Automation at your Fingertips

Metadata-based autocompletion for Primo (and possibly others)

Felix Ostrowski (graphthinking GmbH, Berlin)
Uwe Dierolf (KIT Library, Karlsruhe)
Introduction

- In the beginning there was EXIT!
Royal Library sunbeds - a project idea was born
KIT Library

- KIT = Karlsruhe Institute of Technology
- 24/7 since April 2006
- 26,000 students
- 10,000 staff and researchers
- 5000 visitors per day
KIT Library as a service provider

- Hochschule Karlsruhe
  - university of applied sciences
  - 8500 students
  - 1000 staff & researchers

- DHBW Karlsruhe
  - Baden-Württemberg Cooperative State University
  - 3500 students
  - 700 assistant professors
  - 80 full-time professors
The KIT catalog

- is based on Ex Libris' resource discovery system Primo
- running at KIT since april 2012
- includes titles from different scopes
  - 3 universities
  - institutional repository KITopen
- offers features such as faceting
- but lacks an useable autocomplete feature
Requirements for a self-made autocompletion

- Autocomplete current word
- Suggest next word
- Correct spelling errors
- Avoid zero-hit suggestions
- Consider Primo scopes and facets
- Offer advanced mode for librarians and power-users
- Integration of this service should be possible also in the libraries website
Google-like Autocomplete

- Suggestions are based on previous queries
- This works well with large scale usage and index size
- Avoids zero hit queries
Problems in Library Land

- Usage of systems like Primo typically experience much less usage
- Deriving sensible suggestions from user input is almost impossible
- Thus Ex Libris’ autocompletion is based on global user input
- At least in case of KIT catalog the results are mediocre
  - far too many suggestions lead to zero hit queries
  - and many words present in the index are not suggested at all
Metadata to the rescue!

While we don’t have enough user input to generate sensible word suggestions,

But we do have another source: our beloved metadata!

If it contains enough textual information to be used for search, can’t it be used for word completion and suggestion as well?
Metadata to the rescue!

While we don't have enough user input to generate sensible word suggestions, we do have another source: our beloved metadata! If it contains enough textual information to be used for search, can't it be used for word completion and suggestion as well?
Main idea - deconstructing highly structured metadata

- In the end, we simply need a list of possible words
- So we throw all metadata fields that contain sensible words for suggestions into a single field
  - “autocomplete”
- And another field for internal usage such as faceting and scoping into another one
  - “property”
Main idea - further details

● Extract all words from our metadata

● Create an index with only 2 fields
  ○ autocomplete & property
  ○ the transformation is configurable using XPath expressions

● “autocomplete”-field
  ○ correct typing mistakes
  ○ create auto-completion for the current word
  ○ create auto-suggestion for next word

● “property”-field as a helper
  ○ scopes - restrict the results in all 3 use cases to relevant parts of the whole index (like a view)
  ○ facets - contains all internally used facet values
How to implement your own autocomplete feature?

- transform your data
  - use the language of your choice
- create an index using a search engine technology
  - Elasticsearch
- implement the autocompletion queries
  - JSON
- incept your user interface code
  - Javascript
Transforming XML to JSON

- `xml2json.php`
- Xpath to extract data

```php
$metadataXpaths = array(
    '/metadata/record/titelsatz/search/titel',
    '/metadata/record/titelsatz/search/autor[not(@typ) or @typ!="idn"]',
    '/metadata/record/titelsatz/search/verlag',
    '/metadata/record/titelsatz/search/schlagwort[not(@typ) or @typ!="idn"]',
    '/metadata/record/titelsatz/search/fachgebiet'
);

$propertyXpaths = array(
    '/metadata/record/eigenschaften/eigenschaft',
    '/metadata/record/titelsatz/search/fachkuerzel',
    '/metadata/record/titelsatz/search/sprache',
    '/metadata/record/titelsatz/search/erfassungsquartal',
    '/metadata/record/lokalsatz/search/bibliothek',
    '/metadata/record/lokalsatz/search/zweigstelle',
    '/metadata/record/lokalsatz/search/standort',
);```
Elasticsearch

- Lucene based full-text search
- Near real-time
- Schema-less
- Open source
- JSON over HTTP
Index configuration

- Use the Elasticsearch analyzer
  - Tokenize
  - Lowercase & more normalization
  - Remove stopwords
- analyze “autocomplete” field
- do not analyze “property” field

```
$ curl -s -XGET 'localhost:9200/autocomplete/_analyze' -d ' {
   "analyzer" : "autocompletion",
   "text" : "Innovations in Automation, Robotics and Measurement Techniques"
}' | json_pp
{
   "tokens" : [
   {
      "token" : "innovations"
   },
   {
      "token" : "automation"
   },
   {
      "token" : "robotics"
   },
   {
      "token" : "measurement"
   },
   {
      "token" : "techniques"
   }
   ]
}
```
Implement the autocompletion queries with Elasticsearch

- Term suggester for spelling corrections
- Term aggregations to autocomplete current
- Term aggregations to suggest next word
- Filter aggregations queries to avoid zero-hit suggestions
- Exact match queries to limit to scopes and facets
- All tied together in a single query issued by a simple Javascript plugin
Staying up-to-date - automated ingest process

- RDS workflow has been adapted by IT team
- KIT XML for all records is built every weekend
- Ingest script works on Elasticsearch server
  - harvests the KIT XML from RDS server via HTTP
  - transforms XML into JSON
  - runs for both Primo servers (test & production)
Stream processing pipeline

- Unzip compressed XML metadata to STDOUT
- Stream process XML from STDIN using XPaths, map XML to two JSON fields in PHP:
  - autocomplete
  - property
- Output JSON format suitable for Elasticsearch Bulk API to STDOUT
- Pipe directly via cURL to Elasticsearch Index
- 1.5 Mio records indexed in ~8min on a virtual Dual Core with 8 GB RAM running both the conversion and Elasticsearch
Automation at your fingertip - scopes
Poweruser mode with ".:" as prefix
Beyond Primo - autocompletion everywhere
How to manage a remote project?

- Distance between Karlsruhe and Berlin are in about 6 hours by train
- Which toolset did we use?
  - telephone and teamviewer
    - to communicate and see what you are talking about
  - github
    - to manage source code and issues
  - work in Berlin as if you were located in Karlsruhe
    - use tunneling ( SOCKS )
  - incept the live Primo service running in Karlsruhe from Berlin
    - "Resource Override" plugin for Google Chrome
Resource Override for Google Chrome

<table>
<thead>
<tr>
<th>Tab URL: <em>primo</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>From: *kit_primo4.js</td>
</tr>
<tr>
<td>From: *kit_primo4.js</td>
</tr>
<tr>
<td>From: *kit_primo4_test.js</td>
</tr>
</tbody>
</table>

Add Tab Group
Results

- autocompletion within Primo
- autocompletion on every page of the KIT Library Website having a searchslot
More Results

- suggestions depend on the "context"
  - works within the live production system but also in the testing environment
  - Primo scopes are supported
    - location specific search restrictions (KIT, HSKA, DHBW)
    - data specific search restrictions (KITopen repository)
  - many Primo facettes are supported
- advanced mode for power users offers more suggestions
- solution is adaptable to other environments
  - blogs, CMS, database
Questions?

Here and now or any time to 

uwe.dierolf@kit.edu

and

felix.ostrowski@graphthinking.com