

VISIT - TECHNISCHE DOKUMENTATION

Tobias Baumgärtner¹ & Emanuel Berndl² &
Robert Kathrein³ & Kris Raich³ &
Florian Schlenker⁴

19. August 2019



INHALTSVERZEICHNIS

1	Einleitung	6
2	Installationsprozess	7
2.1	Infrastruktur	7
2.2	Projekt auf dem LAS installieren	7
3	TYPO3	10
3.1	Allgemein	10
3.2	Login	10
3.3	Aufbau von TYPO3	11
3.4	Konfiguration des Backends mit TYPO3	12
3.5	Anpassung	13
3.6	Hinzufügen einer Applikation aus dem App-Bundle	14
3.7	Erzeugung des Layouts	15
3.8	Das Karten-Plug-In	16
3.9	Erstellung eines Templates	17
4	Karten-Applikation	17
4.1	Einpflegen der Daten in die Karten-Applikation	17
4.2	Erstellung der Startseite für die Karten-Applikation	18
4.3	Neues Kartenelement hinzufügen	19
4.4	Bearbeitung und Löschen von angelegten Kartenelementen	20
5	Glossar-Applikation	22
5.1	Einpflegen der Daten in die Glossar-Applikation	23
5.2	Erstellung der Startseite für die Glossar-Applikation	23
5.3	Hinzufügen einer neuen Zelle	24
5.4	Hinzufügen eines Events	24
5.5	Neuen Insassen anlegen	24
5.6	Bearbeitung und Löschung von Insassen	28
6	Galerie-Applikation	28
6.1	Einpflegen der Daten in die Glossar-Applikation	28
6.2	Erstellung der Startseite für die Glossar-Applikation	28
6.3	Neuen Insassen anlegen	28
7	Dateiverwaltung	28
7.1	Zugangsdaten zum Dateimanagement	28
7.2	ViSIT-Partner-Liste	29
7.3	Upload von 3D-Objekten und Bildern, Videos und anderen Dateien	29
7.4	Hochladen von Dateien	30
7.5	Veröffentlichung einer Datei im ViSIT-Netzwerk	32
8	Tobi - App Framework	33
9	Kompression	34
9.1	Motivation	34
9.2	Grundlagen	34
9.3	Voraussetzungen	38
9.4	Funktionsweise	39
9.5	Installation und Steuerung	42
9.6	Konfiguration	44
9.7	Zugriff über die Web-Oberfläche	49
9.8	Zugriff über die API	52
10	ViSIT Metadaten und die Semantische Datenbank	57
10.1	Theoretische Grundlagen für die Semantische Datenbank	57
10.2	Technische Details zur Semantischen Datenbank	62
10.3	WissKI - Wissenschaftliche KommunikationsInfrastruktur	65
10.4	Technischer Zugang zu den Metadaten - die ViSIT REST API	69
10.5	Wichtige Technische Charakteristika der Entwicklung und den Betrieb der Semantischen Datenbank	72

10.6 Semantische Datenbank - FAQ und häufig auftretende Probleme	73
11 Schluss	77
12 Appendix	78

ABBILDUNGSVERZEICHNIS

Abbildung 1	Das Login-Fenster für TYPO3 im Browser	10
Abbildung 2	Das Login-Fenster für TYPO3	11
Abbildung 3	Aufbau des TYPO3-Backends	11
Abbildung 4	Installation der Extensions	13
Abbildung 5	Änderung der Sprache	13
Abbildung 6	Konfiguration der Benutzereinstellungen	14
Abbildung 7	Änderung der Benutzereinstellungen und der Sprache	14
Abbildung 8	Änderung des Passworts	15
Abbildung 9	Hinzufügen einer neuen Seite	15
Abbildung 10	Erzeugung des Layouts der neu erstellten Seite	16
Abbildung 11	Auswahl der Plug-Ins	16
Abbildung 12	Einbindung eines Plug-Ins	17
Abbildung 13	Leere Kartenübersicht	18
Abbildung 14	Erstellung der Startseite für die Karten-Applikation	18
Abbildung 15	Text für die Startseite der Applikationen, zu finden auf https://github.com/ViSIT-Dev/appbundle	19
Abbildung 16	Erstellung der Startseite für die Karten-Applikation	20
Abbildung 17	Ein neues Kartenelement hinzufügen	20
Abbildung 18	Listenansicht über alle angelegten Kartenelemente	21
Abbildung 19	Startseite der Karten-Applikation	21
Abbildung 20	Ansicht der Karte im Browser	21
Abbildung 21	Kartenelement - Point of Interest - mit Detailinformation	22
Abbildung 22	Ansicht der Glossar-Applikation auf einem Tablet	22
Abbildung 23	Übersicht über alle angelegten Insassen	23
Abbildung 24	Konfiguration der Glossar-Applikation	23
Abbildung 25	Befüllung der Startseite der Glossar-Applikation	24
Abbildung 26	Startseite der Glossar-Applikation	24
Abbildung 27	Hinzufügen einer neuen Zelle	25
Abbildung 28	Erstellen einer neuen Zelle	25
Abbildung 29	Angelegte Zellen in der Listenübersicht	25
Abbildung 30	Hinzufügen eines neuen Events	26
Abbildung 31	Angelegte Events in der Listenansicht	26
Abbildung 32	Anlegen eines neuen Insassens	27
Abbildung 33	Anlegen eines Insassens	27
Abbildung 34	Angelegte Insassen in der Listenansicht	27
Abbildung 35	Ansicht der Insassen im Browser	28
Abbildung 36	Einstellung der ViSIT App Extension	28
Abbildung 37	ViSIT App Extensions	29
Abbildung 38	Ansicht der ViSIT-Partner mit Zugang zur Mediendatenbank im Peer-to-Peer-Netzwerk	30
Abbildung 39	Ansicht der bereits verfügbaren Dateien in der Dateiliste	30
Abbildung 40	Ansicht des ausgefüllten Formulars für den Dateiupload	31
Abbildung 41	Bestätigungenachrichten nach einem erfolgreichen Upload	31
Abbildung 42	Hochgeladene Datei in der Listenübersicht	32
Abbildung 43	Grundlegende Elemente der Geometrie eines 3D-Modells	35
Abbildung 44	Beispiel einer Edge-Collapse-Operation. Die Zielposition der beiden an das Edge angrenzenden Vertices ist <i>rot</i> markiert	37

