to.science

Dr. Ingolf Kuss, Dr. Peter Reimer, Dr. Andres Quast, Jan Schnasse

Mar 30, 2023

CONTENTS:

Über dieses Dokument

Dieses Dokument kommt zusammen mit einem Vagrantfile und beschreibt eine beispielhafte Installation von Regal. Unter *Vagrant* findet sich eine Anleitung zur Installation in einer Virtualbox.

Eine Kurzaufstellung der wichtigsten API-Calls findet sich unter *API*.

Dieses Dokument ist im Format rst geschrieben und kann mit dem Werkzeug sphinx in HTML übersetzt werden. Mehr dazu im Abschnitt *Colophon*.

CHAPTER

ONE

ÜBERSICHT

to.science (ehemals “Regal”) ist eine Content Repository zur Verwaltung und Veröffentlichung elektronischer Publikationen. Es wird seit 2013 am Hochschulbibliothekszentrum NRW (hbz) entwickelt.

to.science basiert auf den folgenden Kerntechnologien:

- Fedora Commons 3
- Elasticsearch 1.1
- Drupal 7
- Playframework 2.4
- MySQL 5
- Java 8
- PHP 5

Für die Webarchivierung kommen außerdem Openwayback, Heritrix und WPull zum Einsatz.

- openwayback hbz-2.3.2
- pywb
- heritrix 3.2.0
- wpull

Regal ist ein mehrkomponentiges System. Einzelne Komponenten sind als Webservices realisiert und kommunizieren über HTTP-APIs miteinander. Derzeit sind folgende Komponenten im Einsatz:

- to.science.api
- to.science.labels
- to.science.forms
- skos-lookup
- thumby
- deepzoomer
- to.science.drupal

Drupal Themes

- zbmed-drupal-theme
- edoweb-drupal-theme

Über die Systemschnittstellen können eine ganze Reihe von Drittsystemen angesprochen werden. Die folgende Abbildung verschafft einen groben Überblick über eine typische Regal-Installation und die angebundenen Drittsysteme.

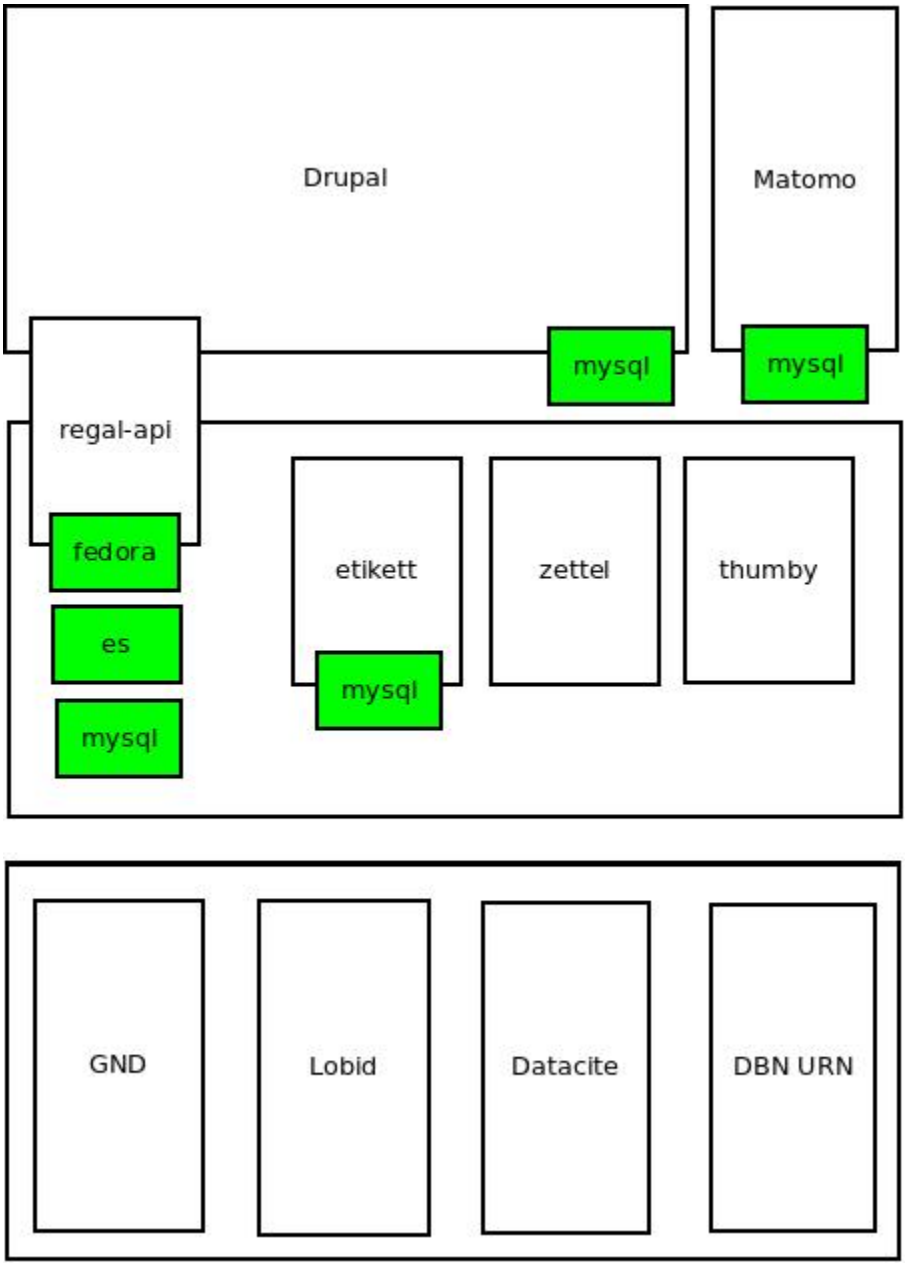


Fig. 1: Typische Regal-Installation mit Drupal Frontend, Backendkomponenten und angebundenen Drittsytemen

1.1 Konzepte

1.1.1 Objektmodell

Regal realisiert ein einheitliches Objektmodell in dem sich eine Vielzahl von Publikationstypen speichern lassen. Die Speicherschicht wird über *Fedora Commons 3* realisiert.

Eine einzelne Publikation besteht i.d.R. aus mehreren *Fedora Commons 3*-Objekten, die in einer hierarchischen Beziehung zueinander stehen.

Table 1: Fedora Object

Daten-strom	Pflicl	Beschreibung
DC	Ja	Von Fedora vorgeschrieben. Wird für die fedorainterne Suche verwendet
RELS-EXT	Ja	Von Fedora vorgeschrieben. Wird für viele Sachen verwendet - (1) Hierarchien - (2) Steuerung der Sichtbarkeiten - (2) OAI-Providing
data	Nein	Die eigentlichen Daten der Publikation. Oft ein PDF.
metadata oder meta-data2	Nein	Bibliografische Metadaten. Metadata2 wurde mit dem Umstieg auf die Lobid-API v2 einge-führt.
object-Times-tamp	Nein	Eine Datei mit einem Zeitstempel. Der Zeitstempel wird bei bestimmten Aktionen gesetzt.
seq	Nein	Eine Hilfsdatei mit einem JSON-Array. Das Array zeigt an, in welcher Reihenfolge Kindob-jekte anzuzeigen sind. Dieses Hilfskonstrukt existiert, da in der RELS-EXT keine RDF-Listen abgelegt werden können.
conf	Nein	Websites und Webschnitte speichern in einem conf-Datenstrom alle Parameter mit denen die zugehörige Webseite geharvested wurde.

Die Metadaten werden als ASCII-Kodierte N-Triple abgelegt. Da alle Fedora-Daten als Dateien im Dateisystem abgelegt werden, ist diese Veriante besonders robust gegen Speicherfehler. N-Triple ist ein Format, dass sich Zeilen-weise lesen lässt. ASCII ist die einfachste Form der Textkodierung.

Die Daten werden als “managed”-Datastream in den Objektspeicher der Fedora abgelegt. Eine Ausnahme bilden Web-seiten. Die als WARC gespeicherten Inhalte werden “unmanaged” lediglich verlinkt. Im Fedora Objektspeicher wird nur eine Datei mit der ensprechenden Referenz abgelegt.

1.1.2 Namespaces und Identifier

Jede Regal-Installation arbeitet auf einem festgelegten Namespace. Wenn über die *regal-api* Objekte angelegt werden, finden sich diese immer in dem entsprechenden Namespace wieder. Hinter dem Namespace findet sich, abgetrennt mit einem Dopplepunkt eine hochlaufende Zahl, die i.d.R. über *Fedora Commons 3* bezogen wird.

Der so zusammengesetzte Identifier kommt in allen Systemkomponenten zum Einsatz.

Table 2: Beispiel Regal Identifier

ID	Komponente	URL
regal:1	drupal	http://localhost/resource/regal:1
regal:1	regal-api	http://api.localhost/resource/regal:1
regal:1	fedora	http://localhost:8080/fedora/objects/regal:1
regal:1	elasticsearch	http://localhost:9200/regal/_all/regal:1

1.1.3 Deskriptive Metadaten

Regal unterstützt eine große Anzahl von Metadatenfeldern zur Beschreibung von bibliografischen Ressourcen. Jedes in Regal verspeicherte Objekt kann mit Hilfe von RDF-Metadaten beschrieben werden. Das System verspeichert grundsätzlich alle Metadaten, solange Sie im richtigen Format an die Schnittstelle gespielt werden.

Darüber hinaus können über bestimmte Angaben, bestimmte weitergehende Funktionen angesteuert werden. Dies betrifft u.a.:

- Anzeige und Darstellung
- Metadatenkonvertierungen
- OAI-Providing
- Suche

Alle bekannten Metadateneinträge werden in der Komponente *Etikett* verwaltet. In *Etikett* kann konfiguriert werden, welche URIs aus den RDF-Daten in das JSON-LD-Format von *regal-api* überführt werden. Außerdem kann die Reihenfolge der Darstellung, und das Label zur Anzeige gesetzt werden.

Table 3: Etikett-Eintrag für dc:title

Label	Pictogram	Name (json)	URI	Type	Container	Comment
Titel	keine Angabe	title	http://purl.org/dc/terms/title	String	keine Angabe	keine Angabe

Etikett-Eintrag als Json.

```
"title":{ "@id"="http://purl.org/dc/terms/title", "label"="Titel" }
```

Die etikett Datenbank wird beim Neustart jeder *regal-api*-Instanz eingelesen. Außerdem wird die HTTP-Schnittstelle von Etikett immer wieder angesprochen um zur Anzeige geeignete Labels in das System zu holen und anstatt der rohen URIs einzublenden. Das *regal-api*-Modul läuft dabei auch ohne den Etikett-Services, allerdings nur mit eingeschränkter Funktionalität; beispielsweise fallen Anzeigen von verlinkten Ressourcen (und das ist in Regal fast alles) weniger schön aus.

Wie kommen bibliografische Metadaten ins System?

In Regal können bibliografische Metadaten aus dem hbz-Verbundkatalog an Ressourcen “angelinkt” werden. Dies erfolgt über Angabe der ID des entsprechenden Titelsatzes (z.b. HT017766754). Mit Hilfe dieser ID kann Regal einen Titelimport durchführen. Dabei wird auf die Schnittstellen der Lobid-API zugegriffen.

Regal bietet außerdem die Möglichkeit, Metadaten über Erfassungsmasken zu erzeugen und zu speichern. Dies erfolgt mit Hilfe des Moduls *Zettel*. *Zettel* ist ein Webservice, der verschiedene HTML-Formulare bereitstellt. Die Formulare können RDF-Metadaten einlesen und ausgeben. Zettel-Formulare werden über Javascript mit Hilfe eines IFrame in die eigentliche Anwendung angebunden. Über Zettel werden Konzepte aus dem Bereich Linked Data umgesetzt. So können Feldinhalte über entsprechende Eingabeelemente in Drittsystemen recherchiert und verlinkt werden. Die Darstellung von Links erfolgt in Zettel mit Hilfe von *Etikett*. Umfangreichere Notationssysteme wie Agrovoc oder DDC werden über einen eigenen Index aus dem Modul *skos-lookup* eingebunden. Zettel unterstützt zur Zeit folgende Linked-Data-Quellen:

- Lobid (GND)
- Lobid (Ressource)
- Agrovoc
- DDC

- CrossRef (Funder Registry)
- Orcid
- Geonames
- Open Street Maps Koordinaten

1.1.4 Anzeige und Darstellung

Über die Schnittstellen der *regal-api* können unterschiedliche Darstellungen einer Publikation bezogen werden. Über Content Negotiation können Darstellungen per HTTP-Header angefragt werden. Um unterschiedliche Darstellungen im Browser anzeigen zu lassen, kann außerden, über das Setzen von entsprechenden Endungen, auf unterschiedliche Representationen eine Resource zugegriffen werden.

Auswahl von Pfaden zu unterschiedlichen Representationen einer Ressource.

/resource/regal:1 /resource/regal:1.json /resource/regal:1.rdf /resource/regal:1.epicur /resource/regal:1.mets

In der HTML-Darstellung greift *regal-api* auf den Hilfsdienst *Thumby* zu um darüber Thumbnail-Darstellungen von PDFs oder Bilder zu kreieren. Bei großen Bildern wird außerdem der *Deepzoomer* angelinkt, der eine Darstellung von hochauflösenden Bildern über das Tool OpenSeadragon erlaubt. Video- und Audio-Dateien werden über die entsprechenden HTML5 Elemente gerendert.

1.1.5 Der hbz-Verbundkatalog

Metadaten, die über den Verbundkatalog importiert wurden, können über einen Cronjob regelmäßig aktualisiert werden. Außerdem können diese Daten über OAI-PMH an den Verbundkatalog zurückgeliefert werden, so dass dieser, Links auf die Volltexte erhält.

1.1.6 Metadatenkonvertierung

Für die Metadatenkonvertierung gibt es kein festes Vorgehensmodell oder Werkzeug. I.d.R. gibt es für jede Representation eine oder eine Reihe von Javaklassen, die für eine On-the-fly-Konvertierung sorgen. Die HTML-Darstellung basiert grundlegend auf denselben Daten, die auch im Elasticsearch-Index liegen und ist im wesentlichen eine JSON-LD-Darstellung, die mit Hilfe der in *Etikett* hinterlegten Konfiguration aus den bibliografischen Metadaten gewonnen wurde.

1.1.7 OAI-Providing

Öffentlich zugängliche Publikationen sind auch über die OAI-Schnittstelle verfügbar. Dabei wird jede Publikation einer Reihe von OAI-Sets zugeordnet und in unterschiedlichen Formaten angeboten.

Table 4: OAI Set

Set	Kriterium
ddc:*	Wenn ein dc:subject mit dem String “http://dewey.info/class/” beginnt, wird ein Set mit der entsprechenden DDC-Nummer gebildet und die Publikation wird zugeordnet
contentType	Der “contentType” weist darauf hin, in welcher Weise die Publikation in Regal. Abgelegt ist.
open_acc	All Publikationen, die als Sichtbarkeit “public” haben
urn-set-1	Publikationen mit einer URN, die mit urn:nbn:de:hbz:929:01 beginnt
urn-set-2	Publikationen mit einer URN, die mit urn:nbn:de:hbz:929:02 beginnt
epicur	Publikationen, die in einem URN-Set sind
aleph	Publikationen , die mit einer Aleph-Id verknüpft sind
edoweb0	spezielles, pro *reg al-api*-Instanz konfigurierbares Set für alle Publikationen, die im aleph-Set sind
ellinet01	spezielles, pro *reg al-api*-Instanz konfigurierbares Set für alle Publikationen, die im aleph-Set sind

Table 5: OAI Metadatenformat

Format	Kriterium
oai_dc	Alle öffentlich sichtbaren Objekte, die als bestimmte ContentTypes angelegt wurden.
epicur	Alle Objekte, die eine URN haben
aleph	Alle Objekte, die einen persistenten Identifier haben
mets	Wie oai_dc
rdf	Wie oai_dc
wgl	Format für LeibnizOpen. Alle Objekte die über das Feld “collectionOne” einer Institution zugeordnet wurden und über den ContentType “article” eingeliefert wurden.

1.1.8 Suche

Der Elasticsearch-Index wird mit Hilfe einer JSON-LD Konvertierung befüllt. Die Konvertierung basiert im wesentlichen auf den bibliografischen Metadaten der einzelnen Ressourcen und wir mit Hilfe der in *Etikett* hinterlegten Konfiguration erzeugt.

