Dennis Blöte

Eventkalender mit Ruby on Rails

RESTful API und Plugins für Exportformate im Praxiseinsatz

wevent.org ist ein auf Ruby on Rails basierender Eventkalender in dem Nutzer Veranstaltungen eintragen, Freunde einladen und sich über Events in ihrer Nähe informieren können. Um eine Schnittstelle für externe Anwendungen (bspw. Desktop-Kalender oder Feedreader) zu bieten, wurde eine RESTful API [1] realisiert, über welche verschiedene Datenformate angefordert werden können. Dieser Artikel erläutert die für diese Zwecke eingesetzten Plugins.

Keep it RESTful

Wer seine Anwendung RESTful entwickelt (siehe Serie in T3N Magazin Nr. 7 und 8) bietet eine uniforme Schnittstelle über welche die eigenen Daten (Resourcen) anderen Anwendungen zur Verfügung gestellt werden können. Bei einem web-basierten Eventplaner bietet es sich an, dass die Nutzer ihren Veranstaltungskalender auch offline auf dem Desktop nutzen oder sich per RSS über neu eingetragene Events informieren lassen können.

Die Ausgabe solcher Daten für das Frontend (sprich als HTML) ist oftmals bereits in einer Action implementiert, es muss also lediglich auf das vom Client angeforderte Format reagiert werden. Kalenderapplikationen fragen das Standardformat iCal (meine_events.ics) an, für Feedreader kann eine Repräsentation als RSS oder Atom angeboten werden. Wem das noch nicht reicht, der stellt für weitere Anwendungen (z.B. Web Services) auch noch XML, YAML oder JSON zu Verfügung – dank Rails mit wenigen Zeilen Code.

respond_to whatever

Mit respond_to lässt sich auf den vom Client gesendeten Accept-Header oder das explizit angeforderte Datenformat reagieren. Erhält Rails bspw. eine Anfrage deren HTTP-Header `Accept: application/xml` enthält oder auf http://wevent.org/events.xml zugegriffen wird, so rendert die Action index eine Liste aller Events als XML.

RUBY ON RAILS - Controller

```
class EventsController < ApplicationController
  def index
    @page_title = „Alle Events“
    @events = Event.find(:all)

    respond_to do |format|
      format.html
      format.xml  { render_xml_for  @events }
      format.json { render_json_for @events }
      format.yaml { render_yaml_for @events }
      format.rss  { render_rss_for  @events, :title => @page_title }
      format.atom { render_atom_for @events, :title => @page_title }
      format.ics  { render_ical_for @events, :title => @page_title }
    end
  end
end
```

Die Action index gibt eine Liste aller Events aus

Innerhalb des `respond_to` Blocks werden die möglichen Ausgabeformate aufgelistet – die Reihenfolge spielt dabei keine Rolle, da der Client die Reihenfolge seiner gewünschten Formate im Accect-Header festlegt (oder das Format durch die Dateiendung explizit angegeben wurde).

Wer schon einmal eine XML-Ausgabe seiner Modelldaten implementiert hat, dem wird auffallen, dass die im Listing gezeigte Implementation nicht zu den Rails-Bordmitteln gehört. Standardmäßig würde das mit folgendem Befehl realisiert werden:

RUBY ON RAILS - Controller

```
format.xml  { render :xml => @events.to_xml }
```

Üblicher Aufruf zum Rendern des XML eines Objekts

Um einen einheitlichen Aufruf der render-Methoden zu haben, implementiere ich diese als Methoden des ApplicationControllers (Listing folgt später). Dieses Vorgehen macht außerdem Sinn, um den Aufruf des RSS-, Atom- und iCal-Renderings zu wrappen, wie wir im folgenden sehen werden.

RSS- und Atom-Export in zehn Minuten

Der Rails 1.2 Core bringt kein RSS- oder Atom-Rendering von Objekten wie bspw. `@events.to_rss` mit sich. Solche Anforderungen lassen sich aber Rails-typisch mit einem Plugin lösen: Der Resource Feeder [2] macht es einem sehr leicht, RSS- und Atom-Feeds aus Modelldaten zu generieren.

Zunächst müssen die beiden Plugins Simply Helpful und der darauf basierende Resource Feeder installiert werden:

Befehlseingabe über die Konsole

```
$ script/plugin install simply_helpful
$ script/plugin install resource_feeder
```

Plugin-Installation von Simply Helpful und dem Resource Feeder

Für die Implementation der Ausgabe in den verschiedenen Formaten stellt das Plugin im Controller die beiden Methoden `render_rss_feed_for` und `render_atom_feed_for` zur Verfügung, welche zwei Parameter entgegennehmen:

1. Ein Array mit Objekten, zu denen jeweils ein Item im Feed erzeugt werden soll (in unserem Fall ein Eintrag pro Event).
2. Optional: Ein Hash mit Optionen, welcher unter anderem das Mapping der Objektattribute auf die Feeditem-Elemente angibt.

