

inkl. **CD!**

Was war auf der Mobile TechCon in München los? S. 10

S&S

Deutschland € 6,50 Österreich € 7,00 Schweiz sFr 13,40

entwickler
magazin

entwickler

www.entwickler-magazin.de

magazin

CD-INHALT:



■ **Wapiti 2.2.1:** Ein Web-Application-Vulnerability-Scanner. Er überprüft mit Black-box-Scans die Sicherheit der Anwendung.

■ **skipfish 1.85 beta:** Skipfish ist ein Schwachstellenscanner für Webanwendungen. Er testet Applikationen auf verschiedene Sicherheitslücken.

■ **MongoDB 1.8:** MongoDB ist eine skalierbare, leistungsfähige und dokumentenorientierte NoSQL-Datenbank.

■ **Firebird 2.5.0:** Zu den Neuerungen gehören Auditing-Möglichkeiten und verbesserte Skalierbarkeit.

■ **IT Survival Guide: Karriere- und Alltagsratgeber für Einsteiger und Professionals in der IT-Branche von Yasmine Limberger:** Buchauszug – entwickler.press

■ **Video von der BASTA! Spring 2011:** Microsoft 2011 – ein Statusbericht

Bonus

Zusätzlich finden Sie auf unserer aktuellen Magazin-CD ausgewählte Tools sowie alle Quellcodes und Beispiele zu den Artikeln im Heft.

►►► Alle Infos auf Seite 3

Weitere Artikel

Lizenzverwaltung, Updates und Vertrieb
PHP App Store – Wie es geht

Grundlagen und Implementierung
Algorithmen

Web Services Security als Standard
SOA Security in der Praxis

Design Patterns programmieren
Iteratoren mit Delphi

Webarchitekturen mit JavaScript

RIAS mit ExtJS und Jersey



Datenzugriff per Show- und List-Funktion CouchDB-Daten

Sicher genug für die Webentwicklung?