1.1.9 Zugriffsberechtigungen und Sichtbarkeiten

Regal setzt ein rollenbasiertes Konzept zur Steuerung von Zugriffsberechtigungen um. Eine besondere Bedeutung kommt dem lesenden Zugriff auf Ressourcen zu. Einzelne Ressourcen können in ihrer Sichtbarkeit so eingeschränkt werden, dass nur mit den Rechten einer bestimmten Rolle lesend zugegriffen werden kann. Dabei kann der Zugriff auf Metadaten und Daten separat gesteuert werden.

Die Konfiguration hat Auswirkungen auf die Sichtbarkeit einer Publikation in den unterschiedlichen Systemteilen. Die folgende Tabelle veranschaulicht den derzeitigen Stand der Implementierung.

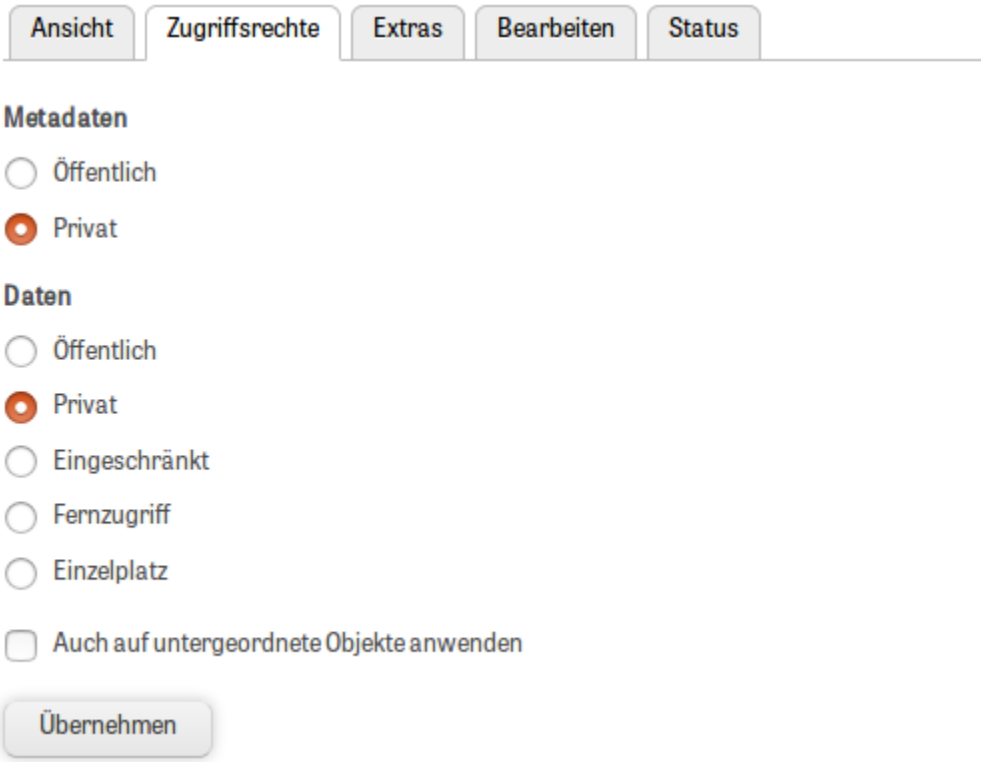


Fig. 2: Screenshot zur Verdeutlichung von Sichtbarkeiten in Regal

Sichtbarkeiten, Operationen, Rollen

Table 6: **Schreibender** Zugriff auf Daten und Metadaten

Rolle	Art der Aktion
ADMIN	Darf alle Aktionen durchführen. Auch Bulk-Aktionen und "Purges"
EDITOR	Darf Objekte anlegen, löschen, Sichtbarkeiten ändern, etc.

Table 7: **Lesender** Zugriff auf Metadaten

Sichtbarkeit	Rolle
public	GUEST,READ ER,SUBSCRIBER,REMOTE,ADMIN,EDITOR
private	ADMIN,EDITOR

Table 8: **Lesender** Zugriff auf Daten

Sichtbarkeit	Rolle
public	GUEST,READ ER,SUBSCRIBER,REMOTE,ADMIN,EDITOR
restricted	READ ER,SUBSCRIBER,REMOTE,ADMIN,EDITOR
remote	READ ER,SUBSCRIBER,REMOTE,ADMIN,EDITOR
single	SUBSCRIBER,ADMIN,EDITOR
private	ADMIN,EDITOR

1.1.10 Benutzerverwaltung

Die Benutzerverwaltung von Regal findet innerhalb von Drupal statt. Zwar können auch in der *regal-api* Benutzer angelegt werden, jedoch ist die Implementierung in diesem Bereich erst rudimentär.

Drupal

Benutzer in Drupal können über das Modul *regal-drupal* unterschiedlichen Rollen zugewiesen werden. Die Authorisierung erfolgt passwortbasiert. Alle Drupal-Benutzer greifen über einen vorkonfigurierten Accessor auf die *regal-api* zu. Alle Zugriffe erfolgen verschlüsselt unter Angabe eines Passwortes. Die Rolle mit deren Berechtigungen zugegriffen wird, wird dabei in *regal-drupal* gesetzt. Die Drupal-BenutzerId wird als Metadatum in Form eines proprietären HTTP-Headers mit an *regal-api* geliefert.

Regal-Api

Auch in regal-api können Api-Benutzer angelegt werden. Zur Benutzerverwaltung wird eine MySQL-Datenbank eingesetzt, in der die Passworte der Nutzer abgelegt sind.

1.1.11 Ansichten

Um Daten, die in *regal-api* abgelegt wurden zur Anzeige zu bringen sind i.d.R. mehrere Schritte nötig. Die genaue Vorgehensweise ist davon abhängig, wo die Daten abgelegt werden (in welchem Fedora Datenstrom). Grundsätzlich basiert die HTML-Darstellung auf den Daten, die unter dem Format `.json2` einer Ressource abrufbar sind und einen Eintrag in context.json haben.

Daten zur Ansicht bringen

1. Eintrag des zugehörigen RDF-Properties in die entsprechende *Etikett*-Instanz, bzw. in die `/conf/labels.json`. Der Eintrag muss einen Namen, ein Label und einen Datentyp haben. *regal-api* neu starten, bzw mit `POST /context.json` das neu Laden der Contexteinträge erzwingen.
2. Dies müsste reichen, um eine Standardanzeige in der HTML-Ausgabe zu erreichen
3. Wenn die Daten nicht erscheinen, sollte man überprüfen, ob sie unter dem Format `.json2` erscheinen. Wenn nicht, stellt sich die Frage, wo die Daten abgelegt werden. Komplett werden nur die Daten aus dem Fedora Datenstrom /metadata2 prozessiert. Befindet sich das Datum in z.B. im /RELS-EXT Datenstrom so muss es zunächst manuell unter `helper.JsonMapper#getLd2()` in das JSON-Objekt eingefügt werden.
4. Einige Felder werden auch ausgeblendet. Dies geschieht in *regal-api* unter `/public/stylesheets/main.css` und in Drupal innerhalb der entsprechenden themes.
5. Um spezielle Anzeigen zu realisieren muss schließlich im HTML-Template angefasst werden, unter `/app/views/tags/resourceView.scala.html` .

Insgesamt läuft es also so: Alles was in *Etikett* konfiguriert ist, wird auch ins JSON und damit ins HTML und in den Suchindex übernommen. Dinge, die im HTML nicht benötigt werden, werden über CSS wieder ausgeblendet.

1.2 Software

Die technische Dokumentation der HTTP-Schnittstelle findet sich unter *API*:

Nachfolgend sei eine Innenansicht der einzelnen Module aufgestellt. Die Integration der Module erfolgt i.d.R. über HTTPs. Die Module werden über entsprechende Einträge in der Apache-Konfiguration sichtbar gemacht. Es handelt sich also um eine Webservice-Architektur, in der alle Webservices über einen Apache-Webserver und entsprechende Einträge in ihren Konfigurationsdateien miteinander verbunden werden.

1.2.1 regal-api

Table 9: Überblick

Source	*regal-api <https: //github.com/edoweb/regal-api>*
Technik	*Play 2.4 .2 <https://www.playframework.com /documentation/2.4.x/JavaHome>*
Ports	9000 / 9100
Verzeichnis	/opt/regal/apps/regal , /opr/regal/src/regal
HTTP Pfad	/

Mit regal-api werden alle grundlegenden Funktionen von Regal bereitgestellt. Dies umfasst:

- HTTP Schnittstelle
- Sichtbarkeiten, Zugriffskontrolle, Rollen
- Speicherung, Datenhaltung
- Konvertierungen

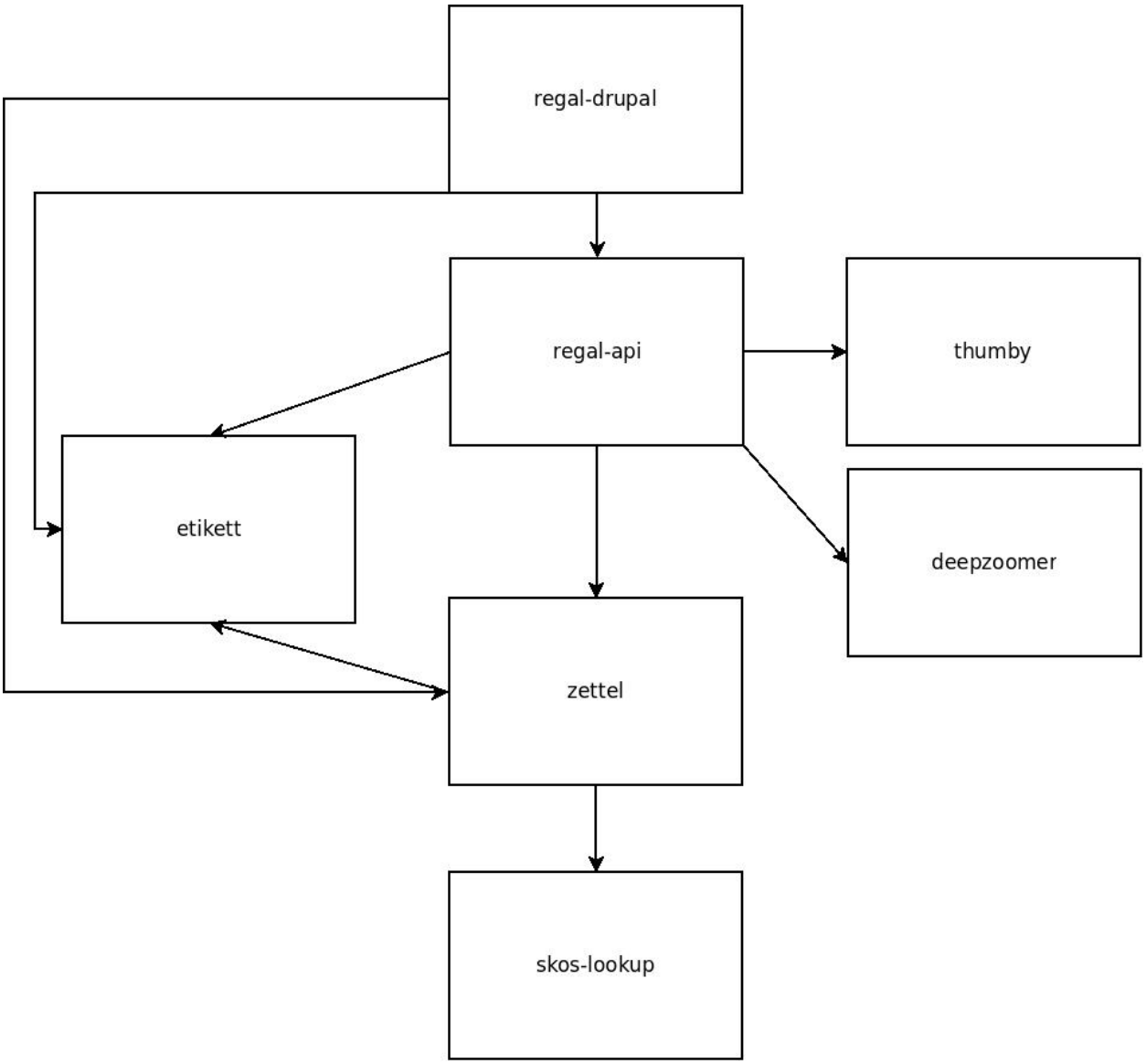


Fig. 3: Regal Abhängigkeiten

- Ansichten
- Suche
- Webarchivierung

Der Webservice ist auf Basis von Play 2.4.2 realisiert und bietet eine reichhaltig HTTP-API zur Verwaltung von elektronischen Publikationen an. Die *regal-api* operiert auf *Fedora Commons 3*, *MySql* und *Elasticsearch 1.1*. Über die API werden auch Funktionalitäten von *Etikett*, *Thumby*, *Zettel* und *Deepzoomer* angesprochen. Für die Webarchivierung werden *heritrix*, *wpull* und *openwayback* angebunden.

Konfiguration

Table 10: Dateien im /conf Verzeichnis

Datei	Beschreibung
aggregations.conf	Diese Datei wird verwendet um die Schnittstelle `/browse` zu konfigurieren. Die Einträg im Object "aggs" können direkt über die `/browse` Schnittstelle angesprochen werden. Mit Hilfe des Elasticsearch-Indexes wird dann eine entsprechende Antwort generiert. Beispiel: `/browse/rdftype` liefert eine Liste mit allen Publikationstypen, die im Index vorhanden sind.
application.conf.tmp	Eine template Datei für die Hauptkonfiguration von *regal-api*. Diese Datei sollte zur lokalen Verwendung einmal nach application.conf kopiert werden. In der Datei sind alle Passwörter auf *admin* gesetzt.
crawler-beans.cxml	Die Datei wird verwendet, wenn im Webarchivierungsmodul eine neue Konfiguration für eine Webseite angelegt wird.
ehcache.xml	die Konfiguration der Ehcache Komponente
fedora-users.xml	deprecated - Zur Löschung vorgeschlagen
hbz_edoweb_	deprecated - Zur Löschung vorgeschlagen
html.html	deprecated - Zur Löschung vorgeschlagen
in-stall.propertie	deprecated - Zur Löschung vorgeschlagen
labels-edoweb.de	Labels für eine bestimmt Regal-Instanz
labels-for - proceeding-and-researchData.	deprecated - Zur Löschung vorgeschlagen
labels-lobid.json	deprecated - Zur Löschung vorgeschlagen
labels-publisso.de	Labels für eine bestimmte Regal-Instanz
labels.json	Eine sinnvolle Startkonfiguration. Die Datei wurde mit *Etikett* erzeugt. Beim Start von *regal-api* wird zunächst versucht eine ähnliche Konfiguration direkt von einer laufenden *Etikett*-Instanz zu holen. Wenn dies nicht klappt, wird auf die labels.json zurückgegriffen.
list.html	deprecated - Zur Löschung vorgeschlagen
log-back.develope	Eine logging Konfiguration. Ich kopiere die immer nach logback.developer.js.xml (in .gitignore) und trage sie dann in die application.conf ein. Auf diese Weise kann ich an Loglevels herumkonfigurieren ohne das in diese Änderungen in die Versionsverwaltung spielen zu müssen.
log-back.xml	Konfiguration des Loggers. Diese Datei ist in application.conf eingetragen.
mab xml-string-template-on-record.xml	Eine template-Datei zur Generierung von MAB-Ausgaben.
mail.propertie	Konfiguration zur Versendung von Mails. Standardmäßig schickt die Applikation eine Mail, sobald sie im Production-Mode neu gestartet wurde. Auch der Umzugsservice im Webarchivierungsmodul verschickt Mails.
nwbib-spatial.ttl	deprecated - Zur Löschung vorgeschlagen
nwbib.ttl	deprecated - Zur Löschung vorgeschlagen
public-index-config.json	Konfiguration des Elasticsearch-Indexes. Da in dem Index vorallem Metadaten liegen, soll fast nicht tokenisiert werden.
routes	Hier sind alle HTTP-Pfade übersichtlich aufgeführt.
scm-info.sh	Diese Datei kann man unter Linux in die profile-Konfiguration seines Benutzers einbinden. Dann erhält man im Terminal farbige Angabgen zu Git-Branches,etc.
start-regal.sh	deprecated - Zur Löschung vorgeschlagen
tomcat	deprecated - Zur Löschung vorgeschlagen

