gegebenenfalls anzulegenden View, der zu dem neuen, abgeleiteten Controller gehört, können Sie nun die Seite gestalten und darin das Formular unter Zuhilfenahme von `OmcFormHelper` erstellen.

Weitere Informationen über die Formularerzeugung finden Sie in der mitgelieferten RDoc-Dokumentation zur Klasse `DefaultOmcFormController`.

2.19 Website-Funktionen aktivieren

2.19.1 Kommentare auf Webseiten

Der Rails Connector unterstützt die Eingabe und Anzeige von Kommentaren auf Webseiten. Hierfür enthält er ein Datenbankmodell sowie einen Controller und ein Partial.

Um die Kommentarfunktion nutzen zu können, muss sie zunächst aktiviert werden. Bearbeiten Sie hierzu bitte die Datei `config/initializers/rails_connector.rb` und entfernen Sie das #-Zeichen aus der Zeile `#:comments,:`

```
RailsConnector::Configuration.enable(
…
:comments,
…
)
```

Lassen Sie anschließend mit dem folgenden Kommando die erforderlichen Datenbank-Migrationsschritte erzeugen:

```
rails generate rails_connector:comments
```

Führen Sie nun die beiden generierten Migrationsschritte aus, um die für die Kommentare erforderlichen zusätzlichen Tabellen in Ihrer Datenbank zu erzeugen:

```
rake db:migrate
```

Um die Kommentare und das Formular zur Erfassung eines neuen Kommentars auf einer Seite anzeigen zu lassen, können Sie das mitgelieferte Partial `comments` verwenden:

```
<%= render :partial => 'cms/comments' %>
```

Um die Verarbeitung von Kommentaren zu erweitern, können Sie eine eigene Klasse schreiben, die vom `DefaultCommentsController` abgeleitet ist:

```
class CommentsController < RailsConnector::DefaultCommentsController
# Ihre Anpassungen
end
```

2.19.2 RSS-Feeds bereitstellen

Der Rails-Connector enthält einen Controller, mit dem RSS-Feeds ausgeliefert werden können. Bevor dieser genutzt werden kann, muss das RSS-Feature aktiviert werden. Bearbeiten Sie hierzu die Datei `config/initializers/rails_connector.rb` und entfernen Sie aus der Zeile `# :rss,` das Kommentarzeichen `#`:

```
RailsConnector::Configuration.enable(
…
:rss,
…
)
```

Voreingestellt wird der RSS-Feed aus den CMS-Dateien generiert, die sich in einem definierten Ordner befinden. Dieses Ordner-Objekt muss in der Datei `config/initializers/rails_connector.rb` geladen werden, beispielsweise über einen NamedLink:

```
# RSS-Feed:
#
# Specify which Object should be used for the RSS feed's parent folder
RailsConnector::Configuration::Rss.root = NamedLink.get_object('news')
```

Der RSS-Feed wird voreingestellt über den Controller `DefaultRssController` und mit dem Template `app/views/rss/_item.rss.builder` generiert. Um den Feed auf andere Art und Weise zu erzeugen, können Sie dieses Template ändern oder einen eigenen Controller schreiben, der auf ein anderes Template verweist:

```
class RssController < RailsConnector::DefaultRssController
  def podcast_feed
    @obj = NamedLink.get_object('podcast_de')
    headers['Content-Type'] = 'application/rss+xml'
    render :layout => false
  end
end
```

2.20 Integration der "Time Machine"

Damit die Time Machine funktioniert, erzeugen Sie bitte zunächst die erforderlichen Dateien, indem Sie den Generator `rails_connector_addons` aufrufen (hierfür muss das Gem `infopark_rails_connector_addons` installiert sein):

```
$ script/generate infopark_rails_connector_addons
```

Der Generator legt im Verzeichnis `config/initializers` die Konfigurationsdatei `infopark_rails_connector_addons.rb` an, in der alle verfügbaren Addons aktiviert oder deaktiviert werden können – darunter die Time Machine, die voreingestellt eingeschaltet ist. Um sie zu deaktivieren, entfernen Sie den Eintrag `:time_machine` aus der Liste `RailsConnector::Configuration.enable()` in dieser Datei.