Abbildung 45	Beziehung von mediaTripleID und mediaTripleURL anhand eines Beispiels	40
Abbildung 46	Startseite, unter anderem mit einem Überblick über die laufenden und anstehenden Kompressions-Aufträge	50
Abbildung 47	Ansicht zum Absetzen eines neuen Kompressions-Auftrags .	51
Abbildung 48	Archiv-Ansicht mit einem Überblick über die ausgeführten Kompressions-Aufträge	51
Abbildung 49	Einstellungsmöglichkeiten über die Web-Oberfläche	53
Abbildung 50	Informationen aus obigen Aussagen, kombiniert als Graph. .	58
Abbildung 51	Grundlegende eigene Wissensbasis (oben), erweitert um zwei externe Wissensbasen (unten).	59
Abbildung 52	Arbeitsprozess hinter der Entwicklung des ViSIT Modells. . .	62
Abbildung 53	Technische Infrastruktur der Semantischen Datenbank des ViSIT Projekts.	63
Abbildung 54	Zwei Beispiel-Pfade aus der WissKI Konfiguration des ViSIT Projekts.	67
Abbildung 55	Erste Maske zum Editieren eines WissKI Pfads.	68
Abbildung 56	Zweite Maske zum Editieren eines WissKI Pfads.	69
Abbildung 57	Beispiel-Pfad aus der WissKI Konfiguration des ViSIT Projekts für eine Relation zwischen zwei Entitäten der Datenbank. .	69
Abbildung 58	Modell der digitalen Repräsentationen.	70
Abbildung 59	WissKI Pathbuilder Ansicht, die den Download der Pfad-Datei anbietet.	75
Abbildung 60	pom.xml Konfiguration zum produktiven Deployment der REST API.	75
Abbildung 61	Einstellungen zum Verbinden des Triplestores. Hier gezeigt: lokale Konfiguration, während die Konfiguration für Produktiv- und Testsystem oben vorhanden aber auskommentiert ist. . .	76
Abbildung 62	Übersicht der Konfiguration eines WissKI Salz Adapters, Teil 1. .	90
Abbildung 63	Übersicht der Konfiguration eines WissKI Salz Adapters, Teil 2. .	91
Abbildung 64	Detailansicht einer definierten Ontology im WissKI System am Beispiel VisMo für das ViSIT Projekt.	92
Abbildung 65	Übersicht der ersten Pfade des für das ViSIT Projekt definierten Pathbuilders.	93
Abbildung 66	Ausgangs-Interface zum Erzeugen einer Hauptentität im WissKI System.	94
Abbildung 67	Eingabe-Interface für ein Ausstellungsobjekt im ViSIT WissKI System.	95
Abbildung 68	Beispiel Ausgabe-Interface einer Partisane des ViSIT WissKI Systems.	96

TABELLENVERZEICHNIS

Tabelle 1	Für ein 3D-Modell notwendige und optionale Dateien	39
Tabelle 2	Überblick über alle Konfigurationsmöglichkeiten der Kompressionskomponente	45
Tabelle 3	Beispiele für die sich bei unterschiedlichen Vertexanzahlen ergebenden Texturauflösungen bei der Standardkonfiguration.	47
Tabelle 4	Pro Kompressionsstufe notwendige Parameter bei der Bildkompression	48
Tabelle 5	API Funktionen für die Digital Representations.	71

Tabelle 6 API Funktionen zum Auslesen von Objekten. 71

¹ TODO Universität Passau

² Lehrstuhl für verteilte Informationssysteme und Data Science, Universität Passau

- Lehrstuhl für verteilte
3 TODO, FH Kufstein

3 *TODU, FH Kufstein*

1 EINLEITUNG

blub hi!