Die Applikation

Table 11: Das /app Verzeichnis

Package	Beschreibung
default package	Hier befindet sich die Datei Global, die in Play 2 .4 noch eine große Rolle spielt. In der Datei können zum Beispiel Aktionen vor dem Start der Applikation erfolgen, auch können hier HTTP-Requests mit geloggt werden. Bestimmte Aktionen werden nur im Production-Mode ausgeführt, d.h. nur wenn die Applikation mit `start` gestartet wurde oder über `dist` ein entsprechendes Binary erzeugt wurde.
actions	Hier sind Funktionen versammelt, die meist unmittelbar aus den Controller-Klassen aufgerufen werden.
archi	Ein Reihe von Dateien, über die Zugriffe auf *Fedora Commons 3* organisiert werden. Hier finden sich auch einige Hilfsklassen (`Utils`). Das FedoraInterface zeigt an, welche Aktionen auf der Fedora ausgeführt werden. Der Code in diesem Paket gehört mit zu dem ältesten Code im gesamten Regal-Projekt.
archi	Zugriff auf die Elasticsearch
authenticate	Regal verwendet Basic-Auth zur Authentifizierung. Um die entsprechenden Aufrufe in den Controllern zu Schützen wird eine Annotation `@BasicAuth` verwendet. Diese findet sich hier. Die Annotation selbst bewirkt, dass jeder Controller-Aufruf durch die Methode `basicAuth` der Klasse `BasicAuthAction.java` läuft. Ziel dieser Prozedur ist es, dem aktuellen Zugriff die Berechtigungen einer bestimmten Rolle zuzuordnen.
controlle	Der Code, der in diesen Klassen organisiert ist, wird bei den entsprechenden HTTP-Aufrufen ausgeführt. In der `/conf/routes` Datei kann man sehen, welcher HTTP-Aufruf, welchen Methoden-Aufruf zur Folge hat. Die Controller-Klassen sind i.d.R. von der Klasse MyController abgeleitet, die Hilfsfunktionen bereitstellt, aber auch Funktionen zur Überprüfung von Zugriffsrechten. Die Überprüfung von Zugriffsrechten erfolgt durch eingebettet Calls und wird über die internen Klassen von MyController realisiert. Beispiel: Die Funktion `listNodes` in der Klasse `controllers.Resource` ruft ihre Prozeduren eingebettet in eine Funktion der Klasse `ListAction` auf. Die Klasse `ListAction` ist in `MyController` implementiert und überprüft, ob der Aufruf mit der nötigen Berechtigung erfolgte. Vgl. *Zugriffsberechtigungen und Sichtbarkeiten*
converte	Diese Datei realisiert das OAI-Providing von MAB-Daten. Ursprünglich war geplant, wesentlich umfangreichere MAB-Datensätze an den Verbundkatalog zu liefern. Daher wird hier mit einer eigenen Template-Engine gearbeitet, etc. Ein lustiges Produkt in diesem Kontext ist auch die Klasse `models.MabRecord`.
de.hb	Der hier befindliche Code kommt ursprünglich aus einem anderen Paket, wurde dann aber beim Neuaufbau des Lobid 2 Datendienstes gemeinsam mit den Kollegen weiterentwickelt und ist schließlich wieder hier gelandet. Mittlerweile ist die offizielle JSON-LD-Library soweit entwickelt, dass man die Konvertierung auch darüber machen kann. Achja, denn dafür ist der Code: Lobid N-Triples in schönes JSON umzuformen, das dann auch in den Elasticsearch-Index kann.
helpe	Die mit Abstand wichtigste Klasse in diesem Package heißt `JsonMapper`. Hier wird das JSON für Index und Ansichten erzeugt.
helpe	Emails verschicken.
helpe	Einige Klassen zur Regelung des OAI-Providings. Der `OAIDispatcher` analysiert, ob und wie ein `Node` an die OAI-Schnittstelle gelangt.
models	Die wichtigste Klasse hier ist `Node` über diese Klasse läuft der Großteil des Datentransportes.
views	Templates in der Sprache `Twirl` und einige Java-Hilfsklassen.
views	Ein paar Viewer, die über die Hilfsklasse `ViewerInfo` in `tags.resourceView` eingebunden werden können.
views	Mit `Twirl` XML zu generieren war keine gute Idee.
views	Hilfstemplates.

1.2.2 Etikett

Table 12: Überblick

Source	`Etikett`_
Technik	Play
Ports	9002 / 9102
Verzeichnis	/opt/regal/apps/etikett , /opr/regal/src/etikett
HTTP Pfad	/tools/etikett

Etikett ist eine einfache Datenbankanwendung, die es erlaubt

1. Menschenlesbare Labels für URIs abzulegen. Über eine HTTP-Schnittstelle kann dann nach dem Label gefragt werden.
2. Auch Konfigurationen zur Erzeugung eines JSON-LD Kontextes können abgelegt werden.
3. Die Etikett-Datenbank erweitert sich dynamisch. Wird in einem authentifizierten Zugriff nach einer noch nicht bekannten URI gefragt, so versucht die Applikation ein Label für die URI zu finden.

In Etikett sind verschiedene Lookups realisiert, die dynamisch Labels für URIs finden können. Beispiele:

- Crossref
- Geonames
- GND
- Openstreetmap
- Orcid
- RDF, Skos, etc.

Fragt man Etikett nach einem Label, so antwortet Etikett mit dem Ergebnis des Lookups. Wenn Etikett nicht in der Lage ist, ein Label zu finden, wird die URI, mit angefragt wurde, zurückgegeben.

Etikett kann auch als Cache verwendet werden. So werden authentifizierte Anfragen in einer Datenbank persistiert. Erneute Anfragen werden dann aus der Datenbank beantwortet, ein erneuter Lookup wird eingespart. Einmal persistierte Labels werden nicht invalidiert. Die Invalidierung kann von außerhalb über authentifizierte HTTP-Zugriffe realisiert werden, stellt aber insgesamt noch ein Desiderat dar.

Etikett kann auch mit Labels vorkonfiguriert werden. Dabei können zusätzliche Informationen zu jeder URIs mit abgelegt werden. Folgende Informationen können in etikett abgelegt werden:

- URI
- Label
- Weight - Zur Definition von Anzeigereihenfolgen.
- Pictogram Iconfont-ID - Kann anstatt oder zusätzlich zum Label angezeigt werden.
- ReferenceType - JSON-LD Typ
- Container - JSON-LD Container
- Beschreibung - Kommentar als Markdown

Mit Hilfe dieser Angaben kann Etikett auch einen “JSON-LD Context” bereitstellen. Insgesamt wird über Etikett eine Art “Application Profile” realisiert. Das Profil gibt Auskunft, welche Metadatenfelder (definiert als URIs) in welcher Weise (Typ, Container) Verwendung finden und wie sie angezeigt werden sollen (Label, Weight, Pictogram).

Im Regal-Kontext wird *Etikett* an vielen Stellen verwendet.

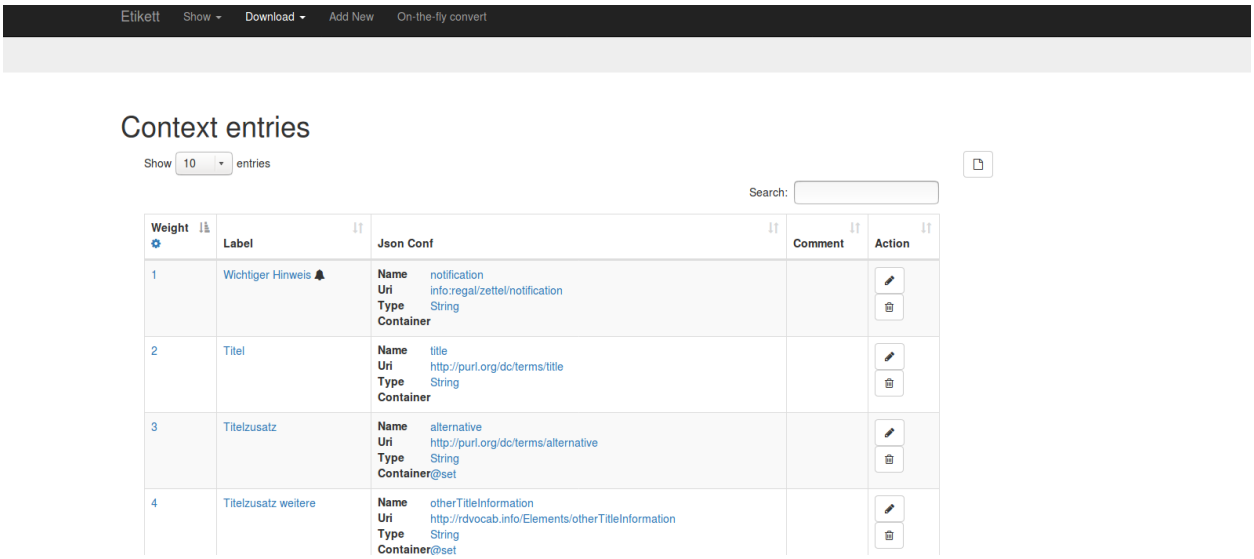


Fig. 4: Etikett Oberfläche

- Zur Wandlung von RDF nach JSON-LD
- Zur Anreicherung von RDF Importen
- Zur menschenlesbaren Darstellung von RDF
- Zur Konfiguration von Labels, Anzeigereihenfolgen und Pictogrammen
- Als Cache

Konfiguration

Table 13: Dateien im /conf Verzeichnis

Datei	Beschreibung
evolutions	Dieses Verzeichnis enthält SQL-Skripte, die bei Änderungen des Datenbankschemas automatisch über EBean angelegt werden. Beim nächsten Deployment einer neuen Etikett-Version werden die Skripte automatische angewendet. Die Skripte enthalten immer einen mit "Up" markierten Part, und einen mit "Down" markierten Part (für rollbacks).
application.co	Hier kann ein Benutzer eingestellt werden. Alle Klassen im Verzeichnis `models.*` erhalten eine SQL-Tabelle.
ddc.tur	Eine DDC Datei. Die Datei bietet Labels für DDC-URIs an.
labels.jso	Eine Labels-Datei, die zur initialen Befüllung verwendet werden kann.
regal.turt	Eine Labels-Datei, die zur initialen Befüllung verwendet werden kann.
routes	Alle HTTP-Schnittstellen übersichtlich in einer Datei
rpb.tur	Eine Labels-Datei, die zur initialen Befüllung verwendet werden kann.
rpb2.tu	Eine Labels-Datei, die zur initialen Befüllung verwendet werden kann.

Die Applikation

Table 14: Das /app Verzeichnis

Package	Beschreibung
default	In `Global` werden die Requests mit geloggt.
controllers	In `Application` werden alle HTTP-Operationen implementiert. Unterstützt wird BasicAuth.
helper	Verschiedene Klassen, die eine URI verfolgen und versuchen ein Label aus den zurückgelieferten Daten zu kreieren.
models	Das Model `Etikett` ist persistierbar.
views	Die meisten HTTP-Operationen lassen sich auch über eine Weboberfläche im Browser aufrufen.

1.2.3 Zettel

Table 15: Überblick

Source	**`zettel < https://github.com/hbz/zettel>`__**
Technik	Play Play 2.5 .4
Ports	9003 / 9103
Verzeichnis	/opt/regal/apps/zettel, /opr/regal/src/zettel
HTTP Pfad	/tools/zettel

Zettel ist ein Webservice zur Bereitstellung von Webformularen. Die Webformulare können über ein HTTP-GET geladen werden. Sollen existierende Daten in ein Formular geladen werden, so können diese Daten (1) als Form-encoded, (2) als JSON, oder (3) als RDF-XML über ein `HTTP POST` in das Formular geladen werden. Gleichzeitig kann spezifiziert werden, in welchem Format das Formular Daten zurückliefern soll.

Zettel verfügt über keine eigene Speicherschicht. Daten die über ein Formular erzeugt wurden, werden in der HTTP-Response zurückgeliefert. Zur Integration von Zettel in andere Applikationen wurde ein Kommunikationspattern entwickelt, das auf Javascript beruht. Das Zettel-Formular wird hierzu in einem IFrame in die Applikation eingebunden. Die Applikation muss außerdem ein Javascript einbinden, das auf bestimmte Nachrichten aus dem IFrame lauscht. Bei bestimmte Aktionen sendet das Zettel-Formular dann Nachrichten an die Applikation und erlaubt dieser darauf zu reagieren. Um Daten von Zettel in die Applikation zu bekommen, werden diese im HTML-DOM gespeichert und können von dort durch die Applikation entgegengenommen werden.

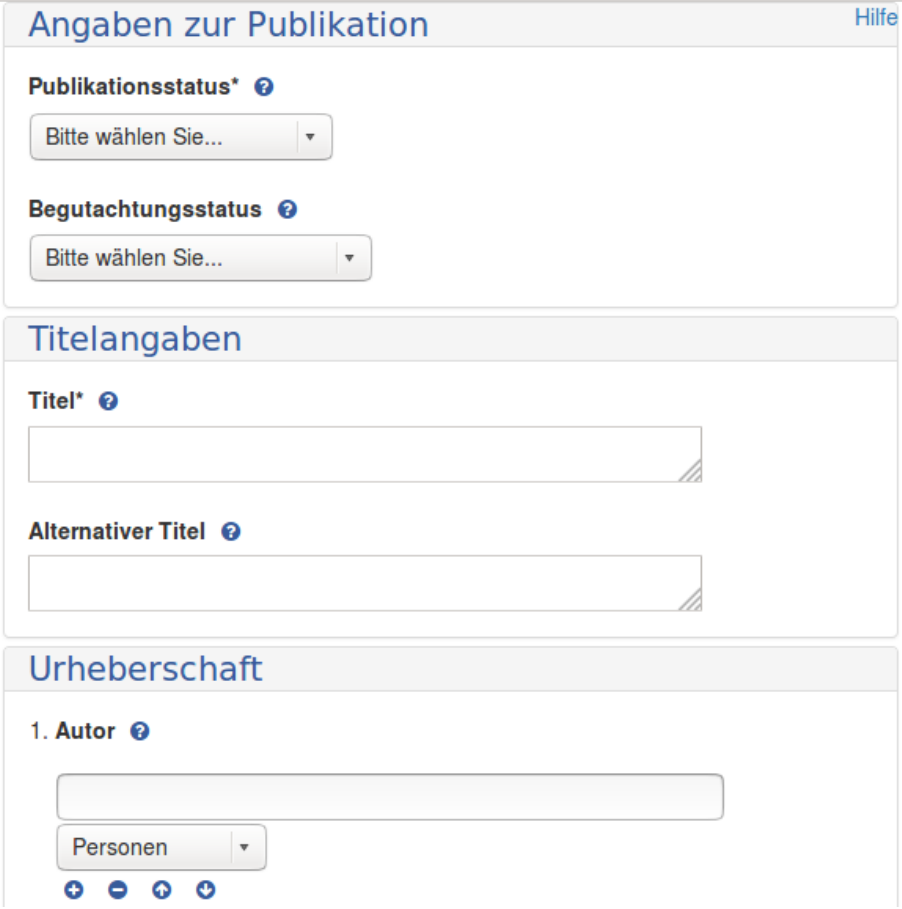


Fig. 5: Zettel Oberfläche

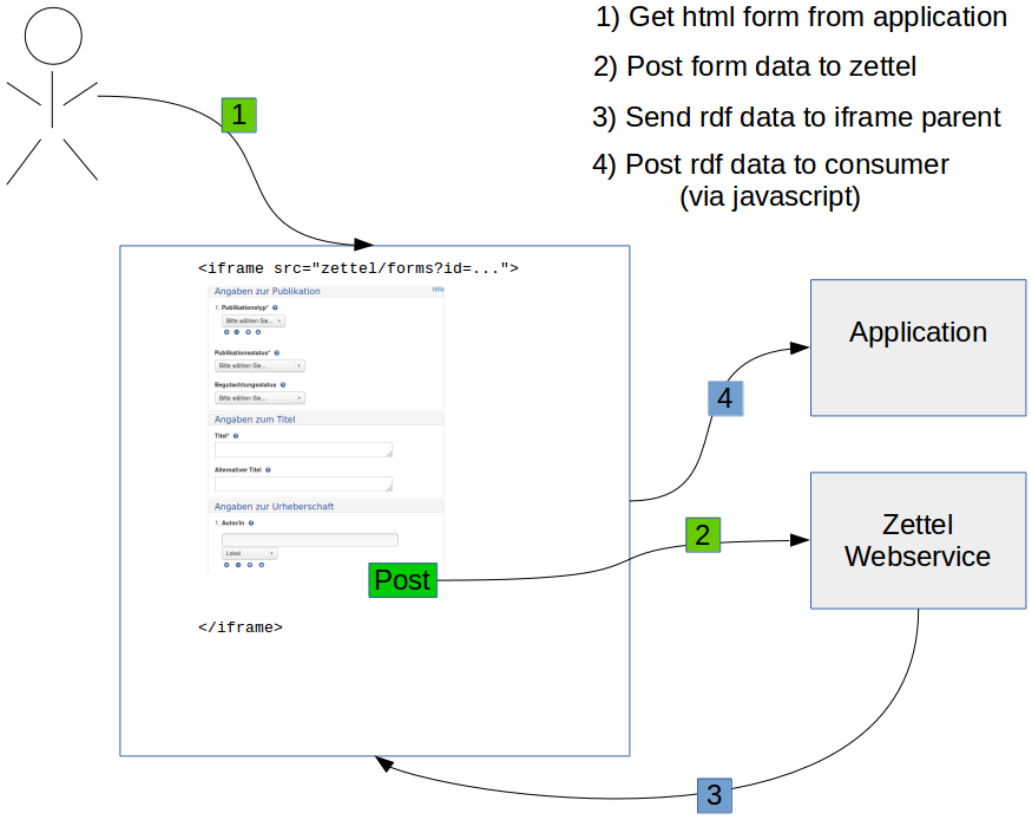


Fig. 6: Zettel Datenfluss

Konfiguration

Table 16: Dateien im /conf Verzeichnis

Datei	Beschreibung
application.conf	Die Datei enthält einen Eintrag zur Konfiguration von *Etikett*. Über einen weiteren Eintrag können "Hilfetexte" angelinkt werden. Die Hilfetexte müssen in einer statischen HTML abgelegt sein. Am Ende der Datei werden einige Limits deutlich über den Standard erhöht, damit die großen RDF-Posts auch funktionieren.
collectionOne.	Die Datei regelt den Inhalt eines Combo-Box widgets mit id collectionOne.
ddc.csv	Die Datei regelt den Inhalt eines Combo-Box widgets mit id ddc.
labels.json	Ein paar labels, falls keine Instanz von *Etikett* erreichbar ist.
logback.xml	Logger Konfiguration.
professionalGroup.cs	Die Datei regelt den Inhalt eines Combo-Box widgets mit id professionalGroup.
routes	Alle HTTP-Pfade übersichtlich in einer Datei