Mit Hilfe des `TimeMachineHelper` können Sie einen Link in die Seite einbinden, der die Time Machine öffnet:

```
<%= time_machine_link 'Zeitmaschine' %>
```

Falls auf derselben Seite bereits der `MenuHelper` verwendet wird, so ist es unbedingt erforderlich, in seinem Aufruf den Parameter `current_time` anzugeben, damit der Zeitpunkt auch für die in der erzeugten Navigation verlinkten Dokumente gilt. Siehe hierzu die Dokumentation des Helpers im `doc`-Verzeichnis des Infopark-Rails-Connector-Addon-Gems.

2.21 Die Suche mit dem Infopark Search Server aktivieren

Der Rails Connector enthält eine fertige Suchseite, die Sie in Ihrer Rails-Applikation mit der URL `/search` aufrufen können.

Wenn Sie den Rails Connector nach der Installationsanleitung installiert haben, liegt in Ihrer Rails-Applikation im Verzeichnis `config/initializers` die Konfigurationsdatei `rails_connector.rb`, in der alle verfügbaren Addons aktiviert oder deaktiviert werden können – darunter die Suche, die voreingestellt eingeschaltet ist. Um die Suche zu deaktivieren, entfernen Sie bitte den Eintrag `:search` aus der Addon-Auflistung in dieser Datei.

Search Controller, Search Helper sowie mehrere Views, u. a. zur Darstellung des Suchformulars und der Trefferliste, werden vom RailsConnector bereitgestellt und sind im ausgelieferten Zustand bereits funktionsfähig.

Die Suche anpassen

Die mitgelieferten Views können Sie auf einfache Weise an Ihre Anforderungen anpassen.

Der Search Controller ist über die Konfigurationsdatei `rails_connector.rb` konfigurierbar. Darin können der Host und der Port des Search Servers sowie die zu durchsuchende Collection geändert werden:

```
RailsConnector::Configuration.search_options = {
  :host => 'custom_host',
  :port => 3011,
  :collection => 'cm-contents-de'
}
```

Sie können das Verhalten der Suche weiter anpassen, indem Sie in Ihrer Applikation einen eigenen Search Controller anlegen:

```
class SearchController < RailsConnector::DefaultSearchController
  # Ihre Anpassungen
end
```

Der Search Controller macht von Objekten der Klasse `SearchRequest` Gebrauch, um Suchanfragen auf Basis der eingegebenen Suchbegriffe zusammenzusetzen. Um steuern zu können, welche Dokumente gefunden werden dürfen und wie Benutzereingaben bereinigt und in Suchanfragen umgeformt werden, können Sie eine eigene `SearchRequest`-Klasse in Ihrer Applikation anlegen:

```
class SearchRequest < RailsConnector::DefaultSearchRequest

  def self.sanitize(text)
    # Ihre Implementierung
  end

  def base_query
    # Ihre Implementierung für die Zusammenfügung
    # der `base_query_conditions`
  end

  def base_query_conditions
    super.merge(
      :eigene_bedingung => 'VQL Code' #, ...
    )
  end
end
```

Welche Methoden Sie überschreiben, hängt davon ab, welche Aspekte Sie anpassen möchten – die obige Auswahl ist daher nur als Beispiel zu verstehen.

Die Helpers können über einen ähnlichen Mechanismus überschrieben werden. Legen Sie hierzu in Ihrer Applikation eine Datei unter dem Namen `app/helpers/search_helper.rb` mit folgendem Inhalt an:

```
module SearchHelper
  include RailsConnector::DefaultSearchHelper

  # Hier können Sie mitgelieferte Helpers überschreiben
  # und eigene anlegen.
end
```

2.22 Den PDF-Generator aktivieren

Bitte gehen Sie folgendermaßen vor, um den PDF-Generator des Infopark Rails Connectors zu aktivieren.