Things to maybe mention here:

- ViSIT Projekt Bitbucket/github, welche wir zur Entwicklung nutzen

2 INSTALLATIONSPROZESS

2.1 Infrastruktur

Die ViSIT-Applikationen basieren auf der Server-Client-Architektur. Damit diese Applikationen installiert werden können, wird ein hausinternes Netzwerk (Intranet) und ein damit verbundener Server - lokaler Applikations-Server (kurz LAS) - benötigt. Die ViSIT-Applikationen sind in diesem Zusammenhang die Clients, welche über das Netzwerk mit dem lokalen Applikations-Server verbunden sind. Auf dem LAS ist das ViSIT-System installiert, welches über das Internet Zugang zum globalen ViSIT-Netzwerk hat. Das ViSIT-System ist eine Ansammlung von mehreren kleinen Applikationen, welche parallel auf dem LAS laufen können. Jeder Client, auf welchem eine der ViSIT-Applikationen läuft, hat eigene Server-Software, welche auf dem LAS installiert ist und für die serverseitigen Berechnungen zuständig ist.

Die Applikationen wurden mit der IT-Technologie "Docker" erstellt. Mit Docker hat man die Möglichkeit, Anwendungen in sogenannten Containern auszuführen und diese Container können aufeinander aufbauen und miteinander kommunizieren. Im Gegensatz zu einer virtuellen Maschine, ist eine Docker-basierte Anwendung nur ein Prozess, der auf dem System ausgeführt wird. Es ist somit kein Gastbetriebssystem erforderlich, wie dies bei Virtuellen Maschinen der Fall ist. Container sind einfach konfigurierbare, abgeschlossene Einheiten, in welchen die Anwendung ausgeführt werden. Mit Docker können Linux-Container erstellt und verwendet werden können. Die erstellten Container sind eine Virtualisierung auf der Ebene des Betriebssystems. Durch das Erstellen von Containern, werden isolierte Linux-Systeme auf dem gleichen Host erzeugt. Diese Container können flexibel erstellt, bereitgestellt, kopiert und zwischen Umgebungen verschoben werden. Zweck dieser Container ist die Unabhängigkeit und die Fähigkeit, mehrere Prozesse und Applikationen getrennt voneinander betreiben zu können. Die Vorteile von Docker-Containern sind unter anderem Modularität und Versionsverwaltung. Modularität ermöglicht es, bei zum Beispiel einer Reparatur oder Aktualisierung einer Applikation, nur einen Teil dieser Applikation außer Betrieb zu nehmen, ohne die gesamte Applikation außer Betrieb nehmen zu müssen. Docker bietet eine eingebaute Versionsverwaltung, welche es erlaubt, den aktuellen Stand eines Containers in ein sogenanntes Image zu sichern. Somit ist es möglich, die unterschiedlichen Zustände eines Images in einer Historie nachzuverfolgen. Ein Image ist ein Speicherabbild eines Containers und es besteht aus mehreren Layern, welche schreibgeschützt sind und somit nicht verändert werden können. Ein Layer ist wiederum ein Teil eines Images und enthält einen Befehl oder eine Datei, welche dem Image hinzugefügt wurde. Aufgrund dieser Layer kann die ganze Historie eines Images nachvollzogen werden.

2.2 Projekt auf dem LAS installieren

Als erster Schritt muss die Datenbank für die Applikation angelegt werden. Wie oben erklärt, wurde für das ViSIT-Projekt Docker verwendet. Damit gespeicherte Daten auch außerhalb eines Containers abgelegt oder in einem anderen Container eingebunden werden können, werden sogenannte Volumes erstellt. Volumes haben viele Vorteile, vor allem aber sind sie einfacher zu sichern oder zu migrieren. Volumes funktionieren sowohl auf Linux- als auch auf Windows-Containern. Im ersten Schritt wird ein Volume mit der Datenbank auf dem lokalen Rechner im Terminal mit dem Kommando

```
| docker volume create visit-database
```

Listing 1: Example XML

erstellt. Einen eigenen Volume benötigt man deshalb, weil die dort abgelegten Daten permanent gespeichert werden müssen - würde z.B.: der Container gelöscht oder beendet werden - dann wären die nur im Docker Container gespeicherten Daten ebenfalls gelöscht worden. Damit dies nicht passieren kann, werden die Daten parallel lokal auf dem Rechner gespeichert. Damit Dateien zwischen Geräten in einem lokalen Netzwerk oder zwischen entfernten Geräten über das Internet synchronisiert werden können, wird eine Datensynchronisation mit Peer-to-Peer-Übertragung benötigt. Dies wird im ViSIT-Projekt mit Syncthing realisiert und auch dafür muss ein eigener Volume lokal auf dem Rechner erstellt werden. Dies geschieht mit

```
1 | docker volume create visit-syncthing
```

Listing 2: Example XML

-Befehl, welcher ebenfalls im Terminal ausgeführt wird. Als nächster Schritt wird das gesamte ViSIT-Projekt von GitHub mittels

```
1 | docker run -d --name visit -p 80:80 -p 22000:22000 -p 21027:21027
2 | -v visit-syncthing:/var/syncthing
3 | -v s:/p2p/visit:/var/p2p
4 | -v visit-database:/var/lib/mysql
5 | --restart unless-stopped visitapp/maincontainer
```

Listing 3: Example XML

geklont. Beim erstmaligen Starten benötigt der Vorgang länger, da das Projekt aus dem Git Repository sowie das Appbundle (<https://github.com/ViSIT-Dev/appbundle>) heruntergeladen werden.