RUBY ON RAILS – Options-Hash für das Feed-Rendering

```
options[:feed][:title]          # Titeldes Feeds
options[:feed][:link]           # Link zum Feed
options[:feed][:description] # Beschreibung des Feeds
options[:feed][:language]    # Sprache, default: "en-us"
options[:feed][:ttl]            # Time to live, Minuten die das Feed gecacht werden kann, default: 40
options[:item][:title]          # Methode die den Titel des Items zurückgibt
options[:item][:description] # Methode die den Inhalt des Items in plain text zurückgibt
options[:item][:content_encoded] # Optional, encodierter Inhalt (mit HTML o.ä.)
options[:item][:pub_date]   # Methode die das Datum des Eintrags zurückgibt, default: created_at
options[:url_writer]            # Optional, eigener UrlWriter, nützlich bei verschachtelten Resourcen
```

Liste aller Optionen die `render_rss_feed_for` entgegennimmt.

Die auf die Items bezogenen Optionen geben jeweils den Namen der Objektmethode an, welche das Veröffentlichungsdatum, den Titel oder die Beschreibung zurückgeben. Im Idealfall können diese Angaben entfallen, Vorraussetzung dafür ist, dass das Modell schon über die richtigen Attribute (z.B. `created_at`, `title`, `description`) verfügt – ansonsten muss man, wie im folgenden Listing zu sehen, noch etwas nachkonfigurieren.

Da die meisten dieser Optionen durch die Anwendung hin gleich bleiben (bspw. Language und Time to live), kann man die Methoden zum Rendern der Formate wrappen, um Wiederholungen in den einzelnen Aufrufen zu vermeiden (stay DRY!):

RUBY ON RAILS - Controller

```
class ApplicationController < ActionController::Base
  protected
    def render_xml_for(objects, options = {})
      render :xml => objects.to_xml(options)
    end
    def render_json_for(objects)
      render :json => objects.to_json
    end
    def render_atom_for(objects, options = {})
      render_atom_feed_for objects, {
        :feed => { :title => options[:title] },
        :item => {
          # Die Methode summary des Eventobjekts gibt den Titel zurück,
          # description_for_feed die Beschreibung, welche im Feed auftauchen soll
          :title => :summary,
          :description => :description_for_feed } }
    end
    def render_rss_for(objects,options = {})
      render_atom_feed_for objects, {
        :feed => { :title => options[:title], :language => "de-DE", :ttl => 60 },
        :item => {
          :title => :summary,
          :description => :description_for_feed,
          :content_encoded => :content_encoded } }
    end
    def render_ical_for(objects, options = {})
      ical = Icalendar::Calendar.new # Erstellen eines neuen Kalenderobjekts
      # Einstellungen für den Kalender
      ical.product_id = "-//wevent.org//iCal 1.0//DE"
      ical.custom_property("X-WR-CALNAME;VALUE=TEXT", "wevent.org: #{options[:title]}")
      ical.custom_property("X-WR-TIMEZONE;VALUE=TEXT", "Europe/Berlin")
      # Eventobjekte als iCalendar-Daten rendern und dem Kalender hinzufügen
      ical.add(objects.to_ical)
      render_without_layout :text => ical.to_ical # Ausgabe der fertigen iCalendar-Datei
    end
  end
end
```

Wrapper für das Rendern der Formate XML, JSON, ATOM, RSS und iCal

Das Listing enthält auch schon das Rendern des Formats iCalendar, welches wir im nächsten Schritt implementieren.

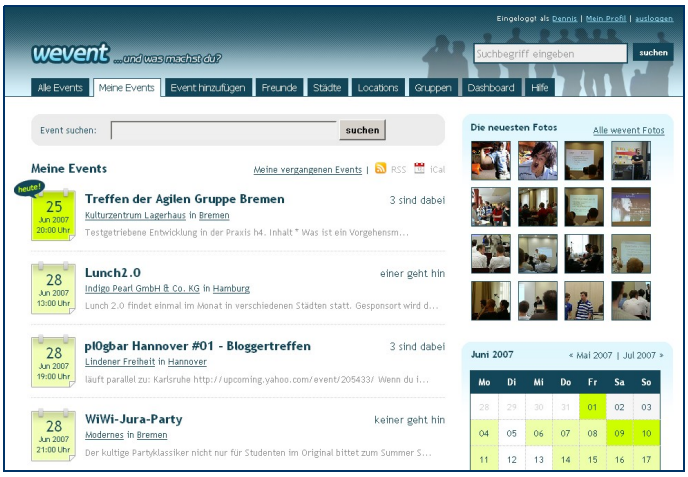
Export von Daten als iCalendar

iCalendar ist ein Standardformat zum Austausch von Kalenderdaten, welches von allen verbreiteten Kalenderapplikationen unterstützt wird. Der Export des Formats (Dateiendung .ics) gehört zwar nicht zu den Rails-Bordmitteln, in diesem Fall schafft aber die Library iCalender Abhilfe, welche sich als Ruby Gem installieren lässt.