2.22.1 Apache FOP installieren

Der PDF-Generator nutzt den Apache FOP (Formatting Objects Processor), um PDF-Dateien zu erzeugen.

1. Wechseln Sie zunächst in das Verzeichnis, in dem der Apache FOP installiert werden soll, beispielsweise in `/opt/local`:

   ```
   $ cd /opt/local
   ```

2. Laden Sie nun den Apache FOP in der Version 0.94 von der offiziellen Website herunter:

   ```
   $ wget http://archive.apache.org/dist/xmlgraphics/fop/binaries/fop-0.94-bin-jdk1.4.zip
   ```

   Der PDF-Generator ist mit neueren Versionen des Apache FOP nicht kompatibel.

3. Entpacken Sie das heruntergeladene Archiv:

   ```
   $ unzip fop-0.94-bin-jdk1.4.zip
   $ rm fop-0.94-bin-jdk1.4.zip
   ```

4. Wechseln Sie in das Installationsverzeichnis des FOP und machen Sie das Start-Skript ausführbar:

   ```
   $ cd fop-0.94
   $ chmod 755 fop
   ```

5. Der Apache FOP verbraucht bei größeren PDF-Dateien mehr Speicher als er sich initial zuweist. Damit es nicht zu Speichermangel kommt, empfehlen wir, den maximal zulässigen Speicherverbrauch zu erhöhen. Öffnen Sie hierzu die Datei `fop-0.94/fop` und ändern Sie die Zeile

   ```
   fop_exec_command="exec \"$JAVACMD\" $LOGCHOICE $LOGLEVEL -classpath (...)
   ```

   zu Folgendem:

   ```
   fop_exec_command="exec \"$JAVACMD\" -Xmx500M $LOGCHOICE $LOGLEVEL -classpath (...)
   ```

6. Fügen Sie den Pfad des FOP-Installationsverzeichnisses zu Ihren Suchpfaden in der Umgebungsvariablen `PATH` hinzu.

2.22.2 Tidy installieren

Der PDF-Generator verwendet Tidy, um invalides XML zu bereinigen. Installieren Sie daher zunächst Tidy. Die Dokumentation zu Tidy finden Sie auf der offiziellen Website, http://tidy.sourceforge.net.

Das außerdem benötigte Tidy-Gem ist bereits mit dem Rails Connector installiert worden, da beim Rails Connector eine Abhängigkeit von diesem Gem definiert wurde.

2.22.3 Den PDF-Generator aktivieren

Um den PDF-Generator zu aktivieren, kommentieren Sie die entsprechende auskommentierte Zeile in der Datei `RAILS_ROOT/config/initializers/rails_connector.rb` bitte ein:

```
  [...]
  :time_machine,
# :pdf_generator
)
```

2.23 Wartungsaufgaben

2.23.1 Sessions aus der Datenbank löschen

Wenn sie Rails so konfiguriert haben, dass Browser-Sessions in der Datenbank gespeichert werden, sollten diese regelmäßig entfernt werden, um die betreffende Tabelle nicht zu groß werden zu lassen und die Performance der Datenbank nicht zu beeinträchtigen.

Hierfür können Sie beispielsweise den folgenden Ruby-Code als Rake-Task in Ihrem Projekt verwenden. Dieser Task kann und sollte regelmäßig als Cron-Job ausgeführt werden.

```
namespace :db do
  namespace :sessions do
    desc "Prune database-stored sessions older than one week"
    task :prune => :environment do
      CGI::Session::ActiveRecordStore::Session.delete_all ["updated_at < ?", 1.week.ago ]
    end

    desc "Count database sessions"
    task :count => :environment do
      puts "Currently storing #{CGI::Session::ActiveRecordStore::Session.count} sessions"
    end
  end
end
```

2.24 Den Rails Connector anpassen

Der Rails Connector enthält Komponenten, die bereits im Auslieferungszustand in Ihrer Rails-Anwendung lauffähig sind. Darüber hinaus ist die Software mit vielfältigen Schnittstellen ausgestattet, über die Sie die Komponenten an Ihre Anforderungen anpassen können.