Die Applikation

Table 17: Das /app Verzeichnis

Package	Beschreibung
controlle	Es gibt nur einen Controller. Hier ist sowohl die Basisfunktionalität implementiert, als auch die Autocompletion-Endpunkte für die unterschiedlichen Widgets. Die Schnittstelle zu Abhandlung von Formulardaten ist recht generisch gehalten. Über eine ID wird das entsprechende Formular aus dem services.ZettelRegister geholt und das zugehörige Formular wird gerendert. Die Formular erhalten dabei unterschiedliche Templates (z.B. `views.Article`) und unterschiedliche Modelklassen (z.B. models.Article).
models	Das Model "Article" heißt aus historischen Gründen so. Tatsächlich können mittlerweile auch Kongressschriften und Buchkapitel darüber abgebildet werden (vermutlich wird sich der Name nochmal ändern). Das Model "Catalog" dient zum Import von Daten aus dem Aleph-Katalog (über Lobid). Mit ResearchData steht ein prototypisches Model zur Verarbeitung von Daten über Forschungsdaten zur Verfügung. Alle Models basieren auf einem einzigen "fetten" ZettelModel. Das ZettelModel enthält auch Funktionen zur De/Serialisierung in RDF und Json.
services	Hier werden verschiedene Hilfsklassen versammelt. Die Klasse ZettelFields enthält ein Mapping zur RDF-Deserialisierung.
views	Alle HTML-Sichten und die eigentlichen Formulare.

1.2.4 skos-lookup

Table 18: Überblick

Source	skos-lookup
Technik	Play Play 2.5 .8
Ports	9004 / 9104
Verzeichnis	/opt/regal/apps/skos-lookup, /opr/regal/src/skos-lookup
HTTP Pfad	/tools/skos-lookup

skos-lookup dient zur Unterstützung von *Zettel*. Der Webservice startet eine eingebettete Elasticsearch-Instanz und verfügt über eine Schnittstelle um SKOS-Daten in separate Indexe zu importieren und Schnittstellen zur Unterstützung von jQuery-Autocomplete- und Select2-Widgets aufzubauen. Auf diese Weise können auch umfangreichere Thesauri und Notationssysteme in den Formularen von *Zettel* fachgerecht angelinkt werden. *skos-lookup* unterstützt auch mehrsprachige Thesauri.

Fig. 7: SKOS-Lookup Beispiel 1

Fig. 8: SKOS-Lookup Beispiel 2

Konfiguration

Table 19: Dateien im /conf Verzeichnis

Datei	Beschreibung
application.conf	Hier wird der interne Elasticsearch-Index konfiguriert. Auch werden einige Speichereinstellungen erhöht. Damit auch große SKOS-Dateien geladen werden können, sollten auch die Java-Opts erhöht werden.
logback.xml	Logger Konfiguration
routes	Alle HTTP-Pfade übersichtlich in einer Datei
skos-context.jsc	Ein JSON-LD-Kontext zur Umwandlung von RDF nach JSON. (Origianl von: Jakob Voss)
skos-setting.jso	Settings zur Konfiguration des Elasticsearchindexse. (Original von: Jörg Prante)

Die Applikation

Table 20: Das /app Verzeichnis

Package	Beschreibung
controllers	Alles in einem Controller. Die API-Methoden liefern HTML und JSON, so dass man sie aus dem Browser, aber auch über andere Tools ansprechen kann.
elasticsearch	Eine embedded Elasticsearch. Dies hat den Vorteil, dass man eine aktuellere Version nutzen kann, als z.B. die *regal-api*.
services	Hilfsklassen zum Laden der Daten.
views	Ein Formular um neue Daten in die Applikation zu laden. Und ein Beispielformular zur Demonstration der Nutzung.

1.2.5 Thumby

Table 21: Überblick

Source	**`thumby < https://github.com/hbz/thumby>`__**
Technik	Play Play 2.2 .2
Ports	9001 / 9101
Verzeichnis	/opt/regal/apps/thumby, /opr/regal/src/thumby
HTTP Pfad	/tools/thumby

Thumby realisiert einen Thumbnail-Generator. Über ein HTTP-GET wird *Thumby* die URL eines PDFs, oder eines Bildes übergeben. Sofern die *Thumby* den Server kennt, wird es versuchen ein Thumbnail der zurückgelieferten Daten zu erstellen. Die Daten werden dauerhaft auf der Platte abgelegt und zukünftige Requests, die auf dasselbe Bild verweisen werden direkt aus dem Speicher von *Thumby* beantwortet.

Konfiguration

Table 22: Dateien im /conf Verzeichnis

Datei	Beschreibung
application.conf	Hier wird eine Whitelist gesetzt. Thumby verarbeitet nur URLs von den hier angegebenen Quellen. Hier wird auch der Pfad auf der Platte gesetzt, unter dem Thumby thumbnail-Daten ablegt.
routes	Alle HTTP-Pfade übersichtlich in einer Datei

Die Applikation

Table 23: Das /app Verzeichnis

Package	Beschreibung
controllers	Der Controller realisiert eine GET-Methode, über die Thumbnails erzeugt und zurückgegeben werden.
helper	Klassen zur Organisation des Speichers und zur Thumbnailgenerierung.
views	Es gibt eine Oberfläche mit einem Upload-Formular.

1.2.6 Deepzoomer

Table 24: Überblick

Source	DeepZoomService
Technik	**`Servlet 2.3 < https://download.oracle.com/otn-p ub/jcp/7840-servlet-2.3-spec-oth- JSpec/servlet-2_3-fcs-spec.ps>`__**
Ports	9091 / 9191
Verzeichnis	/opt/regal/tomcat-for-deepzoom/, /opr/regal/src/DeepZoomService

Der [DeepZoomService] kann als WAR in einem Application-Server deployed werden. Mit dem Deepzoomer können pyramidale Bilder erzeugt, gespeichert und über eine OpenSeadragon-konforme Schnittstelle abgerufen werden. Auf diese Weise kann in Regal eine Viewer-Komponente realisiert werden, die die Anzeige sehr großer, hochaufgelöster Bilder im Webbrowser unterstützt.

Konfiguration

Table 25: Dateien im /conf Verzeichnis

Datei	Beschreibung
deepzoomer.cfgf	Hier werden lokale Verzeichnisse, aber auch die Server-URLs, unter denen der Service öffentlich abrufbar ist, gesetzt.

1.2.7 regal-drupal

Table 26: Überblick

Source	regal-drupal
Technik	PHP 5
Ports	80 / 443
Verzeichnis	/opt/re gal/var/drupal/sites/all/modules/

Ein Drupal 7 Modul über das Funktionalitäten der *regal-api* angesprochen werden können. Das Modul bietet Oberflächen zur Konfiguration, zur Suche und zur Verwaltung von Objekthierarchien.

Die Applikation

Table 27: Verzeichnisstruktur

Verzeichnis	Beschreibung
edoweb	Hier ist der Code für die Oberflächen.
edoweb-field	Hier werden Felder für unterschiedliche RDF-Properties in der Drupal-Datenbank konfiguriert. Der Code ist größtenteils obsolet, da die Feldlogik nicht mehr benutzt wird.
edoweb_st	Hier sind die Zugriffe auf *regal-api* und *???* zu finden.

1.2.8 edoweb-drupal-theme

Table 28: Überblick

Source	edow eb-drupal-theme
Technik	PHP 5
Ports	80 / 443
Verzeichnis	/opt/r egal/var/drupal/sites/all/themes/

Eine Reihe von Stylsheets, CSS, Icons zur Gestaltung einer Oberfläche für den Server https://edoweb-rlp.de

1.2.9 zbmed-drupal-theme

Table 29: Überblick

Source	zb med-drupal-theme
Technik	PHP 5
Ports	80 / 443
Verzeichnis	/opt/r egal/var/drupal/sites/all/themes/

Eine Reihe von Stylsheets, CSS, Icons zur Gestaltung einer Oberfläche für den Server https://repository.publisso.de

1.2.10 openwayback

Repo: https://github.com/iipc/openwayback Servlet 2.5 .Überblick

Source	openwayback
Technik	Servlet 2.5
Ports	8091 / 8191
Verzeichnis	/o pt/regal/tomcat-for-openwayback/, /opr/regal/src/openwayback

Achtung: Es gibt einen am hbz entwickelten Branch. Dieser ist nicht auf Github.

Openwayback ist eine Webapplikation die im ROOT Bereich eines Tomcats installiert werden will. Sie kann Verzeichnisse mit WARC-Dateien indexieren und darauf eine Oberfläche zur Recherche und zur Navigation aufbauen.

1.2.11 heritrix

Heritrix ist ein Werkzeug zur Sammlung von Webseiten. Heritrix startet standalone als Webapplikation und bietet eine Weboberfläche zur Verwaltung von Sammelvorgängen an. Eingesammelte Webseiten werden als WARC-Dateien in einem bestimmten Bereich der Platte abgelegt.

1.2.12 wpull

Wpull ist ein Kommandozeilen-Wermzeug zur Sammlung von Webseiten. Mit WPull können WARC-Dateien erzeugt werden.

1.2.13 Fedora Commons 3

Fedora Commons 3 ist ein Repository-Framework. Für Regal wird vorallem die Speicherschicht von Fedora Commons 3 benutzt. Fedora-Commons legt alle Daten im Dateisystem (auch) ab. Mit den Daten aus dem Dateisystem lässt sich eine komplette Fedora-Commons 3 Instanz von grundauf neu aufbauen.

1.2.14 MySql

MySQL wir von Fedora, regal-api und etikett verwendet.

1.2.15 Elasticsearch 1.1

Elasticsearch ist eine Suchmaschine und wird von *regal-api* verwendet. Auch *regal-drupal* greift auf den Index zu.

1.2.16 Drupal 7

Über Drupal 7

1.2.17 Vagrant

Zur Veranschaulichung dieser Dokumentation wird ein Vagrant-Skript angeboten, mit dem eine Regal-Installation innerhalb eines VirtualBox-Images erzeugt werden kann.

Zur Installation kannst Du folgende Schritte ausführen. Die Kommandos beziehen sich auf die Installation auf einem Ubuntu-System. Für andere Betriebssyteme ist die Installation ähnlich.

Die VirtualBox hat folgendes Setup

- hdd 40GB
- cpu 2core
- ram 4096M

VirtualBox installieren

```
sudo apt-get install virtualbox
```

Vagrant installieren

```
cd /tmp
wget https://releases.hashicorp.com/vagrant/2.2.3/vagrant_2.2.3_x86_64.deb
sudo dpkg -i vagrant_2.2.3_x86_64.deb
```

Repository herunterladen

```
git clone https://github.com/jschnasse/Regal
cd Regal/vagrant/ubuntu-14.04
```

Eine JDK8 bereitstellen

Hierfür bitte ein JDK8-Tarball herunterladen und unter dem Namen `java8.tar.gz` in einem Verzeichnis `/bin` unterhalb des Vagrant-Directories bereitstellen.

```
mkdir bin
mv ~/downloads/jdk.... bin/java8.tar.gz
```

Geteiltes Entwicklungsverzeichnis

```
mkdir ~/regal-dev
```

Vagrant Guest Additions installieren

```
vagrant plugin install vagrant-vbguest && vagrant reload
```

Vagrant Host anlegen

Damit alle Dienste komfortabel erreichbar sind, muss in die lokale HOSTs Datei ein Eintrag für die Vagrant-Box erfolgen. Im Vagrantfile ist die IP `192.168.50.4` für die Box konfiguriert. Über die `FRONTEND` und `BACKEND` Einträge in der `variables.conf` ist der Servername als `regal.vagrant` definiert.

```
sudo printf "192.168.50.4 regal.vagrant api.regal.vagrant" >> /etc/hosts
```

Vagrant starten

```
vagrant up
```

Auf der Maschine einloggen

```
vagrant ssh
```

1.2.18 Server

Die Installation auf einem Server kann mit Hilfe des mitgelieferten Skriptes regal-install.sh erfolgen. Dazu muss analog zur Vagrant-Installation zunächst das `bin` Verzeichnis mit einem JDK aufgebaut werden. Danach erfolgt die Installation unter `/opt/regal` und mit einem Benutzer `regal` (vgl. `variables.conf`)

Hardware Empfehlung

- hdd >500GB
- cpu 8 core
- ram 32 G

Unterschiede zur Vagrant Installation

Auf dem Server empfehlen ich den fedora tomcat mit erweiterten Speichereinstellungen zu betreiben.

Dazu in `/opt/regal/bin/fedora/tomcat/bin` eine `setenv.sh` anlegen und folgende Zeilen hinein kopieren.

```
CATALINA_OPTS=" \
-Xms1536m \
-Xmx1536m \
-XX:NewSize=256m \
-XX:MaxNewSize=256m \
-XX:PermSize=256m \
-XX:MaxPermSize=256m \
-server \
-Djava.awt.headless=true \
-Dorg.apache.jasper.runtime.BodyContentImpl.LIMIT_BUFFER=true"

export CATALINA_OPTS
```

Entwicklung Java

1.2.19 In der VirtualBox

Hat man über *Vagrant* eine neue VirtualBox erzeugt und alle Konfigurationen wie beschrieben vorgenommen, kann man die VirtualBox zur Entwicklung nutzen. Da im Installationsprozess bereits Eclipse-Projekte der unter `/opt/regal/src` befindlichen Java-Applikationen erzeugt wurden, können die Projekte direkt aus dem "synced folder" unter `~/regal-dev` in eine Eclipse-IDE auf dem Host-System importiert werden.

Damit Änderungen am Code in der VirtualBox direkt sichtbar werden, sollte die Applikation zunächst im Develop-Mode neu gestartet werden. Dazu loggt man sich auf der VirtualBox mit `vagrant ssh` ein und stoppt zunächst den entsprechenden Service, z.B. `sudo service regal-api stop`. Anschließend navigiert man in das Source-Verzeichnis, z.B. `cd /opt/regal/src/regal-api`. Hier startet man die Applikation auf dem korrekten Port (im Zweifel unter `/opt/regal/apps/regal-api/conf/application.conf` nachschauen). Der Start im Develop-Mode erfolgt aus dem Verzeichnis der Applikation, mit z.B. `/opt/regal/bin/activator/bin/activator -Dhttp.port=9100`. Danach kann in die Kosole `run` eingegegeben werden. Die Applikation sollte nun unter dem entsprechenden Port (im Beispiel: 9100) antworten.