Erklärung der einzelnen Befehle:

```
1 | docker run -d --name visit -p 80:80 -p 22000:22000 -p 21027:21027
```

Listing 4: Example XML

```
1 | docker run
```

Listing 5: Example XML

startet den Container und mit den mit den Parametern

```
1 | -d
```

Listing 6: Example XML

gibt man an, dass der Container im Hintergrund dauerhaft laufen soll (Daemonmodus). Weiters wird mit

```
1 | --name visit
```

Listing 7: Example XML

der Name des Containers festgelegt, in diesem Fall heißt der Container visit. Der Container kann im weiteren Verlauf auch über diesen Namen angesprochen werden. Mit dem Parameter

```
1 | -p 80:80
```

Listing 8: Example XML

werden die Ports vom Host an den Container gebunden. Hier wird der lokale Hostport 80 auf den Containerport 80 gemappt. Die weiteren Ports

```
1 | -p 22000:22000 -p 21027:21027
```

Listing 9: Example XML

werden für das Syncthing und für das Peer to Peer-Netzwerk benötigt. Als nächstes folgt der Befehl

```
1 | -v visit-syncthing:/var/syncthing
```

Listing 10: Example XML

Mit dem Parameter

```
1 | -v
```

Listing 11: Example XML

wird ein Verzeichnis (Volume) auf dem Hostrechner zu einem Verzeichnis innerhalb des Containers verbunden, auf diese Weise werden die Daten persistent gespeichert, das heißt, dass ein Ordner auf dem Hostsystem auf einen Ordner im Container gemappt wird. Das bedeutet, dass die Daten in beiden Ordnern immer inhaltsgleich sind. Ohne dem Mapping zu einen Ordner auf dem Hostsystem, wären alle Daten aus dem Docker Container, wenn dieser Container gelöscht wird, ebenfalls gelöscht. Um die Daten persistent, also dauerhaft zu speichern, wird immer ein Ordner im Hostsystem mit dem entsprechenden Ordner im Docker Container gemappt. Zuerst wird das Verzeichnis auf dem Hostrechner angegeben, hier

```
1 | visit-syncthing
```

Listing 12: Example XML

und nach dem Doppelpunkt steht das Verzeichnis innerhalb des Containers, hier

```
1 | /var/syncthing
```

Listing 13: Example XML

Im nächsten Teil des Befehls

```
1 | -v s:/p2p/visit:/var/p2p
```

Listing 14: Example XML

wird ebenfalls zuerst das Verzeichnis auf dem Hostrechner angegeben,

```
1 | s:/p2p/visit
```

Listing 15: Example XML

und dann das Verzeichnis innerhalb des Containers

```
1 | /var/p2p
```

Listing 16: Example XML

Im nächsten Befehl

```
1 | -v visit-database:/var/lib/mysql
```

Listing 17: Example XML

geht es um die Verbindung zur Datenbank. Hier wird ebenfalls zuerst das Verzeichnis auf dem Hostrechner angegeben

```
1 | visit-database
```

Listing 18: Example XML

und nach dem Doppelpunkt steht das Verzeichnis innerhalb des Containers

```
1 | /var/lib/mysql
```

Listing 19: Example XML

Zuletzt wird mittels

```
1 | --restart unless-stopped visitapp/maincontainer
```

Listing 20: Example XML

dem System mitgeteilt, dass der Docker Container

```
1 | visitapp/maincontainer
```

Listing 21: Example XML

automatisch gestartet werden soll außer, wenn er manuell oder anderweitig gestoppt wird.

Wenn der Vorgang abgeschlossen ist, kann über Lokalhost im Browser unter **localhost:80/typo3/** das Backend aufgerufen werden (siehe Abbildung 1). Das erste Mal einloggen in das Backend (TYPO3) erfolgt mit dem **Benutzername: admin** und **Passwort: YoGrZOy1og**.