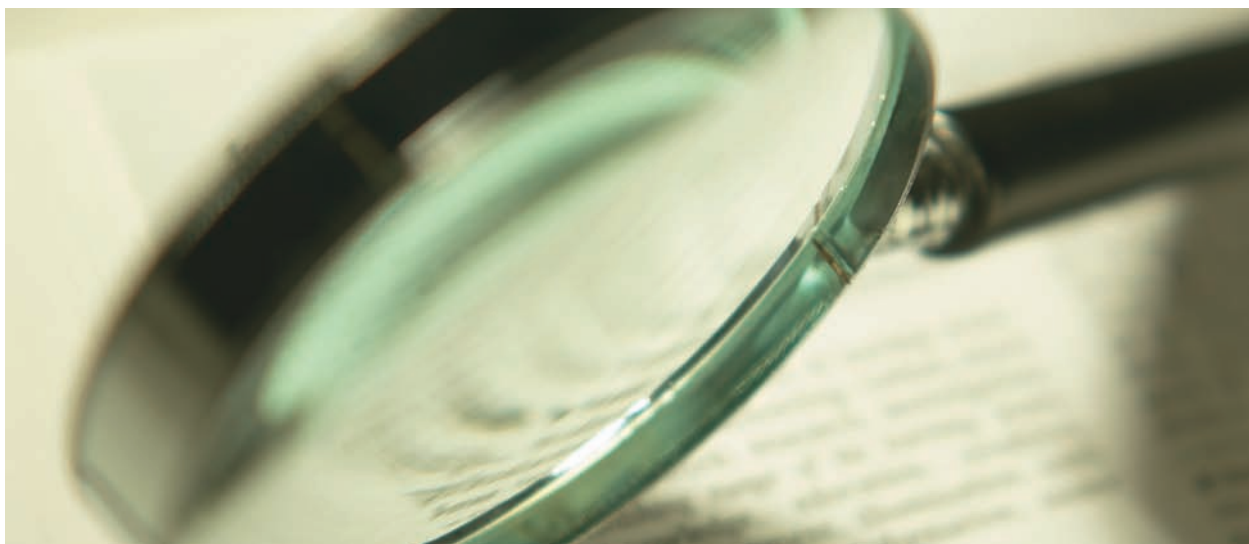
Der neue Personalausweis



jQuery mobile Das Web im Smartphone



Datenträger enthält
Info- und
Lehrprogramme
gemäß §14 JuSchG



...oder wie Abfrageergebnisse in CouchDB darstellbar sind

CouchDB – Show- und List-Funktionen

Relax again! CouchDB liegt in der Version 1.0.2 vor. Aus CouchIO wurde CouchOne und jetzt CouchBase, nachdem eine Fusion bzw. Symbiose mit der Firma Membase stattgefunden hat. Der eine steigt nicht mehr durch das, was alles geschieht, den anderen interessiert es am Ende gar nicht. Denn eines bleibt gleich: CouchDB ist ein exzellenter Vertreter unter den NoSQL-Datenbanken und bleibt als Apache-Projekt Open Source. In diesem Artikel lesen Sie, was Show- und List-Funktionen sind.

von Andreas Wenk

In der Ausgabe 1.2011 des Entwickler Magazins [1] haben Sie bereits lesen können, wie Daten per Map/Reduce-Methodik aus einer CouchDB-Instanz gelesen werden können. Dazu werden Views erstellt. Diese Views wiederum werden in Designdokumenten vorgehalten und gespeichert. Nachdem die Sicht auf die Daten implementiert ist, geht es in einem weiteren Schritt darum, wie diese Daten dargestellt werden. CouchDB liefert hierfür zwei weitere Bausteine. Zum einen können mit Show-Funktionen die Daten eines Dokuments, zum anderen mit List-Funktionen die Daten einer View ausgegeben werden – beide in unterschiedlichen Formaten. Sehen wir uns beide Varianten nacheinander an. Vorerst erstellen wir eine Datenbank *kassenbuch*, füllen sie mit

Daten und erstellen eine View (Kasten „Die *kassenbuch*-Datenbank im Netz“).

Konzept

Bevor wir loslegen, ist es sinnvoll, sich einen einfachen Use Case auszudenken, anhand dessen die Funktionalität der einzelnen Methoden gezeigt werden kann. Los gehts:

- In einer *kassenbuch*-Datenbank soll der Benutzer Buchungen erfassen können.
- Die Buchungen sollen die Attribute *typ*, *datum*, *muust*, *betrag* und *beschreibung* haben.
- Es soll möglich sein, die Buchungen einzeln oder nach *typ*, *muust* oder *datum* zu sortieren.
- Es soll möglich sein, einzelne Buchungen im Webbrowser anzusehen.

- Es soll möglich sein, eine Tabelle aller Buchungen im Webbrowser darzustellen.

Zusätzlich zu den aufgezählten Punkten kommt noch, dass diese Aufgabe nur mit CouchDB erfüllt werden soll.

Die „kassenbuch“-Datenbank

Wie immer werde ich das Kommandozeilenprogramm *curl* [2] nutzen, um mit der CouchDB-Instanz [3] zu sprechen. Es steht Ihnen natürlich frei, ein anderes HTTP-Tool zu nutzen. Sie können die Beispiele problemlos mit Futon [4] – dem CouchDB-eigenen Webinterface – nachbauen. Im ersten Schritt erstellen wir also die Datenbank *kassenbuch*:

```
$ curl -X PUT http://localhost:5984/kassenbuch
HTTP/1.1 201 Created
{"ok":true}
```

Das Dollarzeichen dient hier als Platzhalter für die Kommandozeile. Danach folgt der Aufruf des Programms *curl* mit dem Parameter *X*, gefolgt von der HTTP-Methode *PUT*. Der Parameter *X* dient dazu, *curl* mitzuteilen, dass eine andere Methode als die Standardmethode *GET* für den nachfolgenden URI (Uniform Resource Identifier) genutzt werden soll. In der nächsten Zeile habe ich einen Auszug aus dem Response mit angegeben. Zum einen den zurückgelieferten Statuscode – *HTTP/1.1 201 Created* – zum anderen die Meldung von CouchDB – *{ "ok":true }*. Sie erhalten eine ausführliche Response-Header-Ausgabe, wenn sie den Parameter *i* nutzen. Damit ist der erste Punkt des Konzepts abgeschlossen.