Leider funktioniert das Reloading zwischen Host-System und Guest-VirtualBox nicht richtig. D.h. nach Code-Änderungen im Host, muss auf der Virtualbox zunächst mit `Ctrl+D` und `run` neu gestartet werden, damit die Änderungen sichtbar werden.

1.2.20 Auf dem eigenen System

Die Javakomponenten können problemlos auch auf einem aktuellen Ubuntusystem entwickelt werden. Leider läuft die PHP/Drupal-Implementierung nicht unter neueren Ubuntusystemen. Für die lokale installation können die entsprechenden Funktionen aus dem `regal-install.sh` ausgeführt werden. Dazu einfach eine Kopie anlegen, entsprechend editieren und ausführen.

```
mkdir regal-install
cp -r path/to/Regal/vagrant/ubuntu-XX/* regal-install
cd regal-install
# Edit system user "vagrant" --> "your user"
```

(continues on next page)

(continued from previous page)

```
editor variables.conf
# put drupal stuff in comments
#
#  #installDrush
#  #installDrupal
#  #installRegalDrupal
#  #installDrupalThemes
#  #configureDrupalLanguages
#  #configureDrupal
#
editor regal-install.sh
```

1.2.21 Aktualisierung

Play-Applikationen

Die Aktualisierung der Regal-Komponenten erfolgt über Skripte. Die Aktualisierung funktioniert dabei so, dass der Quellcode der zu aktualisierenden Komponente unter `/opt/regal/src` per `git` auf den entsprechenden Branch gestellt wird. Danach wird ein neues Kompilat der Komponente erzeugt. Die aktuelle Konfiguration wird aus `/opt/regal/conf` genommen und es wird unter `/opt/regal/apps` eine neue lauffähige Version abgelegt.

Neue Versionen werden immer parallel zu alten Versionen gestartet und über einen Wechsel der Apachekonfiguration aktiviert. Erst danach wird die alte Version heruntergefahren.

Der komplette Aktualisierungsprozess erfolgt automatisch. Die alte Version bleibt immer auf dem Server liegen, so dass bei Bedarf wieder zurück gewechselt werden kann.

Tomcat-Applikation

Es wird ein `war`-Container erzeugt und im Tomcat `hot`-deployed.

Drupal-Module

Beinhaltet die Aktualisierung ein Datenbankupdate, so wird Drupal erst in den Wartungszustand versetzt (per drush oder über die Oberfläche). Danach wird die aktualisierte Version einfach per Git geholt. Bei Datenbankupdates wird noch ein Drupal-Updateskript ausgeführt.

Speicherschicht

Aktualisierungen von MySQL, Elasticsearch und Fedora gehen mit einer Downtime einher.

1.2.22 Verzeichnisse

Table 30: Verzeichnisstruktur

Verzeichnis	Beschreibung
/opt/regal	Außer Apache2, Elasticsearch und MySQL befinden sich alle Regal-Komponenten unter diesem Verzeichnis.
/opt/regal/apps	Die auf `Play` beruhenden Komponenten: `etikett fedora regal-a pi skos-lookup thumby zettel`
/opt/regal/bin	Fremdpakete wie activator, fedora, heritrix, python - weitere tomcats.
/opt/regal/conf	Die variables.conf und die application.conf wird von verschiedenen Komponenten verwendet.
/opt/regal/logs	Logfiles der Skripte und Cronjobs
/opt/regal/src	Alle Eigenentwicklungen oder im Quellcode modifizierten Komponenten.
/opt/regal/var	drupal und Datenverzeichnisse.

1.2.23 Ports

Table 31: Ports und Komponenten (typische Belegung)

Port	Komponente
80 /443	Apache 2
8080	fedora tomcat
9090	openwayback tomcat
9200	elasticsearch
9000/9100	regal-api
9001/9101	thumby
9002/9102	etikett
9003/9103	zettel
9004/9104	skos-lookup

1.2.24 Logs

Table 32: Logfiles

Komponente	Pfad
Apache	/var/log/apache2
Tomcat	/opt/regal/bin/fedora/tomcat/logs
Fedora	/opt/regal/bin/fedora/server/logs
Elasticsearch	/var/log/elasticsearch
regal-api	/opt/regal/apps/regal-api/logs
drupal	/var/log/apache2 #otherhosts ! und/var/log/apache2/error.log (hier ist auch die Debugausgabe)
MySql	/var/log/mysql
monit	/var/log/monit.log
regal-scripts	/opt/regal/logs

1.2.25 Configs

Table 33: Configfiles

Komponente	Pfad
Apache	/etc/apache2/sites-enabled
Tomcat	/opt/regal/bin/fedora/tomcat/conf
Fedora	/opt/regal/bin/fedora/server/conf
Elasticsearch	/etc/elasticsearch
regal-api	/opt/regal/conf enthält Konfigurationsvorschläge des Installers
regal-api	/opt/regal/apps/regal-api/conf
drupal	Konfig kann gut mit dem Tool drush überwacht werden
Elasticsearch Plugins	/etc/elasticsearch
oai-pmh	/opt/regal/ bin/fedora/tomcat/webapps/dnb-unr /WEB-INF/classes/proai.properties
monit	/etc/monit

1.2.26 Apache2

Table 34: Frontend Pfade

Komponente	HTTP-Pfad	Lokaler Pfad/Proxy
Drupal	/	/opt/regal/var/drupal
Alte Importe von Webarchiven	/webharvests	/data/webharvests
Täglich generierte Datei mit Kennziffern	/crawlreports	/o pt/regal/crawlreports

Table 35: API Pfade

Komponente	HTTP-Pfad	Lokaler Pfad/Proxy
Über wget erstellte Webarchive	/wget-data	/op t/regal/var/wget-data
Über wpull erstellte Webarchive	/wpull-data	/opt /regal/var/wpull-data
Über heritrix erstellte Webarchive	/heritrix-data	/opt/re gal/var/heritrix-data
OAI-Schnittstelle für die DNB	/dnb-urn	http://loc alhost:8080/dnb-urn$1
OAI-Schnittstelle	/oai-pmh	http://loc alhost:8080/oai-pmh$1
Deepzoomer	/deepzoom	http://loca lhost:7080/deepzoom$1
Openwayback privat	/wayback	http://l ocalhost:9080/wayback
Openwayback öffentlich	/weltweit	http://lo calhost:9080/weltweit
Thumby	/tools/thumby	http://localh ost:9001/tools/thumby
Etikett	/tools/etikett	http://localho st:9002/tools/etikett
Zettel	/tools/zettel	http://localh ost:9004/tools/zettel
Elasticsearch GET	/search	http://localhost:9200
Fedora	/fedora	http:// localhost:8080/fedora
JSON-LD Context	/ public/resources.json	http:/ /localhost:9002/tools /etikett/context.json
regal-api	/	h ttp://localhost:9000/
heritrix	/tools/heritrix	https://localhos t:8443/tools/heritrix

1.2.27 Matomo

Matomo wird einmal täglich per Cronjob mit Apache-Logfiles befüllt. Dabei erfolgt eine Anonymisierung. Die Logfiles verbleiben noch sieben Tage auf dem Server und werden dann annoynmisiert.

1.2.28 Monit

Das Tool Monit erlaubt es, den Status der Regal-Komponenten zu überwachen und Dienste ggfl. neu zu starten. Folgende Einträge können in /etc/monit/monitrc vorgenommen werden

```
check process apache with pidfile /var/run/apache2/apache2.pid
    start program = "/etc/init.d/apache2 start" with timeout 60 seconds
    stop program  = "/etc/init.d/apache2 stop"

check process regal-api with pidfile /opt/regal/apps/regal-api/RUNNING_PID
     start program = "/etc/init.d/regal-api start" with timeout 60 seconds
     stop program = "/etc/init.d/regal-api stop"

check process tomcat6 with pidfile /var/run/tomcat6.pid
     start program = "/etc/init.d/tomcat6 start" with timeout 60 seconds
     stop program = "/etc/init.d/regal-api stop"

check process elasticsearch with pidfile /var/run/elasticsearch.pid
     start program = "/etc/init.d/elasticsearch start" with timeout 60 seconds
     stop program = "/etc/init.d/elasticsearch stop"

check process thumby with pidfile /opt/regal/apps/thumby/RUNNING_PID
     start program = "/etc/init.d/thumby start" with timeout 60 seconds
     stop program = "/etc/init.d/thumby stop"

check process etikett with pidfile /opt/regal/apps/etikett/RUNNING_PID
     start program = "/etc/init.d/etikett start" with timeout 60 seconds
     stop program = "/etc/init.d/etikett stop"

check process zettel with pidfile /opt/regal/apps/zettel/RUNNING_PID
     start program = "/etc/init.d/zettel start" with timeout 60 seconds
     stop program = "/etc/init.d/zettel stop"
```

1.2.29 Scripts und Cronjobs

Für das Funktionieren von Regal sind einige regal-scripts sinnvoll. Die Skripte sind sämtlich unter Github zu finden.

https://github.com/edoweb/regal-scripts

Die folgenden Abschnitte zeigen ein typisches Setup.

OAI-Providing

Der OAI-Provider läuft nicht die ganze Zeit mit, da dies Probleme gemacht hat. Er wird nur für einen bestimmten Zeitraum angestellt und dann wieder ausgestellt. Auf diese Weise liefert die OAI-Schnittstelle tagesaktuelle Daten.

```
0 2 * * * /opt/regal/src/regal-scripts/turnOnOaiPmhPolling.sh
0 5 * * * /opt/regal/src/regal-scripts/turnOffOaiPmhPolling.sh
```

URN-Registrierung

Die URN-Registrierung erfolgt mit einem gewissen Verzug. Das dafür zuständige Skript überprüft daher zunächst das Anlagedatum der Ressource.

```
05 7 * * * /opt/regal/src/regal-scripts/register_urn.sh control  >> /opt/regal/regal-
↪scripts/log/control_urn_vergabe.log
1 1 * * * /opt/regal/src/regal-scripts/register_urn.sh katalog >> /opt/regal/regal-
↪scripts/log/katalog_update.log
1 0 * * * /opt/regal/src/regal-scripts/register_urn.sh register >> /opt/regal/regal-
↪scripts/log/register_urn.log
```

Katalog-Aktualisierung

Das System gleicht einmal am Tag Metadaten mit dem hbz-Verbundkatalog ab und führt ggf. Aktualisierungen durch.

```
0 5 * * * /opt/regal/src/regal-scripts/updateAll.sh > /dev/null
```

Matomo

Matomo wird mit Apache-Logfiles befüllt. Innerhalb von Matomo werden die Einträge annonymisiert.

```
0 1 * * * /opt/regal/regal-scripts/import-logfiles.sh >/dev/null
```

Logfile Annonymisierung

Apache-Logfiles werden sieben Tage unverändert aufbewahrt. Danach erfolgt eine Annonymisierung.

```
0 2 * * * /opt/regal/src/regal-scripts/depersonalize-apache-logs.sh
```

Webgatherer

Der Webgatherer prüft Archivierungsintervalle von Webpages und stößt bei Bedarf die Erzeugung eines neuen Snapshots/Version an.

```
0 20 * * * /opt/regal/src/regal-scripts/runGatherer.sh >> /opt/regal/regal-scripts/log/
↪runGatherer.log
# Auswertung des letzten Webgatherer-Laufs
0 21 * * * /opt/regal/src/regal-scripts/evalWebgatherer.sh >> /opt/regal/regal-scripts/
↪log/runGatherer.log
```

(continues on next page)

(continued from previous page)

```
# Crawl Reports
0 22 * * * /opt/regal/src/regal-scripts/crawlReport.sh >> /opt/regal/logs/crawlReport.log
```

Backup

MySQL und Elasticsearch

Der Elasticsearch-Index und die MySQL-Datenbanken werden täglich gesichert. Es werden Backups der letzten 30 Tage aufbewahrt. Ältere Backups werden von der Platte gelöscht.

```
0 2 * * * /opt/regal/src/regal-scripts/backup-es.sh -c >> /opt/regal/logs/backup-es.log␣
↪2>&1
30 2 * * * /opt/regal/src/regal-scripts/backup-es.sh -b >> /opt/regal/logs/backup-es.log␣
↪2>&1
0 2 * * * /opt/regal/src/regal-scripts/backup-db.sh -c >> /opt/regal/logs/backup-db.log␣
↪2>&1
30 2 * * * /opt/regal/src/regal-scripts/backup-db.sh -b >> /opt/regal/logs/backup-db.log␣
↪2>&1
```

Entwicklung

Für die Entwicklung an Regal empfiehlt sich folgende Vorgehensweise…

CHAPTER

TWO

INSTALLATION

2.1 Backend-Installation

2.1.1 Create linux user

Create user toscience with yast2. Generate encrypted password

```
head -c 300 /dev/urandom | tr -cd '[a-zA-Z0-9-_]' | head -c 16
```

Mit yast2 Home-Verzeichnis /opt/toscience und Gruppen **users, root** hinzufügen

Den User zu den sudoern hinzufügen. Zu /etc/sudoers eine Zeile hinzufügen:

```
vim /etc/sudoers
toscience    ALL = (root) /bin/su
wq
sudo su toscience
```

2.1.2 Systemprogramme maven, git, java, mariadb etc. installieren

maven und maven-local mit yast2 installieren.

```
zypper ref
zypper in git
```

CHAPTER

THREE

KOMPONENTEN

3.1 API

3.1.1 Preface

The Regal webservices documented by example `curl`-calls. Examples are assumed to work in the Vagrant-Environment that comes with this document.

3.1.2 Environment

Got to your server or to the Vagrant-Box, that comes with this document.

`vagrant ssh`

Prepare your environment to make the following `curl`-Calls work!

```
source /opt/regal/conf/variables.conf
export REGAL_API=http://$SERVER
export API_USER=edoweb-admin
```

to.science.api

https://github.com/hbz/to.science.api/blob/master/conf/routes

Create

Create a new resource

```
curl -i -u$API_USER:$PASSWORD -XPUT $REGAL_API/resource/regal:1234 -d'{"contentType":
↪"monograph","accessScheme":"public"}' -H'content-type:application/json'
```

Create a new hierarchy

```
curl -i -u$API_USER:$PASSWORD -XPUT $REGAL_API/resource/regal:1235 -d'{"parentPid":
↪"regal:1234","contentType":"file","accessScheme":"public"}' -H'content-
↪type:application/json'
```

Upload binary data

```
curl -u$API_USER:$PASSWORD -F"data=@$ARCHIVE_HOME/src/REGAL_API/test/resources/test.pdf;
↪type=application/pdf" -XPUT $REGAL_API/resource/regal:1235/data
```

Create User

```
curl -u$API_USER:$PASSWORD -d'{"username":"test","password":"test","email":"test@example.
↪org","role":"EDITOR"}' -XPUT $REGAL_API/utils/addUser -H'content-type:application/json'
```

Upload metadata

```
curl -XPUT -u$API_USER:$PASSWORD -d'<regal:1234> <dc:title> "Ein Test Titel" .' -H
↪"content-type:text/plain" $REGAL_API/resource/regal:1235/metadata2
```

Order Child Nodes

```
$ curl -XPUT -u$API_USER:$PASSWORD -d'["regal:2","regal:1249"]' $REGAL_API/resource/
↪regal:1/parts -H"Content-Type:application/json"
```

Ingest unmanaged content

Example address for external stored content, i.e. research data: `https://api.example.com/data/regal:1234/first_set/data.csv`

The base url and the default collection url are configured in the application.conf.

Currently only one level of subpaths is supported.