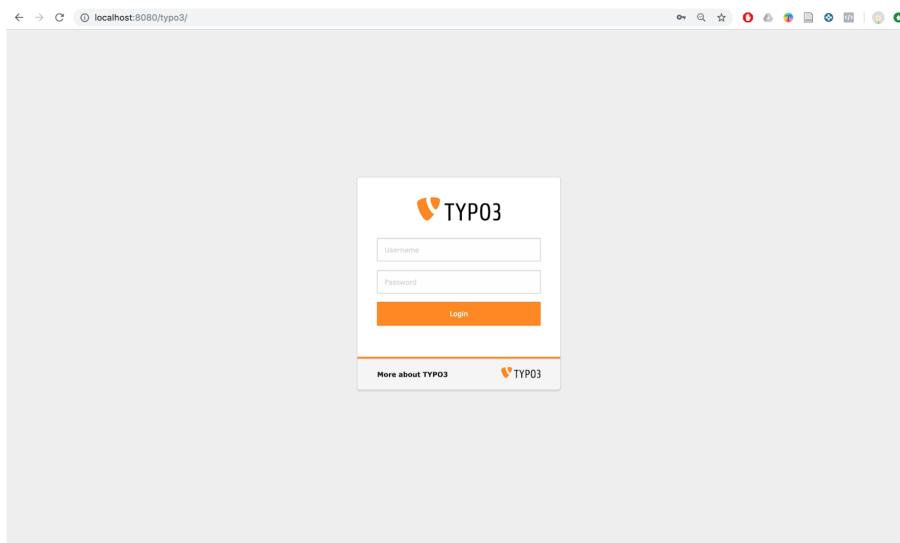


Abbildung 1: Das Login-Fenster für TYPO3 im Browser

3 TYPO3

3.1 Allgemein

TYPO3 ist ein freies Content-Management-System für Webseiten, es wird in Frontend und Backend getrennt. Als Frontend wird die Präsentationsebene bezeichnet, das ist der Teil einer Applikation, den der Betrachter sehen kann. Als Backend hingegen, bezeichnet man die Datenzugriffsebene, das ist der Teil einer Applikation, welcher nicht für den Besucher sichtbar ist. Das Backend ist der Verwaltungsbereich einer Webseite. TYPO3 wird auf einem Webserver installiert und über den Webbrower benutzt.

Das Backend ist die Datenzugriffsebene, dieser Teil ist für den Endbenutzer nicht sichtbar. Es beinhaltet die Programmierung einer Applikation und den Administrationsbereich. Im Gegensatz dazu das Frontend, das ist die tatsächliche Webseite, die der Endbenutzer im Browser sieht, also die Benutzeroberfläche.

3.2 Login

Damit niemand unbefugter im Frontend sowie Backend etwas verändern kann, muss man sich zuerst ins Backend einloggen. Dies geschieht über den Aufruf der Domain **localhost:80/typo3/** im Webbrower (siehe Abbildung 1).



Abbildung 2: Das Login-Fenster für TYPO3

Im Login-Fenster kann der Benutzername sowie das Passwort eingetragen werden (siehe Abbildung 2). Beim ersten Login ist der **Benutzername: admin** und das **Passwort: visit-admin**, dieser muss in weiterer Folge verändert werden. Mehr dazu siehe Anpassung. Nach einem erfolgreichen Login wird das Backend mit den dazugehörigen Modulen im Browser geladen.

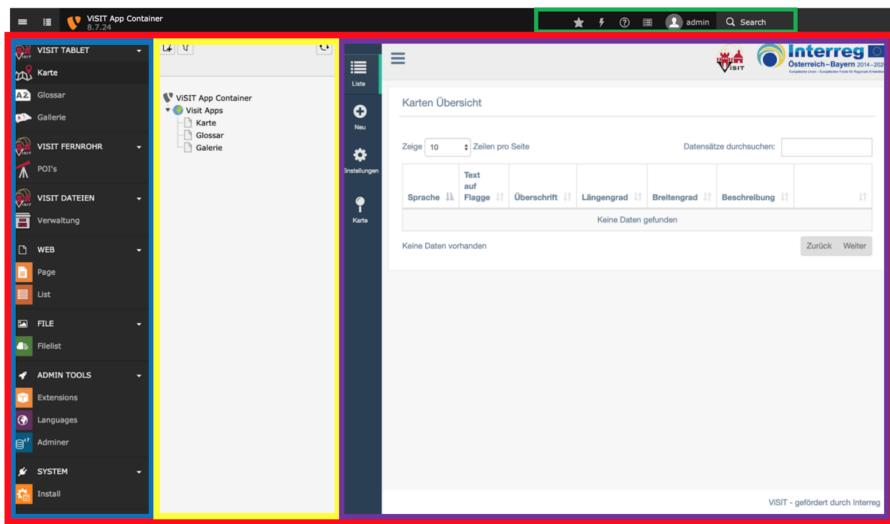


Abbildung 3: Aufbau des TYPO3-Backends

3.3 Aufbau von TYPO3

Das TYPO3-Backend besteht aus einem Kopfbereich (grün eingerahmt) und einem Hauptbereich (rot eingerahmt), welcher aus drei Spalten besteht (siehe Abbildung 3). Im Kopfbereich kann der Administrator seine TYPO3-Benutzererstellungen konfigurieren. Im Hauptbereich werden Webdokumente bearbeitet. Das TYPO3-Backend wird von links nach rechts abgearbeitet.

3.3.1 Kopfleiste

Die Kopfleiste bietet die Möglichkeit, die im TYPO3 Backend gespeicherten Lesezeichen aufzurufen (Stern-Symbol), den TYPO3 Cache der gesamten Webseite zu leeren (Blitz-Symbol) sowie Hilfe und Dokumentationen (Fragezeichen) zu TYPO3 aufzurufen. Das vierte Symbol zeigt die wichtigsten Systeminformationen. Mit einem Klick auf den Benutzernamen, in der Grafik "admin", öffnet sich ein Kontext-Menü mit der Möglichkeit Einstellungen an seinem Benutzer vorzunehmen oder sich aus dem TYPO3 Backend auszuloggen. Rechts neben dem Benutzer befindet sich das Suchfeld, mit dem sich das gesamte TYPO3 Backend durchsuchen lässt.

