

“Einmal durch linked-swissbib mit einem Suchrequest” Vers. 0.1

Aufruf von: <http://localhost/vflinked/Exploration/Search?lookfor=hello>

Diese Zeilen sollen das statische Bild der vorhandenen Klassendiagramme 'dynamisieren' Es wird auf UML Interaktionsdiagramme zu Gunsten von hoffentlich verständlichem Text verzichtet.

Hinweis: Pfadangaben beziehen sich immer auf das Basisverzeichnis der Applikation

0) *Start des requests im FrontController*

alle requests, sie sich nicht direkt auf eine Ressource beziehen, werden durch den sog. FrontController geleitet (public/index.php)

Die wesentliche Zeile im FrontController ist

```
// Run the application!
```

```
Zend\Mvc\Application::init(require 'config/application.config.php') -  
>run();
```

Wie man aus dem Namespace des Application Objekts sehen kann: Es setzt das MVC (Model View Controller) Pattern um.

1) *ZF spezifische tasks in der Methode init()*

- Laden von Konfigurationen und konfigurierten Modulen.

Die LinkedSwissbib Funktionalität ist im Modul module/LinkedSwissbib implementiert.

Ist das Modul nicht konfiguriert (im swissbib classic) kann es nicht geladen werden und die Funktionalität ist nicht vorhanden

- Aufsetzen von sog. Listnern und ServiceManager. Letzterer ist die Instanz, die im wesentlichen alle Typen kennt und bei dem diese angefordert werden können (dies ist der Grundsatz von dem es Erweiterungen gibt. Die wichtigste Erweiterung: Es gib noch verschiedenste spezialisierte ServiceManager (meistens PluginManager genannt), die für die Initialisierung von spezialisierten Typen verantwortlich sind

- Erstellen eines zentralen Objekts 'Application'. Dieses übernimmt die weitere Steuerung des Ablaufs.

- Application startet ein sog. Bootstrapping (im Rahmen des Bootstrapping werden von ZF Methoden der geladenen Module aufgerufen.

Die statische Methode `init` gibt ein erstelltes Objekt `Application` zurück. Auf diesem Objekt ruft der FrontController die Methode `run()` auf. (vgl. Obiges code Schnipsel)

2) *Was passiert in der methode run()*

- wie der Name sagt: jetzt ist eigentlich erst der Moment, an dem der eigentliche request (s. oben)

ausgeführt wird. Alles andere waren vorbereitende Abläufe.

- Vorabbemerkung: Viele der jetzt stattfindenden Aktionen werden über einen sog. EventManager ausgelöst. Dieser kann über den in der init Methode aufgesetzten Service Manger bezogen werden. Prinzip des EventManagers: Im EventManager werden Aktionen (events) definiert. An diese Aktionen können sich 'Codeteile' registrieren. Der EventManager wird 'triggert' events wie Event_Route oder Event_Dispatch oder Event_Finish.

- in der Methode run() wird als erste Aktion der event Event_Route 'gettriggert'. Dieser sucht nach dem passenden Controller für den Aufruf Exploration/Search. In der Regel legt man Definitionen, die unter anderem von der ZF Route-Komponente ausgewertet werden, in der Datei module/LinkedSwissBib/config/module.config.php ab.

s. dort:

```
$staticRoutes = [  
    'Exploration/Search', 'Elasticsearch/Results', 'Sparql/Results'  
];
```

- ist das Routing erfolgreich (das heisst es konnte ein Controller gefunden werden), wird der Event Event_Dispatch gettriggert.

Dort passieren diverse Dinge, letztendlich ruft ZF die Methode searchAction des Controllers LinkedSwissBib/Controller/ElasticSearchController auf. Hier gibt es eine Reihe von Konventionen.

3) Ablauf in der Action des Controllers

Nun ein Blick auf das Klassendiagramm "backend.overview.graphml"

allgemein:

- alle grün eingefärbten Typen gehören zum Namespace des Moduls module/LinkedSwissbib
- alle gelb eingefärbten Typen zum Modul VuFind
- alle rot eingefärbten zum Modul VuFindSearch
- alle violett eingefärbten zum ZF Framework

→ ElasticSearchController ist von allgemeinen VuFind Search Controllern abgeleitet. Das heisst, wir versuchen die Logik der Controller für Suchfunktionalität, wie sie von VuFind auch für andere targets verwendet werden (SOLR, Summon, etc), wiederzuverwenden und nur an den Stellen wo nötig zu erweitern

→ die Suchlogik von VuFind benutzt zur Umsetzung des Modells (Teil M des MVC Patterns) vor allem das 'Dreierpaar' Results / Params / Options

→ deswegen werden diese zu Anfang des requests initialisiert und vom Controller referenziert.