Die folgenden Arten von Bestandteilen sind prinzipiell erweiter- und austauschbar:

- Controller (z. B. `CmsController`, `SearchController`)

- Helper (z. B. `CmsHelper`, `SearchHelper`)
- Views (z. B. für die Darstellung von Suchergebnissen)
- Das Modell `Obj`

2.24.1 Controller

Die Controller-Klassen des Rails Connectors sind über einen Vererbungsmechanismus erweiterbar. Die Klasse `CmsController`, beispielsweise, ist von der Klasse `RailsConnector::DefaultCmsController` abgeleitet und erbt dadurch das gesamte dort definierte Verhalten. Die mitgelieferte Klasse `CmsController` fügt darüber hinaus keine Funktionalität hinzu, da sie als Platzhalter für eigene Erweiterungen der Standard-Implementierung gedacht ist.

Um das Verhalten des CMS-Controllers anzupassen, legen Sie in Ihrem Applikationsverzeichnis unter `app/controllers/` eine Datei `cms_controller.rb` mit folgendem Inhalt an:

```
class CmsController < RailsConnector::DefaultCmsController
  # Ihre Anpassungen
end
```

2.24.2 Helper

Die Helper-Module des Rails Connectors sind hierarchisch genauso aufgebaut wie die Controller-Klassen: es gibt beispielsweise ein Modul `DefaultCmsHelper`, das die eigentliche Implementierung enthält, sowie ein Modul `CmsHelper`, das diese Funktionalität nutzt, jedoch selbst nichts hinzufügt. Um – für dieses Beispiel – Helper-Methoden hinzuzufügen oder vorhandene auszutauschen, legen Sie in Ihrem Applikationsverzeichnis unter `app/helpers/` eine Datei `cms_helper.rb` mit folgendem Inhalt an:

```
module CmsHelper
  include RailsConnector::DefaultCmsHelper

  # Ihre Anpassungen
end
```

2.24.3 Views

Die mitgelieferten Views des Rails Connectors können Sie durch eigene ersetzen; siehe den Abschnitt Mitgelieferte Views des Rails Connectors anpassen.

2.24.4 Das Modell `Obj`

Der Rails Connector sucht bei der Initialisierung innerhalb Ihrer Applikation nach einem Modul namens `ObjExtensions` und ruft, falls es existiert, dessen Methode `enable` auf. Diesen Mechanismus können Sie nutzen, um das Modell `Obj` zu erweitern. Legen Sie hierfür die Datei `obj_extensions.rb` in Ihrer Applikation an und geben Sie ihr als Ausgangsbasis den folgenden Inhalt:

```
module ObjExtensions
  def self.enable
```

```
    Obj.class_eval do
      # Definieren Sie hier Ihre Methoden
    end
  end
end
```

2.25 Empfohlene Programmierpraktiken

2.26 Link-Verwaltung des CMS in Projekten mit Rails Connector nutzen

Infopark CMS Fiona hat eine interne Link-Verwaltung, die vor allem sicherstellt, dass beim Verschieben oder Umbenennen von CMS-Dateien die Referenzen automatisch aktualisiert werden. Der Rails Connector verfügt jedoch nicht über die geänderten Pfad-Informationen. Aus diesem Grund sollten Aufrufe wie `Obj.find(1234)` oder `Obj.find_by_path("/company/news/")` in einer Rails-Anwendung unbedingt vermieden werden.

Um dennoch direkt auf bestimmte CMS-Dateien zugreifen zu können, stellt der Rails Connector mit benannten Links eine geeignete Möglichkeit zur Verfügung: Erstellen Sie im CMS eine Datei mit der Vorlage `NamedLink` und dem Linklistenfeld `related_links`. Über diese Linkliste können Sie nun im CMS Ihre Referenzen pflegen, die dann auch von der Link-Verwaltung erfasst werden. In der Rails-Anwendung können Sie nun auf die referenzierten Dateien über den jeweils in der Linkliste vergebenen Titel zugreifen (Beispiel: `NamedLink.get_object("news")`). Darüber hinaus werden die benannten Links im Cache gehalten, wodurch beim Zugriff darauf Ressourcen geschont werden.