3.3.2 Die Spalten des Hauptbereichs

Linke Spalte: Modulleiste (blau eingerahmt), hier kann das Modul ausgewählt werden, welches bearbeitet werden soll (siehe Abbildung 3).

Mittlere Spalte: Seitenbaum (gelb eingerahmt), hier wird die zu bearbeitende TYPO3-Seite ausgewählt. Der Seitenbaum ist das zentrale Element, wenn es darum geht sich durch die Webseite zu navigieren. Hier wird der Aufbau und die Seitenhierarchie der Webseite in einer Struktur abgebildet, die der Ordnerstruktur ähnlich ist. Einzelne Seiten können Unterseiten enthalten, die im Seitenbaum eingetragen dargestellt werden (siehe Abbildung 3).

Rechte Spalte: Arbeitsbereich (violett eingerahmt), hier wird am ausgewählten TYPO3-Objekt gearbeitet (siehe Abbildung 3).

3.4 Konfiguration des Backends mit TYPO3

Die für die Applikationen benötigten TYPO3 Extensions werden automatisch installiert, sollte eine weitere Extension benötigt werden, befindet sich eine Anleitung für die Installation in diesem Abschnitt. Extensions sind optionale Software-Komponenten, also Zusatzmodule, die eine bestehende Software erweitern.

Installation von TYPO3 Extensions: Dazu wird in der linken Spalte zuerst das Modul "Extensions" ausgewählt. Dann erscheinen im Hauptfenster verschiedene Extensions, welche alphabetisch gelistet sind. Bei der Erstinstallation werden folgende Extensions (unten ist der Key angegeben, welcher sich in der mittleren Spalte befindet) automatisch installiert (siehe Abbildung 37):

- visit_tablets
- scheduler
- tstemplate
- fluid_styled_content
- setup

Mittels einem Klick auf das Würfelsymbol mit einem Plus werden die oben angegebenen Extensions der Reihe nach aktiviert (siehe Abbildung 37). Die aktivierten Extensions erscheinen dann als auswählbare Module in der linken Spalte. Optional kann im nächsten Schritt die Sprache Deutsch installiert werden, sonst ist die Hauptsprache Englisch. Um die Sprache zu installieren, wird in der linken Spalte unter den ADMIN TOOLS "Languages" ausgewählt (siehe Abbildung 5).

Im Hauptfenster erscheinen nach dem Klick die unterstützten Sprachen, hier "German" suchen und zuerst mittels einem Klick auf das Plus-Symbol links von der Sprache die Sprache aktivieren, dabei erscheint oben rechts eine grüne Meldung mit "Success, language was successfully activated.". Als nächstes muss die aktivierte Sprache mittels Klick auf das Download-Symbol rechts von der Sprache