(s. VuFind/Controller/AbstractSearch/resultsAction)

```
$results = $this->getResultsManager()->get($this->searchClassId);
```

\$this->searchClassId enthält den String ElasticSearch. Deswegen werden die Typen aus dem Namensraum LinkedSwissbib/Search/Elasticsearch erstellt (zum grossen Teil unter der Verwendung

von ZF Mechanismen. S. hierzu die Typen in
LinkedSwissbib/Search/Options
LinkedSwissbib/Search/Params
LinkedSwissbib/Search/Results

Dabei handelt es sich um oben bereits erwähnte PluginManager (spezialisierte ServiceManger mit je einer Factory-Methode)

Für die Abfrage von ES werden die Tyen aus dem namespace LinkedSwissbib/Search/ElasticSearch mit Hilfe der erwähnten PluginManager erstellt. Diese Typen sind wiederum von Basistypen aus dem Bereich VuFind abgeleitet.

Hinweis: möchte man zum Beispiel einen zusätzlichen Sparql Server integrieren -ist noch abzuklären-, ist dies durch die Verwendung dieses Mechnaismus relativ leicht möglich)

4) Grobe Einteilung, wofür Results / Params / Options verwendet werden

Results:

- referenziert Params und Options
 - setzt den eigentlich SearchRequest ab (dafür verwendet es Mechanismen aus dem Modul VuFindSearch)
- vgl: `$results->performAndProcessSearch()` ;
- nimmt das Abfrageergebnis von VuFindSearch entgegen und steuert die Erstellung von sog. RecordDrivern. Letztere enthalten die Daten für die View-Komponente

Params: sammelt die Informationen die mit einem request mitgegeben werden (in unserem Beispiel lookfor=hello) und mischt diese mit in der abgelegten Konfigurationen
vergeliche den Aufruf:

```
$params->initFromRequest(  
    new Parameters(  
        $this->getRequest()->getQuery()->toArray()  
        + $this->getRequest()->getPost()->toArray()  
    )  
);
```

Options: enthalten im wesentlichen die konfigurierten Optionen. Für Solr gibt es hier (durch VuFind) sehr viel. Für ElasticSearch müssen wir dies noch machen

5) Abläufe im Modell,

Diese werden mit dem Aufruf

`$results->performAndProcessSearch()` ;
gestartet.

- wie der Name performAndProcess wohl ausdrücken soll, wird zuerst die Suche gestartet (perform) und anschliessend das Ergebnis verarbeitet (process)

- in VuFind/Search/Base/Results erfolgt der Aufruf
`$this->performSearch();`

Diese Methode wurde in LinkedSwissbib/Search/Elasticsearch/Results überschrieben

wichtigste Aktion hier: der Aufruf der Methode:

```
$searchService = $this->getSearchService();
```

Der Typ VuFindSearch\Service ist sozusagen der 'Einstiegspunkt' in das Modul VuFindSearch. Hier sind allgemeine Mechanismen zum Umgang mit den diversen targets von VuFind implementiert. Mit der wichtigsten: Suche nach dem passenden Backend für das aktuelle target (bei uns ES)

Dazu gibt es den Aufruf:

```
$collection = $searchService  
->search($this->backendId, $query, $offset, $limit, $params);
```

Dekodiert heisst dieser Aufruf:

- Der Searchservice soll suchen (VuFindSearch\Service->search())
- bevor er das machen kann, muss er mit der backendID (bei uns Elasticsearch) das richtige Backend initiieren

(vgl. Aufruf `$this->triggerPre($backend, $args);`)

das heisst es wird hierfür der EventManger benutzt

- und dann auf dem Backend die Suche absetzen

```
$response = $backend->search($query, $offset, $limit, $params);
```

Damit das Backend die Suche überhaupt absetzen kann, muss es wissen wonach es suchen soll. Diese Angaben sind in den Parametern der Methode hinterlegt

@Thomas: hier siehst Du Hinweise zu Deiner Farge von gestern zu offset und limit)

- für Elasticsearch kennt VuFind noch kein Backend. Die von VuFind mitgelieferten kann man im Namespace VuFindSearch\Backend finden.

Deswegen wurde das Gerüst von mir in einer ersten Version implementiert (LinkedSwissbib/Backend).