Das kassenbuch mit Daten füllen

Die *kassenbuch*-Datenbank ist erstellt und will nun mit Daten bzw. Buchungen oder Dokumenten gefüllt werden (Listing 1). Mit diesem PUT-Request haben wir ein Dokument mit der ID *b0001* erstellt. Mitgeschickt haben wir einen JSON-String mit einigen Daten, wie oben schon im Konzept beschrieben. Sie sollten mehrere solcher Dokumente erstellen, natürlich mit einer anderen ID. Wenn zehn Dokumente mit den Typen *Einnahme* oder *Ausgabe*, den MwSt.-Sätzen 19 % und 7 % und jeweils unterschiedlichem Datum erstellt sind, hat man später eine recht nette Ausgabe. Damit ist der zweite Punkt des Konzepts abgeschlossen.

Sicht auf die kassenbuch-Daten

Im Konzept gibt es die Anforderung, die Daten sortiert ausgeben zu können. Hierfür wird eine View erstellt. Sie besteht aus einer Map- und aus einer Reduce-Funktion. CouchDB bietet die Möglichkeit, Daten bei der Abfrage einer View zu gruppieren. Dafür wird der Parameter *group_level* mit der Angabe des Levels an den URI einer View mitgegeben. Sehen wir uns die Map-Funktion an:

```
function(doc) {
  emit([doc.typ, doc.mwst, doc.datum], doc.betrag);
}
```

Die CouchDB-eigene Funktion *emit* erwartet zwei Parameter: einen *key* und einen *value*. Um die *group_level*-Funktionalität nutzen zu können, wird in der obigen Map-Funktion ein Array aus Attributen des Dokuments übergeben. Die Reihenfolge spielt eine wichtige Rolle. Denn es gilt:

- *group_level=0*: die Daten werden nicht gruppiert
- *group_level=1*: die Daten werden nach Typ gruppiert
- *group_level=2*: die Daten werden zuerst nach Typ und dann nach MwSt.-Satz gruppiert
- *group_level=3*: die Daten werden zuerst nach Typ, dann nach MwSt.-Satz und schließlich nach Datum gruppiert

Sie können das *group_level* beliebig erweitern, was allerdings ab einem bestimmten Level nicht mehr sinnvoll ist. Die dazugehörige *Reduce*-Funktion ist denkbar einfach: *_sum*. Die Funktion *_sum* ist eine CouchDB-eigene Funktion und tut das, was sie sagt: sie summiert Werte. Wie das Designdokument erstellt wird, ist in Listing 2 zu sehen. Danach kann wie in Listing 3 ein Request abgesetzt werden. Wir haben kein *group_level* angegeben

Listing 1

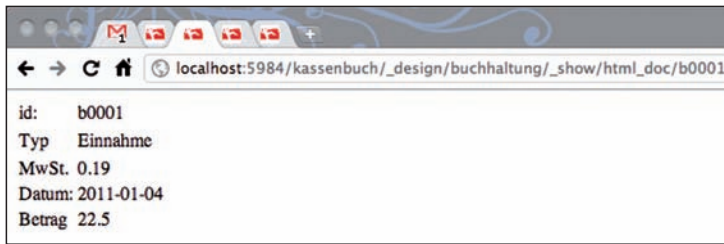
```
$ curl -X PUT http://localhost:5984/kassenbuch/b0001 \
-d '{"typ":"Einnahme","betrag":22.5,"mwst":0.19, \
  "beschreibung":"Rechnung 42","datum":"2011-01-04"}'
HTTP/1.1 201 Created
{"ok":true,"id":"b0001","rev":"1-d160b673cae0174d7bf2c1b32785d22e"}
```