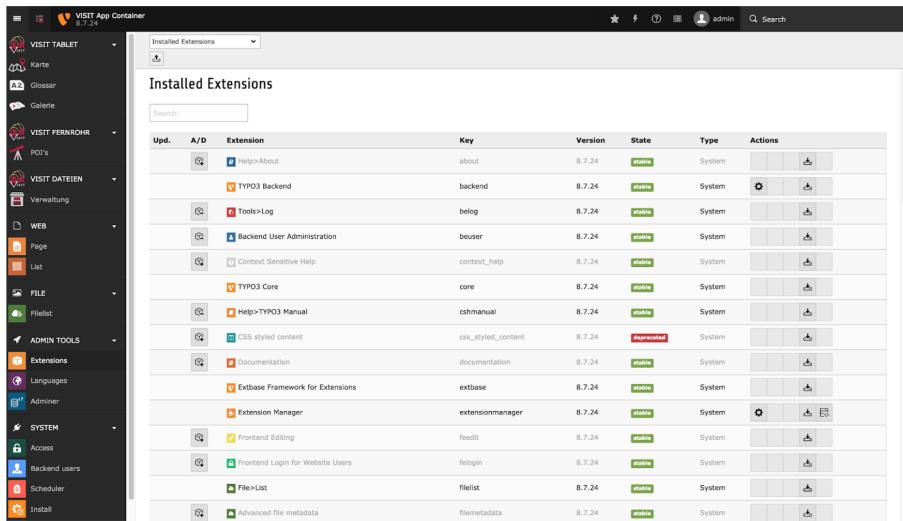


Abbildung 4: Installation der Extensions

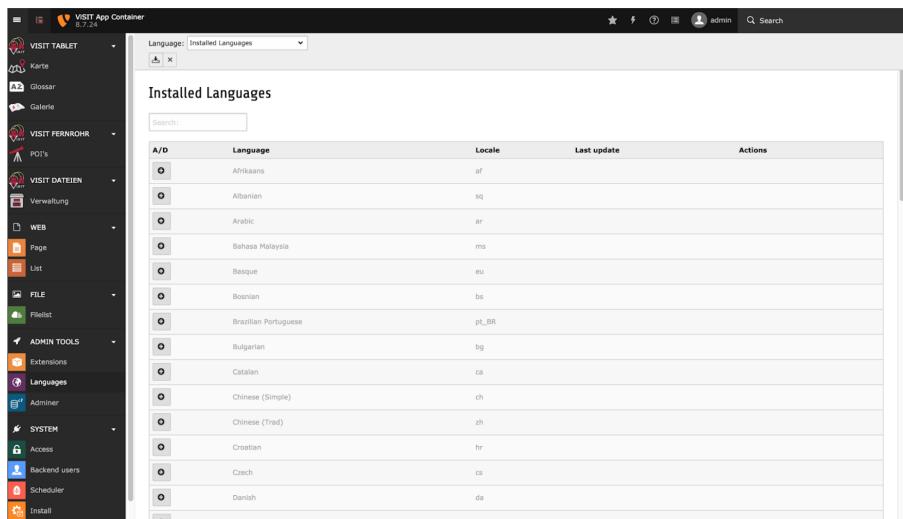


Abbildung 5: Änderung der Sprache

heruntergeladen werden. War der Download erfolgreich, so erscheint oben rechts eine grüne Meldung mit "Success. The translation update has been successfully completed.".

3.5 Anpassung

Im Kopfbereich können die TYPO3-Benutzereinstellungen konfiguriert werden. Dazu wird im Kopfbereich oben rechts zuerst der Benutzer ausgewählt. Bei der Erstinstallation ist es der "admin", dabei wird ein Menü aufgeklappt, aus welchem die "User settings" ausgewählt werden (siehe Abbildung 6).

Jetzt erscheinen im Hauptbereich die User Settings, welche in dieser Maske konfiguriert werden können. Jetzt kann zuerst die Sprache umgestellt werden. Dies kann gleich im ersten Raster "Personal data", im unteren Bereich unter Languages geändert werden (siehe Abbildung 7). Hier kann die heruntergeladene Sprache mittels Dropdown ausgewählt werden. Damit die Auswahl auch gespeichert und angewendet wird, muss auf das Speicher-Symbol (Diskette) ganz oben links im Hauptfenster geklickt werden. Nur durch diesen Klick werden die User Settings upgedated und die Sprache auch angewendet. Jetzt erscheinen oben im Hauptfenster drei Meldungen. Die grüne Meldung besagt, dass die Settings upgedated wurden. Die blaue

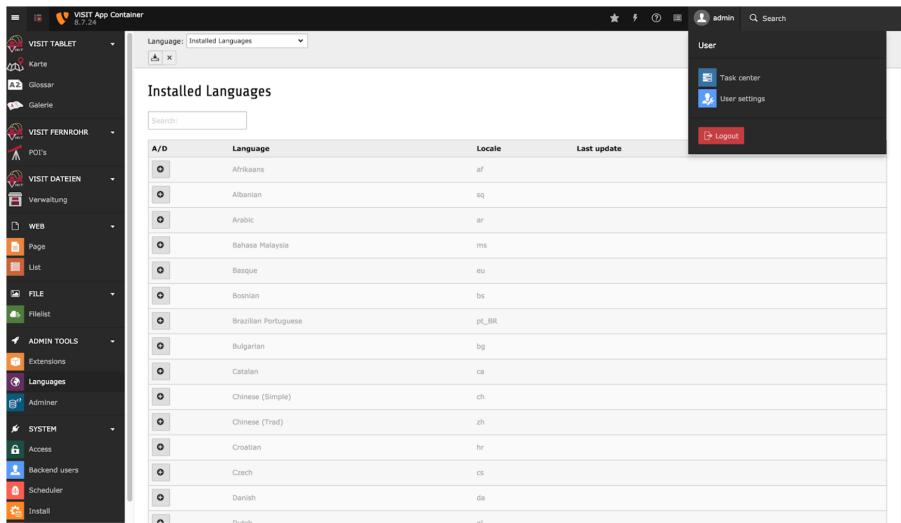


Abbildung 6: Konfiguration der Benutzereinstellungen

Meldung sagt, dass die Seite (`localhost:80/typo3/`) neu geladen werden muss, um die Veränderungen zu aktivieren. Die rote Meldung sagt, dass das neue Passwort nicht upgedated wurde, da es nicht zweimal eingegeben wurde.