Der VuFindSearch-Service ruft deswegen die Methode

LinkedSwissbib\Backend\Elasticsearch\Backend->search() auf











Auch hier kann man eigentlich wieder recht gut erkennen, dass das Einbinden eines anderen Backends (z.B irgendeine Form von Graphenserver oder sonstiges relativ leicht zu malchen ist)

Die aktuelle search Methode ist noch eine Baustelle! Vor allem wie der query gegen Elasticsearch aufgebaut wird ist erst rudimentär. S. die Aufrufe

```
$this->getQueryBuilder()->setParams($params);  
$esDSLParams = $this->getQueryBuilder()->build($query);
```

Die eigentliche Abfrage gegen ES kann man unterschiedlichen clients abgesetzt werden
a) Elastica (objektorientierter aber viele Abhängigkeiten)

```
$esquery = new esQuery(['sb-s2.swissbib.unibas.ch' => '8080', 'sb-  
s6.swissbib.unibas.ch' => '8080', 'sb-s7.swissbib.unibas.ch' =>  
'8080'],  
    'testsb', 'bibliographicResource');  
$esquery->search(10);  
//$bag = $esquery->serialiserdf('turtle');  
$test = $esquery->resultSet;
```

```
▼  $test = {Elastica\ResultSet} [9]  
   _class = "Elastica\ResultSet"  
  ▶  _results = {array} [10]  
   _position = 0  
  ▶  _response = {Elastica\Response} [6]  
  ▶  _query = {Elastica\Query} [3]  
   _took = 132  
   _timedOut = false  
   _totalHits = 21017703  
   _maxScore = 1
```

ElasticSearch native Implementierung

```
$response = $this->connector->search($esDSLParams);
```

mit dem Ergebnis:

```

$response = {array} [4]
  took = 18
  timed_out = false
  _shards = {array} [3]
  hits = {array} [3]
    total = 611
    max_score = 6.6004243
    hits = {array} [10]
      0 = {array} [5]
      1 = {array} [5]
      2 = {array} [5]
      3 = {array} [5]
      4 = {array} [5]
      5 = {array} [5]
      6 = {array} [5]
      7 = {array} [5]
      8 = {array} [5]
      9 = {array} [5]

```

Man sieht, dieses ist array-basiert. Aktuell wird die Abfrage sogar zweimal abgesetzt, für die weitere Verwendung nehme ich aber nur die Arrays.

6) Erzeugen einer Struktur, in der das Ergebnis verpackt wird:

```
$collection = $this->createRecordCollection($response);
```

Den Inhalt der Struktur erkennt man am besten aus dem nächsten screen-shot

```

▼ ▣ $collection = {LinkedSwissbib\Backend\Elasticsearch\Response\RecordCollection} [5]
  ▣ response = null
  ▼ ▣ records = {array} [10]
    ▼ ▣ 0 = {LinkedSwissbib\RecordDriver\ElasticSearchRDF} [8]
      ▶ ▣ propertiesToNameSpaces = {array} [14]
        ▣ sourceIdentifier = "Solr"
        ▣ extraDetails = {array} [0]
      ▶ ▣ mainConfig = {Zend\Config\Config} [4]
      ▶ ▣ recordConfig = {Zend\Config\Config} [4]
      ▼ ▣ fields = {array} [5]
        ▣ _index = "testsb"
        ▣ _type = "bibliographicResource"
        ▣ _id = "212817795"
        ▣ _score = 6.6004243
        ▼ ▣ _source = {array} [12]
          ▶ ▣ @context = {array} [9]
            ▣ @id = "http://data.swissbib.ch/resource/212817795"
            ▣ dc:contributor = null
          ▶ ▣ rdfs:isDefinedBy = {array} [1]
          ▶ ▣ dct:language = {array} [1]
          ▶ ▣ @type = {array} [2]
          ▶ ▣ rdau:mediaType = {array} [1]
            ▣ dct:bibliographicCitation = "Dialog. Dialog Vertrieb. - 63/7"
            ▣ dct:title = "Hello, hello vielsprachige Schweiz"
            ▣ dct:issued = "1990"
          ▶ ▣ rdau:placeOfPublication = {array} [1]
          ▶ ▣ rdau:contentType = {array} [1]
        ▶ ▣ tableManager = {VuFind\Db\Table\PluginManager} [21]
        ▶ ▣ translator = {Zend\Mvc\I18n\Translator} [1]
      ▶ ▣ 1 = {LinkedSwissbib\RecordDriver\ElasticSearchRDF} [8]
      ▶ ▣ 2 = {LinkedSwissbib\RecordDriver\ElasticSearchRDF} [8]
      ▶ ▣ 3 = {LinkedSwissbib\RecordDriver\ElasticSearchRDF} [8]
      ▶ ▣ 4 = {LinkedSwissbib\RecordDriver\ElasticSearchRDF} [8]
      ▶ ▣ 5 = {LinkedSwissbib\RecordDriver\ElasticSearchRDF} [8]
      ▶ ▣ 6 = {LinkedSwissbib\RecordDriver\ElasticSearchRDF} [8]
      ▶ ▣ 7 = {LinkedSwissbib\RecordDriver\ElasticSearchRDF} [8]
      ▶ ▣ 8 = {LinkedSwissbib\RecordDriver\ElasticSearchRDF} [8]
      ▶ ▣ 9 = {LinkedSwissbib\RecordDriver\ElasticSearchRDF} [8]
    ▣ source = null
    ▣ pointer = 10
    ▣ offset = 0