Listing 2

```
curl -X PUT http://localhost:5984/kassenbuch/_design/buchhaltung \
-d '{"views":{"auswertung":{"map":"function(doc) \
{emit([doc.typ,doc.mwst,doc.datum], doc.betrag);},"reduce":"_sum"}}}'
HTTP/1.1 201 Created
{"ok":true,"id":"_design/buchhaltung", \
  "rev":"1-7c97c20a924e489e02627da13cae98b6"}
```

Die „kassenbuch“-Datenbank im Netz

Ich bin davon überzeugt, dass man nur nachhaltig lernen kann, wenn man Beispiele selbst nachbaut. Das gilt besonders für Programmcode. Hier handelt es sich zwar weniger um typischen Programmcode, aber der Umgang und die Kommunikation mit der CouchDB über das RESTful-HTTP-Interface erfordern einige Übung. Deshalb empfehle ich, den URI und die Funktionen selbst zu tippen. Trotzdem sollten Sie die Möglichkeit haben, sich das Endprodukt live ansehen zu können. Das können Sie tun, indem Sie in die Adresszeile eines Browsers diesen URL eingeben: http://couchbuch.couchone.com/_utils/database.html?kassenbuch/_all_docs



id:	b0001
Typ	Einnahme
MwSt.	0.19
Datum:	2011-01-04
Betrag	22.5

Abb. 1: HTML-Tabelle

und erhalten deshalb nur einen Ergebniswert der *reduce*-Funktion. Das Ergebnis für die Gruppierung nach *typ* (Eingaben und Ausgaben jeweils gruppiert) könnte wie in Listing 4 aussehen. Spielen Sie zur Probe mit den weiteren *group_level*-Eingaben. Damit ist der dritte Punkt des Konzepts abgeschlossen.

Listing 3

```
$ curl -X GET http://localhost:5985/kassenbuch2/_design/buchhaltung/_view/auswertung
HTTP/1.1 200 OK
{"rows": [
  {"key": null, "value": 3065.74}
]}
```

Listing 4

```
$ curl -iX GET http://localhost:5985/kassenbuch2/_design/buchhaltung/_view/auswertung?group_level=1
HTTP/1.1 200 OK
{"rows": [
  {"key": ["Ausgabe"], "value": 253.10000000000002},
  {"key": ["Einnahme"], "value": 2812.64}
]}
```

Listing 5

```
function(doc, req){
  return '<table><tr><td>id:</td><td>' + doc._id + '</td></tr> \
<tr><td>Typ</td><td>' + doc.typ + '</td></tr><tr> \
<tr><td>MwSt.</td><td>' + doc.mwst + '</td></tr> \
<tr><td>Datum:</td><td>' + doc.datum + '</td></tr> \
<tr><td>Betrag</td><td>' + doc.betrag + '</td></tr></table>'
}
```

Listing 6

```
function(doc, req) {
  return {
    body: '<docdata><docid>' + doc.title + '</docid> \
<typ>' + doc.typ + '</typ></docdata>',
    headers: {
      "Content-Type": "application/xml"
    }
  }
}
```

Listing 7

```
{
  "_id": "_design/buchhaltung",
```

Die Show-Funktion

Bis hierher wurden Eigenschaften der CouchDB besprochen, die im Artikel der vorigen Ausgabe [1] nachgelesen werden können. Neu dabei ist die Methodik von *group_level*, wobei dies einen essenziellen Punkt von CouchDB darstellt. Kommen wir nun zum Artikelschwerpunkt – die *_show*-Funktion. CouchDB liefert von Haus aus die Möglichkeit, die Daten eines Dokuments formatiert auszugeben. Dazu werden *_show*- und *_list*-Funktionen genutzt (Kasten „Wenn der URL zu lang wird“). Das Ziel ist es, die Werte eines Dokuments in HTML darzustellen. Dafür wird eine einfache Tabelle ausgewählt. Die Show-Funktionen (*value*) werden im Designdokument unter *shows* gespeichert, wobei *n* Funktionen jeweils unter einem anderen Namen (*key*) abgelegt werden können. Bevor das getan wird, betrachten wir die Show-Funktion in Listing 5.

```
"_rev": "2-aa25aee709cebcf6350da7801cfa4647",
"views": {
  "auswertung": {
    "map": "function(doc) {emit([doc.typ, doc.mwst, doc.datum], \
      doc.betrag);}",
    "reduce": "_sum"
  }
},
"shows": {
  "html_doc": "function(doc, req){ \
    return '<table><tr><td>id:</td><td>' + doc._id + '</td> \
      <tr><td>Typ</td><td>' + doc.typ + '</td></tr> \
      <tr><td>MwSt.</td><td>' + doc.mwst + '</td></tr> \
      <tr><td>Datum:</td><td>' + doc.datum + '</td></tr> \
      <tr><td>Betrag</td><td>' + doc.betrag + '</td></tr> \
      </table>'"
  }
}
```