Table 1: URL parameter

parameter	default	description
collectionUrl	data	Path to the storage folder
subPath	-	optional: path of the subfolder, 'first_set' in above example
filename	-	bare filename, but with extension
resourcePid	<empty>	automatically assigned pid of the external resource

```
$ curl -XPOST -u$API_USER:$PASSWORD "$REGAL_API/resource/regal:1234/postResearchData?
↪collectionUrl=data&subPath=$dataDir&filename=$dateiname&resourcePid=$resourcePid" -H
↪"UserId=resourceposter" -H "Content-Type: text/plain; charset=utf-8";
```

Read

Read resource

html

```
curl $REGAL_API/resource/regal:1234.html
```

json

```
curl $REGAL_API/resource/regal:1234.json
curl $REGAL_API/resource/regal:1234.json2
```

rdf

```
curl $REGAL_API/resource/regal:1234.rdf
```

mets

```
curl $REGAL_API/resource/regal:1234.mets
```

aleph

```
curl $REGAL_API/resource/regal:1234.aleph
```

epicur

```
curl $REGAL_API/resource/regal:1234.epicur
```

datacite

```
curl $REGAL_API/resource/regal:1234.datacite
```

csv

```
curl $REGAL_API/resource/regal:1234.csv
```

wgl

```
curl $REGAL_API/resource/regal:1234.wgl
```

oaidc

```
curl $REGAL_API/resource/regal:1234.oaidc
```

Read resource tree

```
curl $REGAL_API/resource/regal:1234/all
```

```
curl $REGAL_API/resource/regal:1234/parts
```

Read binary data

```
curl $REGAL_API/resource/regal:1234/data
```

Read Webgatherer Conf

```
curl $REGAL_API/resource/regal:1234/conf
```

Read Ordering of Childs

```
curl $REGAL_API/resource/regal:1234/seq
```

Read user

```
not implemented
```

Read Adhoc Linked Data

```
curl $REGAL_API/adhoc/uri/$(echo test |base64)
```

Update

Update Resource

Update Metadata

```
curl -s -u$API_USER:$REGAL_PASSWORD -XPOST $REGAL_API/utils/updateMetadata/regal:1234 -H
↪"accept: application/json"
```

Add URN

```
POST /utils/lobidify
```

```
POST /utils/addUrn
```

```
POST /utils/replaceUrn
```

Enrich

```
POST /resource/:pid/metadata/enrich
```

Delete

Delete resource

```
curl -u$API_USER:$REGAL_PASSWORD -XDELETE "$REGAL_API/resource/regal:1234";echo
```

Purge resource

```
curl -u$API_USER:$REGAL_PASSWORD -XDELETE "$REGAL_API/resource/regal:1234?purge=true";
↪echo
```

Delete part of resource

```
curl -u$API_USER:$REGAL_PASSWORD -XDELETE $REGAL_API/resource/regal:1234/seq
```

```
curl -u$API_USER:$REGAL_PASSWORD -XDELETE $REGAL_API/resource/regal:1234/metadata
```

```
curl -u$API_USER:$REGAL_PASSWORD -XDELETE $REGAL_API/resource/regal:1234/metadata2
```

```
curl -u$API_USER:$REGAL_PASSWORD -XDELETE $REGAL_API/resource/regal:1234/data
```

```
curl -u$API_USER:$REGAL_PASSWORD -XDELETE $REGAL_API/resource/regal:1234/dc
```

Delete user

```
not implemented
```

Search

Simple Search

```
GET /find
```

```
GET /resource
```

Facetted Search

Search for field

Misc

Load metadata from Lobid

```
curl -u$API_USER:$PASSWORD -XPOST "$REGAL_API/utils/lobidify/regal:1234?
↪alephid=HT018920238"
```

Reread Labels from etikett

```
curl -u$API_USER:$PASSWORD -XPOST $REGAL_API/context.json
```

Reindex resource

```
curl -u$API_USER:$PASSWORD -XPOST $REGAL_API/utils/index/regal:1234 -H"accept:␣
↪application/json"
```

labels

https://github.com/hbz/to.science.labels/blob/master/conf/routes

Create

Add Labels to Database

```
curl -u$API_USER:$PASSWORD -XPOST -F"data=@$ARCHIVE_HOME/src/REGAL_API/conf/labels.json"␣
↪-F"format-cb=Json" $REGAL_API/tools/etikett -i -L
```

Add Label

Read

```
curl "$REGAL_API/tools/etikett" -H"accept: application/json"
```

Read Etikett

```
curl $REGAL_API/tools/etikett?url=http%3A%2F%2Fpurl.orms%2Fissued -H"accept: application/
↪json"
```

Update

Delete

Delete Cache

```
curl -XDELETE -u$API_USER:$PASSWORD $REGAL_API/tools/etikett/cache
```

Misc

Forms

https://github.com/hbz/to.science.forms/blob/master/conf/routes

Create

Create RDF-Metadata from Form-Data

Read

Read HTML-Form

Search

to.science.thumbs

https://github.com/hbz/thumby/blob/master/conf/routes

Read

```
curl -XGET "$REGAL_API/tools/thumby?url=https://www.gravatar.com/avatar/
↪5fefc19b7875e951c7ea9bfdfc06676d&size=200"
```

skos-lookup

https://github.com/hbz/skos-lookup/blob/master/conf/routes

Create

Create new Index

```
curl -i -X POST -H "Content-Type: multipart/form-data" $REGAL_API/tools/skos-lookup/
↪upload -F "data=@/tmp/skos-lookup/test/resources/agrovoc_2016-07-15_lod.nt.gz" -F
↪"index=agrovoc_test" -F"format=NTRIPLES"
```

Read

```
curl -XGET '$REGAL_API/tools/skos-lookup/autocomplete?lang=de&q=Erdnus&
↪callback=mycallback&index=agrovoc_test'
```

Search

```
curl $REGAL_API/tools/skos-lookup/search?q=http%3A%2F%2Faims.fao.org%2Faos%2Fagrovoc%2Fc_
↪13551&lang=de&index=agrovoc
```

Complex Example of hierarchical content

The newly created resource should meet the following requirements:

- publishScheme and accessScheme should be private
- Metadata are in LRMI Schema
- Resource should be assigned to an existing user in the Drupal frontend

Structure of the Resource:

```
orca:50
├── lrmiData
└── orca:51
    └── document.pdf
```

For the below curl command to work from your local computer it is convenient to put some often used data into environment variables. Prepare a simple textfile e.g. `example` with the following content. The `DRUPAL_USERID` is the numeric id which is automatically assigned to the user account by the Drupal CMS.

```
export TOSCIENCE_API=https://api.example.com
export DRUPAL_USERID="2"
export API_USER=toscience-admin
export PASSWORD=***********
```

Make the variables available by sourcing the file:

```
$ source example
```

Creating resource

Initially we create a yet empty resource with the desired accessScheme, publishSchem and user id:

```
$ curl -i -u$API_USER:$PASSWORD -XPUT $TOSCIENCE_API/resource/orca:50 -d'{"contentType":
↪"researchData","accessScheme":"private", "publishScheme":"private", "isDescribedBy":{
↪"createdBy":"'"$DRUPAL_USERID"'"}}' -H'Content-type:application/json' ; echo
```

The Metadata are given in a special LRMI-Format and passed to a didicated endpoint:

```
$ curl -i -u$API_USER:$PASSWORD -XPOST $TOSCIENCE_API/resource/orca:50/lrmiData  --data-
↪binary '@lrmi.json' -H'Content-Type:application/json;charset=utf-8'; echo
```

The data are stored in a separate resource of contentType `file`. At this point there is no relation between the two newly created resources:

```
$ curl -i -u$API_USER:$PASSWORD -XPUT $TOSCIENCE_API/resource/orca:51 -d'{"contentType":
↪"file","accessScheme":"private", "publishScheme":"private", "isDescribedBy":{"createdBy
↪":"'"$DRUPAL_USERID"'"}}' -H'Content-type:application/json' ; echo
```

Adding the actual data, a pdf-file in this case:

```
$ curl -i -u$API_USER:$PASSWORD -XPUT $TOSCIENCE_API/resource/orca:51/data -F
↪"data=@document.pdf;type=application/pdf" ; echo
```

In a final step we tell the data resource about it's parent resource:

```
$ curl -i -u$API_USER:$PASSWORD -XPUT $TOSCIENCE_API/resource/orca:51 -H'Content-
↪Type:application/json;charset=utf-8' -d'{"parentPid":"orca:50","contentType":"file"}' ;
↪ echo
```

Retrieving the resource

Reading the metadata in standard json format:

```
$ curl -i -u$API_USER:$PASSWORD -XGET $TOSCIENCE_API/resource/orca:51.json2 ; echo
```

The LRMI Metadata are again available via the dedicated endpoint:

```
$ curl -i -u$API_USER:$PASSWORD -XGET $TOSCIENCE_API/resource/orca:51/lrmiData ; echo
```

Downloading the data

```
$curl -i -u$API_USER:$PASSWORD -XGET $TOSCIENCE_API/resource/orca:51/data  --output data.
↪pdf; echo
```

3.2 Core

3.2.1 Why to.science.core?

to.science.core is thought to be the forthcoming core library of the Toolbox Open Science Environment. For keeping usage simple and allowing a widespread usage, **to.science.core** comes as a Java-Library provided either as jar-file with dependencies (via github release) or as Library provided by one of the Maven Central Repositories. The latter allows the usage within Maven or sbt-Project for instance.

3.2.2 Where to find to.science.core?

Bring **to.science.core** into your project by popular dependency management is like this:

Maven

Copy this code into the dependencies section in the projects pom.xml file

```
<dependency>
  <groupId>io.github.hbz</groupId>
  <artifactId>to.science.core</artifactId>
  <version>1.3.6</version>
</dependency>
```

sbt

Copy this code into the dependencies section in the projects build.sbt file

```
libraryDependencies += "io.github.hbz" % "to.science.core" % "1.3.6"
```

Since **to.science.core** is available from Maven Central Repository this is all you need to do if you're using a Build-Tool with dependency management.

3.2.3 I'd like to use the jar-Library in my project without any matured build tools

Please find the latest version of **to.science.core** within the release section of Github.

3.2.4 Use to.science.core from console

As a matter of fact there is nothing very exiting you can do with **to.science.core**. **to.science.core** will only provide some background services for the Toolbox. But if you keen to do something with it you can download the latest jar File with the dependencies and run the ConsoleMapper.class. Actualy it is only performing a conversion (mapping) from a file with AMB metadata to an output of the to.science data model (TOS). Java 1.8 JRE is required to run the ConsoleMapper:

```
$ java -jar to.science.core-VERSION-jar-with-dependencies.jar
```

The ConsoleMapper will lead you through the process from mapping a AMB File to the internal Toolbox json Format.

```
***-------------------------ConsoleMapper-----------------------------***
***     ConsoleMapper is part of the to.science.core library by hbz     ***
***   Maps your AMB file to the to.science json based data model (TOS)  ***
***-------------------------------------------------------------------***
```

It will as you for the submitters name and the submitters email address as this is not part of the AMB file but required by TOS.

```
Please set submitters name (Jane Doe)>
Please set submitters mail address (doe@eexample.org)>
```

The embraced testing values will be used if you return each line without typing. The ConsoleMapper succeed with asking you if you like to map any own AMB file from your filesystem or just run the mapping process with the included example file.

```
Load our own AMB file for mapping? ((y)es)>
Please provide your AMB File relative to our current directory or with an absolute path
your current directory is /home/myhome

AMB File location >
```

The ConsoleMapper starts to map the AMB File and imports the submitters name and email into TOS:

```
"isDescribedBy": {
"submittedBy": "Jane Doe",
"submitterByEmail": "doe.example.org",
"@id": "orca:3fc1c517-2667-403a-b13c-9c4bb83064a6",
"describes": "orca:3fc1c517-2667-403a-b13c-9c4bb83064a6"
},
"department": [
{
"prefLabel": "Physik",
"@id": "https://w3id.org/kim/hochschulfaechersystematik/n0128"
}
```

CHAPTER

FOUR

ENTWICKLUNG

4.1 API-Entwicklung

4.1.1 JSON in Java-Klassen überführen

Mit der Klasse JsonLDMapper steht ein generischer Ansatz zum Einlesen des lobid-JSON (oder anderer JSON-Formate) zur Verfügung, der die im JSON liegenden Metadaten in einer einheitlichen Weise für die Verarbeitung zugänglich macht.

Die Einrichtung der Klasse JsonLDMapper verfolgt das Ziel, kommende Änderungen am Datenmodell vom lobid mit möglichst wenig Aufwand in das Repository übernehmen zu können.

Grundlage bilden die von JSON unterstützten Datentypen. Die in JSON verwendeten Datentypen werden zunächst konzeptuell auf drei Typen reduziert.

- Object
- Array of Values als ArrayList<Hashtable<String,String>>
- Key/Value-Paare als Hashtable<String,String>

Der Datentyp Object wird als Container-Element für weitere Datentypen verwendet und rekursiv bis zu den elementaren Datentypen Array of Values und key/value-Paare aufgelöst. Die dabei verarbeitete Pfad-Struktur wird in der Java-Notation abgebildet und in einem String abgelegt..

Für alle auf den beiden Datentypen Array of Values und Key/Value-Paare aufbauenden Objekte bietet die Mapper-Klasse vereinheitlichte Instanzen mit analogen Zugriffsmethoden an. Der JsonLDMapper bietet jeweils die Methode getElement(Pfad), die transparent ArrayList<Hashtable<String,String> zurückliefert.

Über die Iteration über die jeweilige ArrayList stehen damit entweder zusammengehörende Key/Value zur Verfügung, oder die einzelnen Values eines Arrays of Value in Form des Array-Bezeichners aus dem JSON und des jeweiligen Wertes.

Beispiele

Aus dem Array von Literalen “title”

```
{record : {title: ["Ausdrücke in Java", "Java Expressions", "Expression de Java"]}}
```

erhält man durch mit dem JsonLDMapper

```
JsonLDMapper jMapper = new JsonLDMapper(JsonNode);
ArrayList<Hashtable<String,String> title = jMapper.getElement("root.record.title");
```

eine **ArrayListe** die aus drei Key/Value-Paaren besteht:

```
title = "Ausdrücke in Java"
title = "Java Expressions"
title = "Expression de Java"
```

Aus dem aus zwei Key/Value-Paaren bestehenden Objekt “creator”

```
{record : {creator: {
        prefLabel : "Loki Schmidt",
        @id : "https://orcid.org/000-000-000" }
}}
```

erhält man durch den gleichen Aufruf:

```
JsonLDMapper jMapper = new JsonLDMapper(JsonNode);
ArrayList<Hashtable<String,String> title = jMapper.getElement("root.creator");
```

eine **ArrayListe** die aus zwei Key/Value-Paaren besteht:

```
prefLabel = "Loki Schmidt"
@id : "https://orcid.org/000-000-000"
```

Damit der in Json verwendete Datentyp weiterhin eindeutig unterscheiden werden kann, besitzt die JsonLDMapper-Klasse zusätzlich die Methoden isArray() und isObject().

```
JsonLDMapper jMapper = new JsonLDMapper(JsonNode);
boolean test = jMapper.getElement("\root.creator\").isArray();
```

4.1.2 Neues Metadaten-Format in die OAI-Schnittstelle integrieren

Die Integration eines neuen Metadatenformats in die OAI-Schnittstelle umfasst Aktivitäten an mehreren Stellen.

1. Java-Klassen erweitern und anpassen
2. Konfiguration der regal-api und des OAI-Providers anpassen
3. Testen der Schnittstelle

Java-Klassen erweitern und anpassen

Für die Integration eines neuen Metadaten-Formats in die OAI-Schnittstelle sind die folgenden Dateien relevant.

- regal-api.app.helper.oai/OaiDispatcher.java
- regal-api.app.actions/Transform.java
- regal-api.app.controllers/Resource.java

In diesen drei Klassen müssen an mehreren Stellen Anpassungen, bzw. Erweiterungen des Codes vorgenommen werden, damit das Mapping und die Erstellung eines Metadaten-Stroms im System ausgelöst und gesteuert wird.

In der Datei OaiDispatcher.java muss ein zusätzlicher Transformer-Aufruf generiert werden und eine neue Methode addNeuesFormatTransformer erstellt werden.

```
private static void addNeuesFormatTransformer(Node node) {
  String type = node.getContentType();
    if ("public".equals(node.getPublishScheme())) {
      if ("monograph".equals(type) || "journal".equals(type)
        || "webpage".equals(type) || "researchData".equals(type)
        || "article".equals(type)) {
          node.addTransformer(new Transformer("neuesFormat"));
      }
    }
  }
```

Ebenso muss in die Methode addUnknownTransformer eine zusätzliche If-Abfrage integriert werden.

```
private static void addUnknownTransformer(List<String> transformers,
  Node node) {
  if (transformers != null) {
    for (String t : transformers) {
      if ("oaidc".equals(t))
        continue; // implicitly added - or not allowed to set
  [...]
      if ("neuesFormat".equals(t))
        continue; // implicitly added - or not allowed to set
      node.addTransformer(new Transformer(t));
    }
  }
}
```

In der Methode initContentModels(String namespace)ist dann noch ein zusätzlicher Block transformers.add einzutragen.

```
transformers.add(new Transformer(namespace + "neuesFormat", "neuesFormat",
                internalAccessRoute + "neuesFormat"));
```

Die Datei Transform muss anschließend um eine Methode neuesFormat erweitert werden. Diese Methode wird später über eine, in der Datei Resource.java definierte ApiOperation “asNeuesFormat” als Restful-Request aufgerufen. Die ApiOperation muss entsprechend auch angelegt werden.