```

- diese RecordCollection wird vom SearchService an das ElasticSearch/Results Objekt zurückgeliefert.
Die Zeilen im Überblick:

```

$searchService = $this->getSearchService();
try {
    $collection = $searchService
        ->search($this->backendId, $query, $offset, $limit,
$params);
} catch (\VuFindSearch\Backend\Exception\BackendException $e) {
    throw $e;
}
// Construct record drivers for all the items in the response:
// todo: by now we don't get any result
$this->results = $collection->getRecords();
}

```

\$this->results enthält nach dieser Zuweisung:

```

▼ results = {array} [10]
  ► 0 = {LinkedSwissbib\RecordDriver\ElasticSearchRDF} [8]
  ► 1 = {LinkedSwissbib\RecordDriver\ElasticSearchRDF} [8]
  ► 2 = {LinkedSwissbib\RecordDriver\ElasticSearchRDF} [8]
  ► 3 = {LinkedSwissbib\RecordDriver\ElasticSearchRDF} [8]
  ► 4 = {LinkedSwissbib\RecordDriver\ElasticSearchRDF} [8]
  ► 5 = {LinkedSwissbib\RecordDriver\ElasticSearchRDF} [8]
  ► 6 = {LinkedSwissbib\RecordDriver\ElasticSearchRDF} [8]
  ► 7 = {LinkedSwissbib\RecordDriver\ElasticSearchRDF} [8]
  ► 8 = {LinkedSwissbib\RecordDriver\ElasticSearchRDF} [8]
  ► 9 = {LinkedSwissbib\RecordDriver\ElasticSearchRDF} [8]

```

also: alle sog. RecordDriver, die wiederum in der View-Komponente verwendet werden können. Welche Informationen gesammelt und damit an die View-Komponente geliefert werden ist natürlich weiter ausbaufähig!

- damit sind wir wieder in der Methode
VuFind/Search/Base/Results/performAndProcessSearch()
angekommen. Nochmal die Abläufe als ganzes:


```

$this->startQueryTimer();
$this->performSearch();
$this->stopQueryTimer();
// Process recommendations:
$recommendations = $this->getParams()->getRecommendations(null);
if (is_array($recommendations)) {
    foreach ($recommendations as $currentSet) {
        foreach ($currentSet as $current) {
            $current->process($this);
        }
    }
}

```

Die Informationen u.a. in den RecordDriver Objekten können noch weiter aufbereitet werden. Dafür verwendet VuFind den Oberbegriff Recommendation. Auch hier habe ich noch kaum etwas gemacht.

Damit sind wir zurück in der resultsAction Methode des Controllers. Das Modell ist erstellt und über das Results-Objekt abrufbar. Jetzt können noch spezifische Dinge gemacht werden (z.B. Historisierungen wenn Benutzer angemeldet sind) Wichtig ist jedoch folgende Anweisung:

```

$view->results = $results;

```

Ein ZF spezifischen Objekt View (die Brücke zur Oberfläche) wird unter anderem das Modell übergeben. Damit ist es in den templates abrufbar!

Das ViewObjekt wird dem ZF Framework zurückgegeben und das zu Anfang beschriebene Application Objekt triggert den nächsten event, das Rendering!

Durch diesen Mechanismus kommen wir in den Bereich des Verzeichnis themes. Für linked swissbib gibt es bisher nur:
themes/linkedswissbib/templates/elasticsearch/search.phtml

Dort können zum Beispiel die RecordDriver Objekte benutzt werden (aber natürlich noch weitere Dinge)

```

$results = $this->results->getResults();
foreach ($results as $rdfDriver): ?>
    <div>
        <a href="<?= $rdfDriver->getUniqueID() ?>">document - IRI</a>

```

```
</div>
<div>
    <?= preg_replace(['</>'], ['&lt;','&gt;'], $rdfDriver-
>getRdf()); ?>
</div>
    <div>&nbsp;</div>
<?endforeach;?>
```