Listing 8

```
function(head, req) {
  var row;
  start({
    'headers': {'Content-Type': 'text/html'}
  });
  send('<table>
    <tr>
    <td>ID</td>
    <td>Typ</td>
    <td>MwSt.</td>
    <td>Datum</td>
    <td>Wert</td>
    </tr>');
  while(row = getRow()){
    send('<tr><td>' + row.id + '</td>
      <td>' + row.key[0] + '</td>
      <td>' + row.key[1] + '</td>
      <td>' + row.key[2] + '</td>
      <td>' + row.value + '</td>');
  }
  send('</table>');
```


Der Code ist nicht gerade das, was man an HTML öfter schreiben möchte. Aber für unsere Zwecke reicht das völlig aus. Eine Show-Funktion erwartet zwei Parameter; zum einen das Dokument und zum anderen den Request. Mittels des zweiten Parameters *req* besteht die Möglichkeit auf GET-Parameter des Request-URLs zuzugreifen. Standardmäßig wird HTML ausgeliefert. Wenn ein anderes Format zurückgeliefert werden soll, würde die Funktion etwas aufwendiger aussehen (Listing 6 in gekürzter Form). Wir benutzen an dieser Stelle die HTML-Variante und erweitern das Designdokument um die Show-Funktion (Listing 7).

Die Show-Funktion *html_doc* kann genutzt werden, um einzelne Dokumente in einer HTML-Ausgabe im Browser darzustellen. Folgender URL soll in die Adresszeile eingegeben werden: http://localhost:5985/kassenbuch2/_design/buchhaltung/_show/html_doc/b0001. Das Ergebnis ist eine simple HTML-Tabelle wie in **Abbildung 1**. Damit ist der vierte Punkt des Konzepts abgeschlossen.

Die List-Funktion

Im letzten Punkt des Konzepts sollen alle Buchungen in einer Tabelle im Webbrowser dargestellt werden. Dazu bedient man sich einer List-Funktion. Im Unterschied zur Show-Funktion, wird die Ausgabe nicht aufgrund eines Dokuments, sondern mehrerer Dokumente vorgenommen. Dabei werden die Dokumente durch eine View bereitgestellt. Es ist auch möglich, sämtliche Parameter an den URI mitzugeben. Folglich ist der oben beschriebene *group_level*-Parameter ebenfalls nutzbar. Wie auch die Show-Funktion und die Views, sind die List-Funktionen in einem Designdokument beheimatet. Eine mögliche Funktion für diese Zwecke wird in Listing 8 dargestellt.

Wie aus dem Code ersichtlich, stellt CouchDB einige Funktionen zur Verfügung. Beispielsweise erlaubt *start()* verschiedene Optionen für die Ausgabe zu setzen. In diesem Fall wird ein HTTP-Header gesetzt, der angibt, dass die Ausgabe als HTML stattfindet. Die Methode *getRow()* gibt alle Attribute eines Dokuments zurück. In diesem Fall werden die Daten in eine lokale Variable *row* geschrieben und darüber iteriert. Die Methode *send()* ist schließlich für die Ausgabe von Content zuständig und gibt an dieser Stelle HTML-Code aus. Aber wie sieht nun das Ergebnis aus? Es gibt mehrere Möglichkeiten der Anzeige. Am allgemeinsten ist die reine Ausgabe der Map-Funktion, ohne die Daten an die Reduce-Engine zu geben.