Das Mappen und die Erzeugung eines Metadatenstroms wurde in der Vergangenheit über unterschiedliche Wege umgesetzt, bei denen ebenfalls mehrere Klassen und ggf. ScalaViews beteiligt sind.

Im Package helper.oai wird ein neuer Mapper angelegt, über den die im lobid V2-Format zur Verfügung gestellten Metadaten in das neue Format gemappt werden. Bisher kamen dafür die Klassen ObjectMapper aus der Jackson Library, models.Pair und entweder ein Datenmodell plus Mapper oder eine Record Klasse zum Einsatz.

Innerhalb des Packages view.oai mussten bei der Nutzung eines Datenmodells und eines Mappers zusätzlich die Klassen NeuesFormat.scala.html und NeuesFormatView.scala.html angelegt werden.Diese steuern das Parsing und die Darstellung des neuen Formats über die Scala-Infrastruktur. Im Unterschied dazu erzeugen die Record-Klassen String-Representationen eines XML-Datenstroms.

Um die Umsetzung der neuen Formate zu vereinheitlichen, wurde mehrere neue Klassen eingeführt, die einen strukturierten Zugriff auf das existierende (und künftige) lobid-JSON-Format ermöglichen sollen. Aktuell gibt es hier noch verschiedene Issues bei der Verarbeitung komplexer Strukturen aus Arrays und Objektelementen, die aber gelöst werden sollen. Um strukturiert zuzugreifen, sollte die Klasse regal-api.app.helper.oai.JsonLDMapper verwendet werden. Damit wird auch das Anlegen neuer ScalaViews obsolet.

Konfiguration der regal-api und des OAI-Providers anpassen

Damit das als Dissemination* angelegte neue Format über die regal-api abgefragt werden kann, muss in der Datei conf/routes eine entsprechende Konfigurationszeile erstellt werden.

```
GET /resource/:pid.openaire     controllers.Resource.asOpenAire(pid, validate : Boolean ?
↪= false)
```

Mit dieseem Eintrag wird eine Verbindung zwischen der entsprechenden Java-Methode und dem über das Play Framework stattfindenden Aufruf über eine HTTP-Methode erreicht.

Wie zu sehen ist, wird hier auch bestimmt, ob das erstellte Objekt normalerweise gegen eine xsd-Datei validiert werden soll. Im Beispile ist das nicht der Fall: validate : Boolean ?= false. In der Datei proai.properties müssen die mit der OAI-Schnittstelle zusammenhängenden Konfigurationen angepasst werden. Die Datei wird direkt im entpackten Applikation-Container angepasst.

```
################################################
# Fedora Driver: Metadata Format Configuration #
################################################
# Metadata formats to make available.
driver.fedora.md.formats = oai_dc epicur mabxml-1 mets rdf oai_wgl oai_openaire
[...]
driver.fedora.md.format.oai_ore.loc = http://www.w3.org/2000/07/rdf.xsd

driver.fedora.md.format.oai_openaire.loc = https://www.openaire.eu/schema/repo-lit/4.0/
↪openaire.xsd

[...]

driver.fedora.md.format.oai_ore.uri = http://www.w3.org/1999/02/22-rdf-syntax-ns#

driver.fedora.md.format.oai_openaire.uri = http://namespace.openaire.eu/schema/oaire/

[...]

driver.fedora.md.format.oai_dc.dissType = info:fedora/*/CM:oaidcServiceDefinition/oaidc

driver.fedora.md.format.oai_openaire.dissType = info:fedora/*/
↪CM:openaireServiceDefinition/openaire
```

Testen der Schnittstelle

Die OAI-Schnittstelle ist über die URL http://api.ellinet-dev.hbz-nrw.de/oai-pmh/ oder analog bei edoweb-test erreichbar. Der neue ServiceDisseminator kann über die regal-api aufgerufen werden, wenn der in der routes Datei deklarierte Pfad entsprechend aufgerufen wird. Obwohl GET als Methode deklariert ist, funktioniert jedoch nur der Aufruf mittels POST. Deshalb kommt cUrl zum Einsatz: curl -XGET -uedoweb-admin localhost:9000/resource/frl%3A6402576.openaire

4.1.3 Anlegen neuer Metadaten-Felder in der API

Vorgehen

Das Anlegen neuer Metadatenfelder in der to.science.api erfordert verschiedene Schritte und ist die Grundlage für die anschließende (optionale) Einbindung dieser Felder in to.science.forms.

1. Datei labels.json in to.science.api und to.science.labels erweitern und deployen (Ja wirklich :-( )
2. Notwendige Anpassungen in den für das Mapping zuständigen Java-Klassen (JsonMapper.java und LRMIMapper für ORCA)
3. Testen der Änderungen

labels.json erweitern

Damit die verschiedenen Mappings zwischen Formaten wie JSON und RDF, N-Triples etc. funktionieren, muss in der labels.json ein entsprechende JSON-Abschnitt für das neue Feld angelegt werden.

Beispiel neues Feld academicDegree:

```
{
        "uri": "https://d-nb.info/standards/elementset/gnd#academicDegree",
        "comment": "",
        "label": "Akademischer Grad",
        "icon": "",
        "name": "academicDegree",
        "referenceType": "String",
        "container": "@list",
        "weight": "5",
        "type": "CONTEXT",
        "multilangLabel": {}
},
```

“name” und “uri”:

1. Das Literal **“name”** muss den Variablen-Namen enthalten. Hieran war zunächst die Exposition der Variablen zu to.science.forms gescheitert.
2. Der Name muss **vermutlich** dem in der ResourceUri stehenden veränderlichen Teil entsprechen.

```
"uri": "https://d-nb.info/standards/elementset/gnd#academicDegree",
"name": "academicDegree",
```

“referenceType” und “container”:

1. Die Literale **“referenceType”** , **“container”** enthalten vermutlich Informationen über die Typisierung der Variablen. Hierbei wird bei container das JSON-LD Schlüsselwort @list verwendet. Es gibt 13 Schlüsselwörter.[1]
   Da das jedoch nicht dokumentiert ist, muss man sich Beispiele für verschiedene Variablentypen in der labels.json heraussuchen, probieren und Daumen drücken… (seufz).

```
"referenceType": "String",
"container": "@list",
```

2. Das Literal **“type”** verweist vermutlich überwiegend auf das Schlüsselwort @context. Sollte vermutlich nicht geändert werden.

```
"type": "CONTEXT",
```

Alle labels.json ersetzen:

1. Sowohl die labels.json von *to.science.api* als auch die von *to.science.labels* müssen erweitert werden. Wie die beiden interferieren ist bisher ziemlich unklar.
2. Beide Dateien müssen manuell in das jeweilige conf-Verzeichnis im installierten Modul kopiert werden. Sie werden nicht automatisch aktualisiert
3. to.science.api und to.science.labels müssen neu gestartet werden.

Erweitern der Java-Klasse JsonMapper.Java

Damit das neue Feld in die verschiedenen Metadaten-Formate gemappt werden kann, ist zumindest die Erweiterung der Klasse JsonMapper.Java notwendig

```
to.science.api.helper/JsonMapper.Java
```

Welche Methoden konkret angepasst werden müssen, kann hier nicht pauschal gesagt werden.

Für ORCA ist zusätzlich die Klasse LRMIMapper zu erweitern, damit die neuen Felder aud dem Datenstrom lrmiData gelesen und auch wieder dort hin geschrieben werden können.

```
to.science.api.helper/LRMIMapper.Java
```

Hintergrund

Die Vereinbarung des neuen Metadaten-Felds in `labels.json` ist notwendig, weil die `labels.json` innerhalb der to.science-Komponenten die Ergänzung des neuen Feldes um den für Mappings und Konvertiereungen von JSON-Dateien notwendigen LinkedData-Teil übernimmt. Damit kann die to.science.api an bestimmten API-Calls JSONLD ausliefern.[1]

[1] https://de.wikipedia.org/wiki/JSON-LD

4.1.4 Anlegen neuer Metadaten-Felder in FORMS

Voraussetzungen

1. Das neue Metadaten-Feld muss zunächst in den beiden Dateien `labels.json` von to.science.api und to.science.labels definiert werden (siehe vorherige Seite).

2. Übernahme der Definition des neuen Feldes aus den `labels.json` in die `labels.json` von to.science.forms

3. Kopieren der labels.conf in das installationsverzeichnis von to.science.forms

ja wirklich...

Vorgehen

In den folgenden Java-Klassen müssen zunächst Erweiterungen in Form von Variablen-Deklarationen vorgenommen werden, damit das neue Feld auch in den Formularen funktioniert.

```
to.science.forms.services.ZettelFields.Java
to.science.forms.model.ZettelModel.Java
```

1. In `ZettelFields` werden zunächst Verweise auf die Inhalte der `labels.conf` erzeugt. Dafür werden "Etikett"-Instanzen angelegt. Ist das neue Feld wie im Beispiel `academicDegree` als wiederholbares Literal angelegt, werden folgende Deklarationen benötigt: Das ZF am Ende der Variablen-Deklaration weißt darauf hin, dass es sich um ein "ZettelField" handeln soll.

```
public static Etikett academicDegreeZF =
 ZettelHelper.etikett.getEtikett("https://d-nb.info/standards/elementset/gnd
↪#academicDegree");
public static Etikett academicDegreeIndexZF =
 ZettelHelper.etikett.getEtikett("http://hbz-nrw.de/regal#academicDegreeIndex");
```

2. In der `ZettelModel.Java` werden ebenfalls neue Variablen-Deklarationen als erzeugt

```
public abstract class ZettelModel {

  private List<String> academicDegree = new ArrayList<>();
  private String academicDegreeIndex;
```

3. Die neuen Variablen benötigen `GETTER, SETTER` und im Fall der `ArrayList` eine add-Methode:

```
public abstract class ZettelModel {

public void setAcademicDegree(List<String> AcademicDegree ) {
  this.academicDegree = academicDegree;
}

public List<String> getAcademicDegree(){
  return this.academicDegree;
}

public String getAcademicDegreeIndex() {
  return academicDegreeIndex;
```

(continues on next page)

(continued from previous page)

```
}

public void setAcademicDegreeIndex(String AcademicDegreeIndex) {
  this.academicDegreeIndex = academicDegreeIndex;
}
```

4. Anschließend werden die in `ZettelField` deklarierten Variablen importiert. Ggf. wäre es hier ziemlich sinnvoll, sie nicht einzeln zu importieren?

```
import static services.ZettelFields.academicTitleZF;
import static services.ZettelFields.academicTitleIndexZF;
```

5. Sie werden in der Methode `getMappingForDeserialization` benötigt und verbinden die Felder mit den ZettelField-Variablen:

```
protected Map<String, Consumer<Object>> getMappingForDeserialization() {
        String regalApi = Play.application().configuration().getString("regalApi");
        Map<String, Consumer<Object>> dict = new LinkedHashMap<>();

        [...]

        dict.put(academicDegreeZF.uri, (in) -> addAcademicDegree((String) in));
        dict.put(academicDegreeIndexZF.uri, (in) -> setAcademicDegreeIndex((String) in));

        [...]

}
```

4.1.5 Metadaten-Felder im Formular zugänglich machen

Voraussetzungen

Die vorhergehenden Erweiterungen der labels.json und Java-Klassen entsprechend der Abschnitte

- "Anlegen neuer Metadaten in der API" und
- "Anlegen neuer Metadaten in FORMS"

wurden erledigt.

Vorgehen

Derzeit gibt es zwei Inhaltstypen, für die to.science.forms ein umfangreiches Formular in einem iFrame ausliefert. Das sind "Artikel" und Forschungsdaten". Die Inhaltstypen "Monografie" und "Serie" werden durch ein ganz rudimentäres Formular erfasst, dass nur eine HT-Nummer aus dem hbz-Katalog erfragt und diese zum abholen des Titelsatzes aus dem Katalog über die lobid2-API verwendet.[1]

Die Beschreibung bezieht sich hier auf den Inhaltstyp `ResearchData` ("Forschungsdaten"), der exemplarisch für die notwendigen Erweiterungen steht.

[1] https://lobid.org

1. In der dem Formular zugeordneten Model-Klasse wird für das neue Feld eine get-Methode geschrieben. Wir verwenden hier die Klasse `to.science.forms.model.ResearchData.Java` .[2]

```
/**
* @return a map that can be used in an html select
*/
public static LinkedHashMap<String, String> getAcademicDegreeMap() {
  LinkedHashMap<String, String> map = new LinkedHashMap<>();
  map.put(null, "Bitte wählen Sie...");
  GenericPropertiesLoader GenProp = new GenericPropertiesLoader();
  map.putAll(GenProp.loadVocabMap("AcademicDegree-de.properties"));
  return map;
}
```

2. In einem der ScalaTemplates muss anschließend die zuvor deklarierte get-Methode aufgerufen werden. Wir nehmen das Template `creatorWidget.scala.html` im Package views

```
@(myForm:Form[models.ZettelModel],jsonMap:Map[String,Object])
@import tags._
@import services._
<div class="multi-field-wrapper" defaultValue="-">
    <ol class="multi-fields" id="creator">

            @for(index <- ZettelHelper.getIndex(myForm,jsonMap,"creator") ){
                        <li class="multi-field">
                        @ldInputField(myForm,{"creator["+index+"]"},services.
↪ZettelFields.creatorZF.getLabel(),"search",3,ArticleHelper.getPersonLookupEndpoints()){
                                                @helpText("creator")
                                                }
                        <br />
                        @singleSelect(myForm,{"academicDegree["+index+"]"},services.
↪ZettelFields.academicDegreeZF.getLabel(),"academicDegree-select",ResearchDataHelper.
↪getAcademicDegreeMap(),5)
```

Hier wurde ein @singleSelect-Formular gewählt, das wiederum in als scala-Template im views-Package liegt. Die Syntax ist ziemlich undurchsichtig und in Grenzen dokumentiert.[3]

4.1.6 Der mutmaßliche Weg der Daten durch to.science.forms

Erste Annahme

Beim Aufruf des Formulars werden im Normalfall die Daten aus einem RDF-XML in das Formular eingelesen. Alternativ kann auch die "Nutzlast" einer POST-Response genutzt werden. Letzteres Verfahren wird jedoch normalerweise nur verwendet, wenn das Formular aus dem Formular selber mit dem Submit-Button abgeschickt wurde und aufgrund von Fehlern die Formularfelder neu eingelesen werden sollen.

[2] In diesem Fall soll eine Select-Box erzeugt werden. Die notwendigen Einträge holen wir über die Klasse `GenericPropertiesLoader.Java` aus einer `AcademicDegree-de.properties`-Datei. Damit können wir Einträge der Select-Box auch zur Laufzeit einfach ändern. Die vorherige Methode, in der entsprechenden Methode eine statische Map anzulegen sollte nicht mehr umgesetzt werden.

[3] https://www.playframework.com/documentation/2.8.x/ScalaForms

Zweite Annahme

Das Versenden der Daten aus dem Formular an die to.science.api erfolgt ausschließlich als API-Call bei dem ggf. verändertes RDF-XML an die to.science.api geliefert wird.

Nachverfolgung der Schritte

Von Drupal zu to.science.api

1. Der Reiter "Bearbeiten" in der Drupal-Oberfläche löst einen GET-Request an die to.science-api mit der Route `:pid/edit` aus.
   Als Parameter werden mitgeliefert:
   a) pid = die ID des zu berabeitenden Objekts,
   b) format = das Format, in dem Metadaten in einem späteren Prozess abgeholt werden sollen
   c) topicId = unklar
2. Der Call wird innerhalb der to.science.api entgegen genommen von der Klasse und Methode:
   a) `controllers.Resource.edit(pid:String,format :  String?="json",topicId?=null)`

Von to.science.api zu to.science.forms

1. | Die Methode `Resource.edit` ruft zunächst die Methode `call` der Klasse `ModifyAction` auf. In den Methodenaufruf wurde dabei noch ein Code-Block mit einer If-Clause` integriert, was das Lesen des Codes nochmal erwschwert.

istdabei

4.2 Core Development

4.2.1 How to contribute

Prerequisites

Software

- git
- maven
- Java SDK (OpenJDK 15)
- Recommended Eclipse

Coding

First of all you need to clone or fork the to.science.core Repository to your local system.

```
$ git clone https://github.com/hbz/to.science.core.git
```

Create a new branch then within your local system where to put your code contributions

```
$ git checkout -b new-branch
```

If you're working on a fix for an issue (Github or any other ticket system in use for the code), please provide the issue number within the branches name

```
$ git checkout -b issue-XXX
```

Generate your wonderful peace of contribution :-)

For testing and compilation please use mvn command with appropriate goals

```
$ mvn clean test compile
```

If you're done with that push our branch into the repository or forked repository either and create a pull request

```
$ git push main new-branch
```

Code will be merged by us into main branch of to.science.core if applicable

Extra

If your code passes the checks and compiles you are able to create your own jar-File of **to.science.core** with the command

```
$ mvn clean compile assembly:single
```

4.2.2 Publish to Maven Central

For publishing the most recent versions of **to.science.core** at Maven Central we use the Sonatype OSSRH (OSS Repository Hosting) as it is a rather lightweight way to publish artifacts at Maven Central.