Befehlseingabe über die Konsole

```
$ sudo gem install icalendar
```

Installation des iCalendar Gems



Der Zugriff auf Applikationsdaten kann nicht nur über die Weboberfläche, sondern auch über weitere Schnittstellen (API) erfolgen.

Die Library ermöglicht uns das Erstellen eines Kalenderobjekts, welches zunächst mit den Daten der enthaltenen Veranstaltungen befüllt werden muss. Dazu implementieren wir die Methode `to_ical` in unserem Eventmodell, welche ein Eventobjekt im iCalendar-Format zurückgibt.

RUBY ON RAILS – Modell

```
require 'icalendar' # Einbinden der iCalendar-Library
class Event < ActiveRecord::Base
  def to_ical
    ical_event = Icalendar::Event.new # Erstellen des Eventobjekts im iCalendar-Format
    # Befüllen des Veranstaltungseintrags
    ical_event.summary = summary       # Titel der Veranstaltung
    ical_event.description = description  # Beschreibung der Veranstaltung
    ical_event.dtstamp = created_at.iso8601    # Zeitstempel des Eintrags im ISO-Format
    ical_event.dtstart = datetime_start.iso8601  # Zeitstempel des Beginns
    ical_event.dtend =  datetime_end.iso8601  # Zeitstempel des Endes
    ical_event.uid = „wevent:event=#{id}“ # Eindeutige ID des Events
    ical_event.location = "#{venue_name}, #{city_name}" # Veranstaltungsort
    ical_event.to_ical # gibt das iCal Objekt zurück
  end
end
```

Diese Methode kann nun im Controller genutzt werden, um ein Array mit Eventobjekten als kompletten Kalender auszugeben: Der Aufruf von `to_ical` auf einem Array sorgt dafür, dass durch das Array iteriert und auf jedem Element die Methode `to_ical` ausgeführt wird. Ihr Rückgabewert (der Kalendereintrag des einzelnen Events) wird anschließend dem Kalender hinzugefügt (`ical.add(objects.to_ical)`, welcher letztendlich gerendert wird.

Optimierungsmöglichkeiten

Der Einfachheit halber haben wir in diesem Tutorial immer alle Events aus der Datenbank geladen und sie anschließen ausgegeben. Dies macht in der Praxis natürlich aus mehreren Gründen wenig Sinn. Zum einen wäre die Last auf der Datenbank enorm groß, wenn bei jedem Aufruf der API alle Daten geladen werden würde und zum anderen erzeugt solch eine große Datenmenge auch beim Client zu viel Last, da die Daten vor der Verarbeitung geparst werden müssen.

Wie umgehen wir das? Genau so, wie wir es aus der Weboberfläche gewohnt sind: Partitionierung/Paginierung der Daten. Der Client erhält bei jedem Aufruf nur eine Untermenge der Daten (bspw. 15 Events) und kann durch die Übergabe eines Parameters den nächsten Teil der Daten anfordern (zum Beispiel über http://wevent.org/events.xml?page=2).

Pagination ist in der aktuellen Rails-Version 1.2 noch enthalten, mit Rails 2.0 wird das Konzept allerdings auf Plugins ausgelagert. Es empfiehlt sich daher, sich nach einer Plugin-Lösung des Problems umzusehen – meine Empfehlung an dieser Stelle wäre das Plugin will_paginate [4], welches sich momentan als präferierte Lösung unter Rails-Entwicklern abzeichnet.

Fazit

Wie dieses Tutorial zeigt, erlaubt Ruby on Rails es einem, seine Anwendung nach außen hin öffnen und so weitere Nutzungsmöglichkeiten der Daten für die Nutzer zur Verfügung zu stellen. APIs werden in Zukunft eine immer größere Rolle spielen und besonders die Verbreitung von REST-Schnittstellen wird zunehmen. Ruby on Rails hat das Konzept REST bereits fest integriert und wird auch dadurch weiter an Bedeutung gewinnen.

Links und Literatur

[1] Dokumentation der REST API von wevent.org:
http://wevent.org/api

[2] Resource Feeder Plugin:
http://dev.rubyonrails.org/browser/plugins/resource_feeder

[3] Dokumentation des iCalendar Gems:
http://icalendar.rubyforge.org/

[4] will_paginate Plugin:
http://errtheblog.com/post/4791

DER AUTOR

Dennis Blöte ist freiberuflicher Webentwickler und Mediengestalter aus Bremen. Sein Tätigkeitsschwerpunkt liegt in der Erstellung von Webapplikationen mittels XHTML und CSS, Ruby on Rails, JavaScript und AJAX.

Aktuell entwickelt er zusammen mit Sören Weber die im Artikel erwähnte Plattform wevent (http://wevent.org) und schreibt in seinem Blog (http://blog.dopefreshtight.de) über die Entwicklung mit Ruby on Rails anhand von Beispielen aus der Praxis.