Abb. 2:
Ausgabe
der Map-
Funktion

ID	Typ	MwSt	Datum	Wert
b0003	Ausgabe	0.19	2011-01-12	6.3
b0004	Ausgabe	0.7	2011-01-12	85.2
b0007	Ausgabe	0.7	2011-01-13	15.6
b0009	Ausgabe	0.7	2011-01-14	126.5
b0010	Ausgabe	0.7	2011-01-14	19.5
b0001	Einnahme	0.19	2011-01-04	22.5
b0002	Einnahme	0.19	2011-01-10	423.68
b0005	Einnahme	0.19	2011-01-13	465.3
b0006	Einnahme	0.19	2011-01-13	1234.5
b0008	Einnahme	0.19	2011-01-14	666.66

Der URL sieht wie unter [5] aus. Das Ergebnis könnte dann wie in **Abbildung 2** aussehen.

Das Ergebnis im Browser hängt davon ab, wie der *group_level*-Parameter an den URL angehängt wird [6]. Es erscheint z. B. eine Fehlermeldung, wenn *group_level=0* oder keine Parameter an den URL angehängt werden. Das liegt daran, dass die Reduce-Funktion nur ein einziges aggregiertes Ergebnis zurückgibt. Man kann den Fehler vermeiden, indem man innerhalb der Iteration prüft, ob *row.key* gesetzt ist und falls nicht, mit *return* beendet. Das Projekt ist damit erfolgreich umgesetzt.

Fazit

In diesem Artikel wurde aufgezeigt, wie auf die Daten der Dokumente einer Datenbank mit einer Show- oder List-Funktion zugegriffen werden kann. Natürlich können an dieser Stelle nur die Grundprinzipien gezeigt werden. Die Dokumentation von CouchDB in [7] und [8] zeigt noch weitere Möglichkeiten. Und richtig spannend wird der Einsatz der beiden Funktionen in Zusammenhang mit Templates und vor allem mit CouchApps. Das könnte allerdings den Inhalt eines weiteren Artikels füllen. Falls Interesse besteht, schreiben Sie mir. Bald erscheint ein CouchDB-Praxisbuch [9], das ich gemeinsam mit Till Klampäcker geschrieben habe und das ich Ihnen unverholten empfehlen möchte.



Andy Wenk ist Software Developer bei SinnerSchrader in Hamburg, hat dort viel Spaß mit JavaScript und Ruby on Rails und findet CouchDB cool. Zusammen mit Till Klampäcker (@klimpong) aus Berlin schreibt er momentan eines der ersten deutschsprachigen CouchDB-Bücher. Sie erreichen ihn unter andy@nms.de und [@awenkhk](https://twitter.com/awenkhk).

Links & Literatur

- [1] Wenk, Andreas: „CouchDB- map/reduce – oder wie Abfragen in CouchDB erstellt werden“, in Entwickler Magazin 1.2011, S. 65–72
- [2] <http://curl.haxx.se/>
- [3] <http://couchdb.apache.org>
- [4] http://localhost:5984/_utils/
- [5] http://localhost:5985/kassenbuch2/_design/buchhaltung/_list/html_list/auswertung?reduce=false
- [6] http://localhost:5985/kassenbuch2/_design/buchhaltung/_list/html_list/auswertung?group_level=2
- [7] <http://wiki.apache.org/couchdb/>
- [8] <http://docs.couchone.com/>
- [9] <http://goo.gl/D4oNp>

Wenn der URL zu lang wird

Die Beispiele beinhalten ziemlich viel JSON-Code. Das kann sehr schnell unschöne URLs produzieren. Abhilfe schafft man, indem man den JSON-Code in einer Datei speichert und dann als Parameter von *curl* an den URL anhängt. Dafür nutzt man die Option *-d* für *data* und das *@*-Zeichen für *attachment*:

```
curl -X PUT http://localhost:5984/kassenbuch/_design/buchhaltung -d \
  @design_doc.json
```