The complete documentation how to publish artifacts at sonatype can be found here

For Uploading and publishing Source Code at Maven Central you need to create a Sonatype Account via the Sonatype Jira.

With that account you can either create our own repositories at OSSRH or ask any repository owner for granting access to her/his repositories

After the access is granted for publishing at **to.science.core** you should be able to release and publishing a new **to.science.core** release at Maven Central. Do that by the following steps:

1. Update the version number of the to.science.core artifact within the the to.science.core pom.xml

Although it's possible, please **do not** provide a SNAPSHOT-Version

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
↪XMLSchema-instance"   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://
↪maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>io.github.hbz</groupId>
    <artifactId>to.science.core</artifactId>
    <version>1.3.4</version>
    <packaging>jar</packaging>

    <name>Toolbox Open Science Core</name>
```

2. Deploy your release to the OSSRH by executing maven with some special goals

```
$ mvn clean source:jar javadoc:javadoc javadoc:jar verify gpg:sign deploy
```

If succesfull you should see somethimg like that:

```
[INFO] Installing /home/aquast/git/to.science.core/target/to.science.core-1.3.4-javadoc.
↪jar.asc to /home/aquast/.m2/repository/io/github/hbz/to.science.core/1.3.4/to.science.
↪core-1.3.4-javadoc.jar.asc
[INFO]
[INFO] --- maven-deploy-plugin:2.7:deploy (default-deploy) @ to.science.core ---
Uploading to ossrh: https://s01.oss.sonatype.org/service/local/staging/deploy/maven2/io/
↪github/hbz/to.science.core/1.3.4/to.science.core-1.3.4.jar
Uploaded to ossrh: https://s01.oss.sonatype.org/service/local/staging/deploy/maven2/io/
↪github/hbz/to.science.core/1.3.4/to.science.core-1.3.4.jar (109 kB at 15 kB/s)
Uploading to ossrh: https://s01.oss.sonatype.org/service/local/staging/deploy/maven2/io/
↪github/hbz/to.science.core/1.3.4/to.science.core-1.3.4.pom
Uploaded to ossrh: https://s01.oss.sonatype.org/service/local/staging/deploy/maven2/io/
↪github/hbz/to.science.core/1.3.4/to.science.core-1.3.4.pom (5.8 kB at 7.1 kB/s)
Downloading from ossrh: https://s01.oss.sonatype.org/service/local/staging/deploy/maven2/
↪io/github/hbz/to.science.core/maven-metadata.xml
Uploading to ossrh: https://s01.oss.sonatype.org/service/local/staging/deploy/maven2/io/
↪github/hbz/to.science.core/maven-metadata.xml
Uploaded to ossrh: https://s01.oss.sonatype.org/service/local/staging/deploy/maven2/io/
↪github/hbz/to.science.core/maven-metadata.xml (308 B at 338 B/s)
Uploading to ossrh: https://s01.oss.sonatype.org/service/local/staging/deploy/maven2/io/
↪github/hbz/to.science.core/1.3.4/to.science.core-1.3.4-sources.jar
Uploaded to ossrh: https://s01.oss.sonatype.org/service/local/staging/deploy/maven2/io/
↪github/hbz/to.science.core/1.3.4/to.science.core-1.3.4-sources.jar (100 kB at 89 kB/s)
Uploading to ossrh: https://s01.oss.sonatype.org/service/local/staging/deploy/maven2/io/
↪github/hbz/to.science.core/1.3.4/to.science.core-1.3.4-javadoc.jar
Uploaded to ossrh: https://s01.oss.sonatype.org/service/local/staging/deploy/maven2/io/
↪github/hbz/to.science.core/1.3.4/to.science.core-1.3.4-javadoc.jar (547 kB at 297 kB/s)
Uploading to ossrh: https://s01.oss.sonatype.org/service/local/staging/deploy/maven2/io/
↪github/hbz/to.science.core/1.3.4/to.science.core-1.3.4.jar.asc
Uploaded to ossrh: https://s01.oss.sonatype.org/service/local/staging/deploy/maven2/io/
```

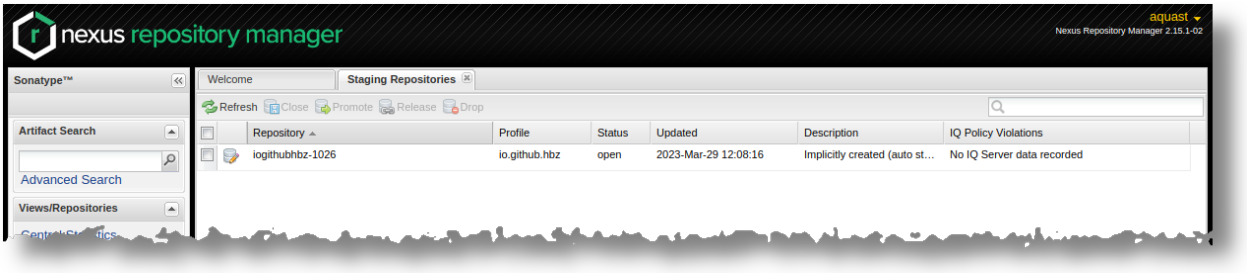
(continues on next page)

(continued from previous page)

```
↪github/hbz/to.science.core/1.3.4/to.science.core-1.3.4.jar.asc (488 B at 1.6 kB/s)
Uploading to ossrh: https://s01.oss.sonatype.org/service/local/staging/deploy/maven2/io/
↪github/hbz/to.science.core/1.3.4/to.science.core-1.3.4.pom.asc
Uploaded to ossrh: https://s01.oss.sonatype.org/service/local/staging/deploy/maven2/io/
↪github/hbz/to.science.core/1.3.4/to.science.core-1.3.4.pom.asc (488 B at 1.5 kB/s)
Uploading to ossrh: https://s01.oss.sonatype.org/service/local/staging/deploy/maven2/io/
↪github/hbz/to.science.core/1.3.4/to.science.core-1.3.4-sources.jar.asc
Uploaded to ossrh: https://s01.oss.sonatype.org/service/local/staging/deploy/maven2/io/
↪github/hbz/to.science.core/1.3.4/to.science.core-1.3.4-sources.jar.asc (488 B at 1.7␣
↪kB/s)
Uploading to ossrh: https://s01.oss.sonatype.org/service/local/staging/deploy/maven2/io/
↪github/hbz/to.science.core/1.3.4/to.science.core-1.3.4-javadoc.jar.asc
Uploaded to ossrh: https://s01.oss.sonatype.org/service/local/staging/deploy/maven2/io/
↪github/hbz/to.science.core/1.3.4/to.science.core-1.3.4-javadoc.jar.asc (488 B at 1.3␣
↪kB/s)
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  25.454 s
[INFO] Finished at: 2023-03-29T12:08:23+02:00
[INFO] ------------------------------------------------------------------------
```

3. To proceed log-in to the NEXUS Webinterfaces staging Repositories section

Within a Browser go to the URL https://s01.oss.sonatype.org/#stagingRepositories . You will be asked to log-in and directed to the stagingRepository-section. There you now should be able to find your uploaded artifact as a "repository". Unfortunately the naming of the repository is not "that" straight forward if you are a human being.

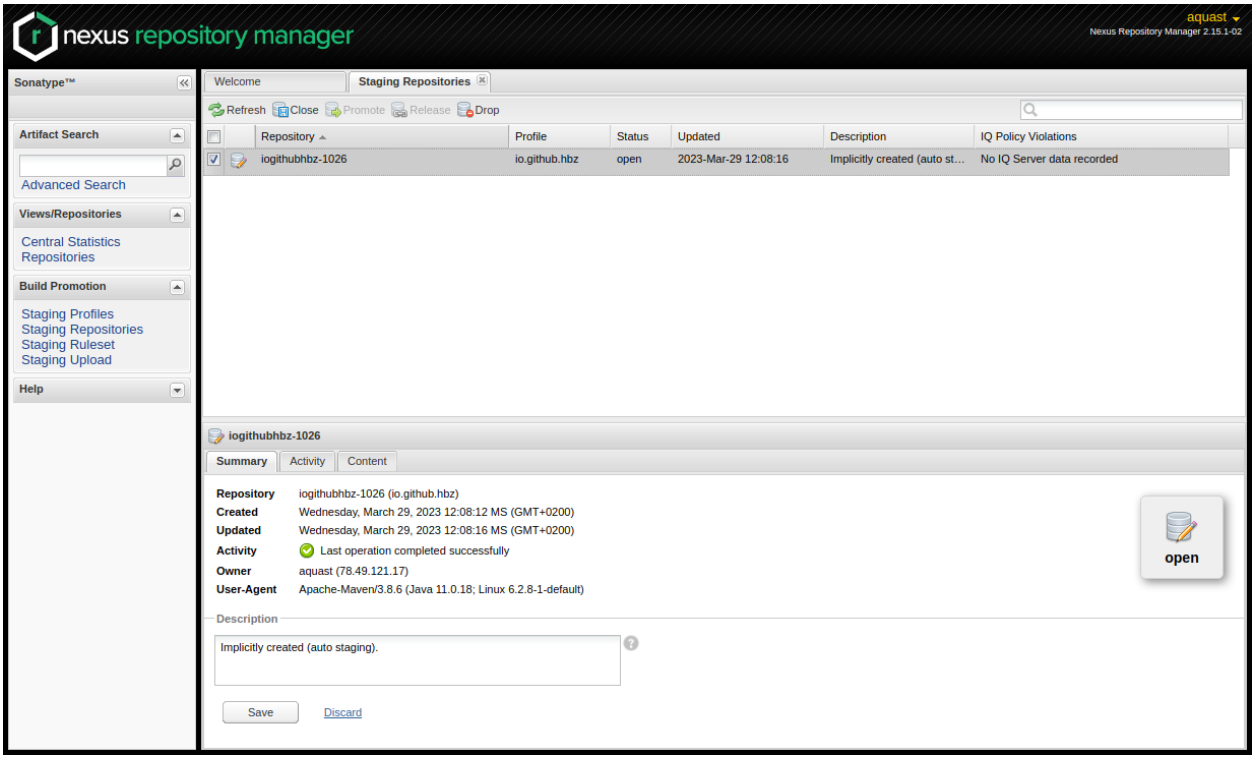


4. Mark youre artifact and proceed with the button "close"

After marking the artifact the buttons 'Close' and 'Drop' will become available for you. You can either proceed with publishing by clicking the 'Close' button or abort the workflow and clear the staging queue by clicking 'Drop'.

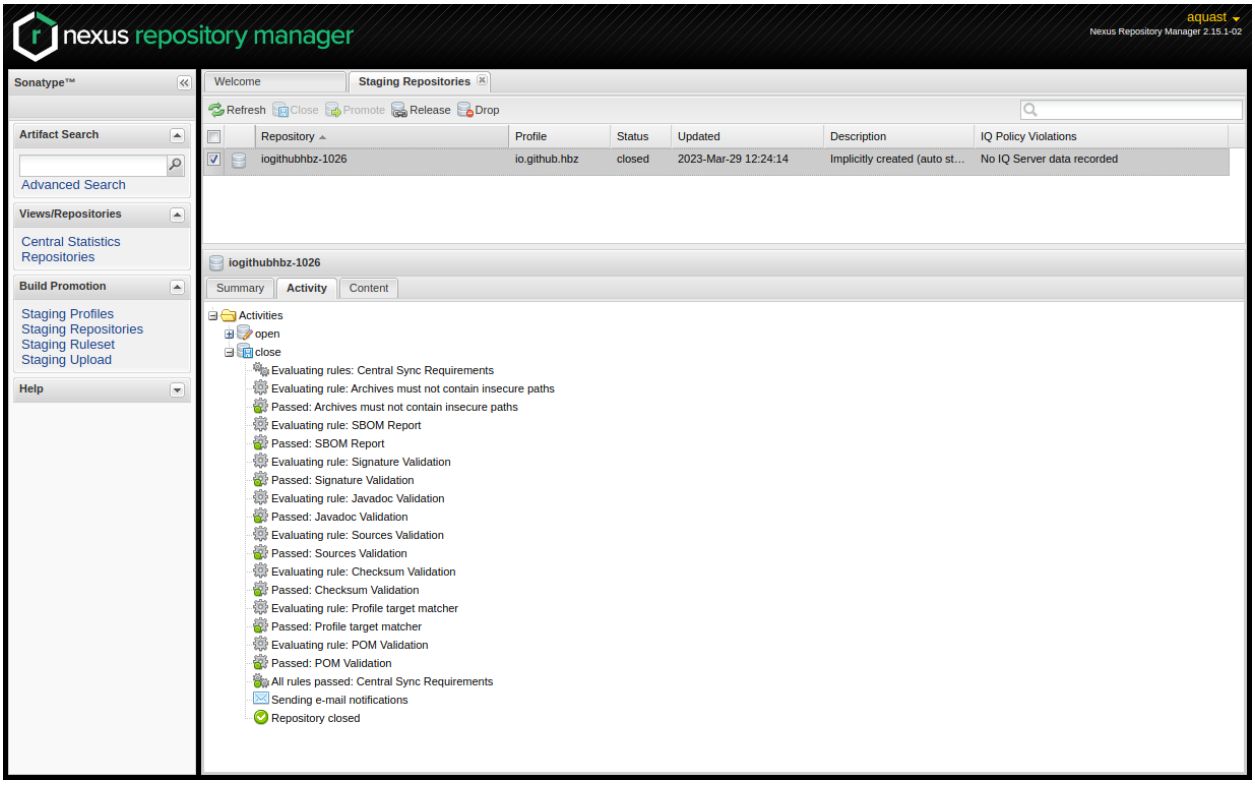
If you proceed with 'Close' the Nexus Repository Manager will check your artifact if it's ready for releasing (and publishing) to Maven Central.

By clicking Refresh you can follow the detailed steps to be processed.



5. Last step: Release your artifact to Maven Central

If all closing steps are done successfully, the ‘Release’ button will become available for you. By clicking ‘Release’ you will release and publish your artifact to Maven Central. Additionally the artifact will be removed from the stagingRepositories section.



It can take some time (in minuites or hours) until you can see your artifact at the Maven Central Searches. Nevertheless the artifact should be already accessible for beeing imported via pom.xml or build.sbt

CHAPTER

FIVE

COLOPHON

Diese Dokumentation ist mit sphinx erstellt. Die Schritte, um an der Doku zu arbeiten sind folgenden

5.1 Dieses Repo herunterladen

```
$ git clone https://github.com/hbz/to.science
```

5.2 Sphinx installieren

Für die Verwendung von Sphinx wird eine virtuelle Pythonumgebung im Verzeichnis `venv` eingerichtet. Das Verzeichnis sollte nicht mit ins git repo committet werden. Das virtuelle Python wird aktiviert und mit pip sphinx und zwei weitere themes installiert.

```
$ cd to.science/docs
$ python3 -m venv ./venv
$ . venv/bin/activate
$ pip install -U sphinx
$ pip install -U sphinx_rtd_theme
$ pip install -U furo
```

5.3 Doku modifizieren und in HTML übersetzen

Die Doku ist in reStructuredText geschrieben wird mittels `make` in html übersetzt.

```
$ cd to.science/docs
$ vi source/colophon.rst
$ make html
```

Das fertige html findet man im Unterverzeichnis `build/html`. Man kann eine einfachen Webserver starten und das Ergebnis unter http://localhost:8000 ansehen.

```
$ python3 -m http.server --directory build/html
```

CHAPTER

SIX

LICENSE

CHAPTER

SEVEN

LINKS

7.1 Slides

- Lobid - http://hbz.github.io/slides/
- Skos-Lookup - http://hbz.github.io/slides/siit-170511/#/
- Regal - http://hbz.github.io/slides/danrw-20180905/#/

7.2 Internes Wiki

- https://wiki1.hbz-nrw.de/display/edd/Dokumentation

7.3 Github

- https://github.com/hbz

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

8.1 Andere Formate

- PDF
- EPUB