Typische Anwendungsfälle für benannte Links sind beispielsweise:

- Auf der Homepage die im News-Ordner enthaltenen Dateien anzeigen:
  `NamedLink.get_object("news").each do ...`
- Für eine bestimmte Seite eine bestimmte Aktion ausführen:
  `if @obj == NamedLink.get_object("homepage") ...`
- Von einer bestimmten Datei ein Feld ermitteln:
  `NamedLink.get_object("important_notes").body`

Weitere Informationen hierzu finden Sie in der API-Dokumentation zur Klasse `NamedLink`.

Bitte beachten Sie folgende Hinweise bei der Nutzung benannter Links:

Im CMS darf nur eine Datei mit der Vorlage `NamedLink` existieren. Andernfalls kommt es zu Fehlern in der Rails-Anwendung, die dazu führen können, dass die Website nicht mehr verfügbar ist. Um zu verhindern, dass ein Redakteur versehentlich weitere Dateien mit der Vorlage `NamedLink` anlegt, sollte die Vorlage deaktiviert werden, nachdem die Datei angelegt wurde.

Darüber hinaus dürfen die CMS-Dateien, auf die die benannten Links zeigen, weder deaktiviert noch zurückgezogen werden. Versucht die Rails-Applikation, mittels `NamedLink` auf zurückgezogene oder deaktivierte Dateien zuzugreifen, wird eine Exception ausgelöst. Wird diese Exception nicht explizit abgefangen, funktionieren die Webseiten nicht mehr, die diesen benannten Link enthalten. Abhängig davon, wo benannte Links auf nicht vorhandene Ziele verwendet werden, kann die gesamte Website ausfallen.

2.27 Rails-Projekt an lokale Entwicklungsumgebung anpassen

Das Rails-Projekt kann in den Dateien `config/environment.rb`, `config/environments/*.rb` und `config/initializers/rails_connector.rb` konfiguriert werden.

Die dort angegebene Konfiguration wird ab Version 6.7.3 durch Ruby-Dateien in `config/local/` überschrieben. Dadurch kann das Rails-Projekt an die lokale Entwicklungsumgebung anpasst werden, ohne dass die Projekt-Konfiguration geändert werden muss.

In diesen Dateien kann sowohl die Konfiguration von Rails als auch die des Rails Connectors lokal angepasst werden. Beispiel:

```
config.action_mailer.delivery_method = :my_local_delivery_method

RailsConnector::Configuration.instance_name = "my_local_instance_name"
```

Die Installation des Rails Connectors hinterlässt eine leere Datei `config/local/configuration.rb`, d.h. es werden keine Änderungen an der Projekt-Konfiguration vorgenommen.

Wenn für die Entwicklung einer Rails-Anwendung ein Versionierungssystem verwendet wird, sollte es so konfiguriert sein, dass es die Dateien im Verzeichnis `config/local` ignoriert.

2.28 Tipps und Tricks

2.29 Datum, Uhrzeit und Zeitzonen

Der Rails Connector konvertiert Datumswerte aus dem CMS automatisch in die Zeitzone, die in der Rails-Applikation gilt. Hierfür verwendet er die Ruby-Klasse `Time`. In der Rails-Applikation kann die eingestellte Zeitzone folgendermaßen ermittelt werden:

```
$ rails console
Loading development environment (Rails 3.0.5)
>> Time.now.zone
=> "CET"
```

Siehe auch die Rails-Dokumentation zu Zeitzonen unter http://apidock.com/rails/ActiveSupport/TimeWithZone.

2.30 API-Dokumentation abrufen

Die API-Dokumentation zum Rails Connector wird bei der Installation des Rails Connectors mittels RDoc generiert. Um auf die Dokumentation zuzugreifen, starten Sie bitte zunächst den RDoc-Server:

```
gem server
```

Anschließend können Sie die Dokumentation aller installierten Gems mit einem Web-Browser auf `http://0.0.0.0:8808/` lesen.