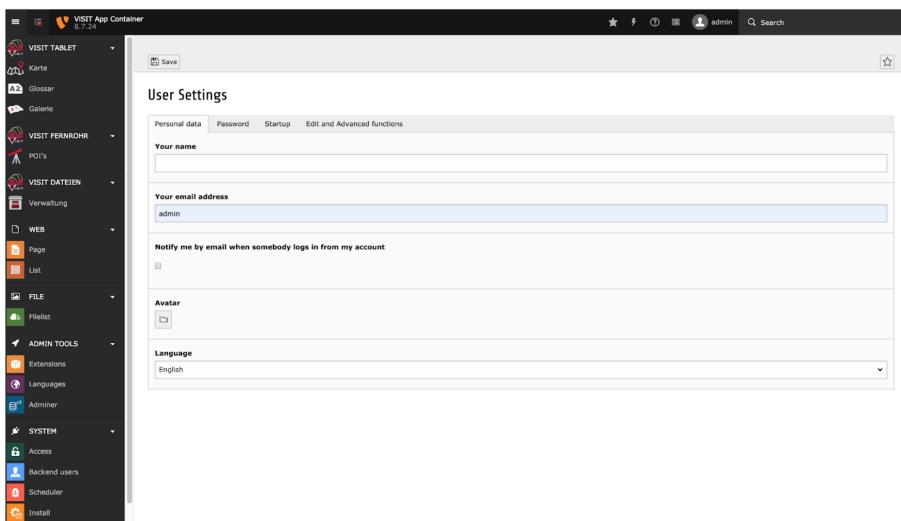


Abbildung 7: Änderung der Benutzereinstellungen und der Sprache

Als nächstes wird das Passwort verändert. Dazu wird die Registerkarte "Password" ausgewählt (siehe Abbildung 8). Jetzt erscheint das zuvor eingegebene Passwort "visit-admin" als eine Punkte-Kette in der ersten Zeile, hier kann das Passwort mit einem neuen Passwort überschrieben werden. Gleiches Passwort wird in der darunter liegenden Zeile nochmals eingegeben. Damit die Änderungen gespeichert werden, wird wieder oben links das Speichern-Symbol geklickt. Ab jetzt werden auch die Änderungen der Sprache angewendet und alles wird auf Deutsch angezeigt. Mit diesem Schritt ist das Backend fertig vorbereitet.

3.6 Hinzufügen einer Applikation aus dem App-Bundle

Dazu wird in der linken Spalte "Seite" ausgewählt. Jetzt kann dem ViSiT App Container eine Seite hinzugefügt werden. Zuerst muss auf das oben ganz links befindlichen Seiten-Symbol geklickt werden, dann erscheint eine Auswahl an möglichen Aktionen. Hier das erste leere Seite-Symbol anklicken und auf den darunter befind-

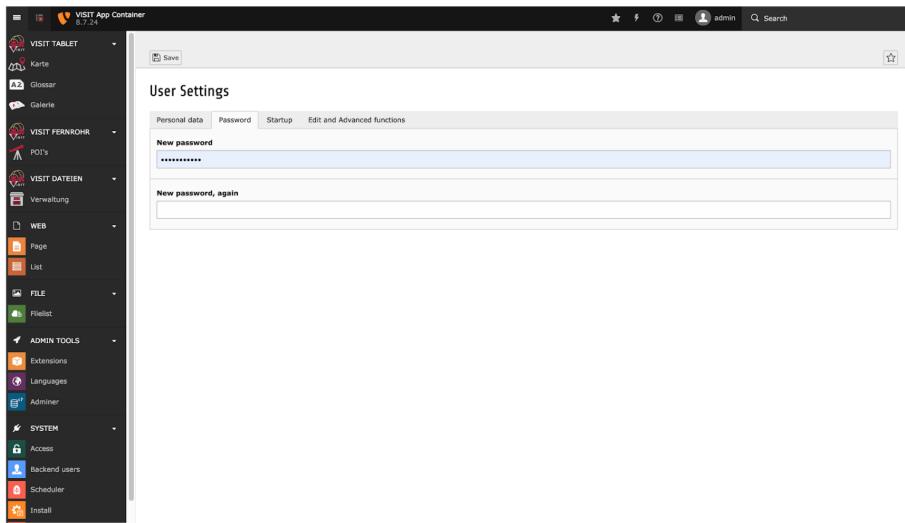


Abbildung 8: Änderung des Passworts

lichen ViSiT App Container ziehen und darüber loslassen, anschließend kann der Seite ein Name gegeben werden (siehe Abbildung 9).

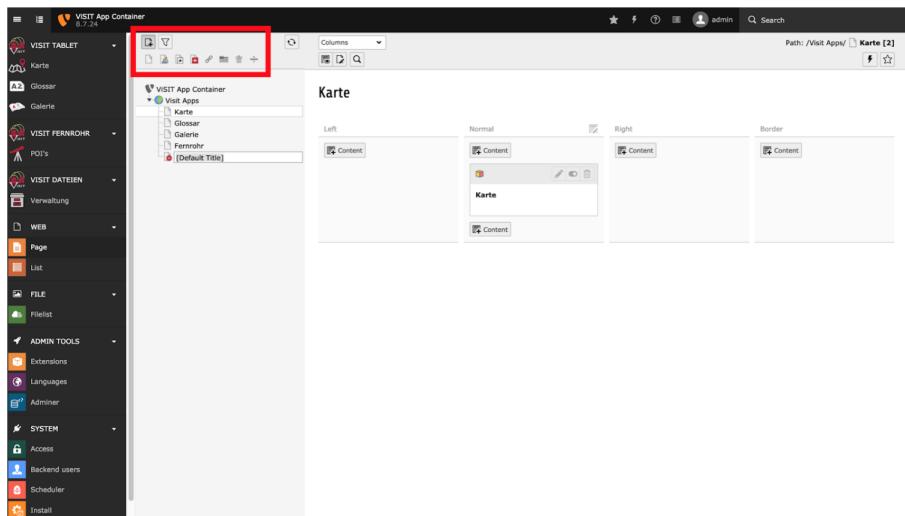


Abbildung 9: Hinzufügen einer neuen Seite

Mittels Rechtsklick auf die soeben erstellte Seite erscheint unter der Seite ein weiteres Menü, aus diesem dann "Bearbeiten" auswählen. Danach kann rechts die Seite konfiguriert werden.

Im nächsten Schritt muss das Verhalten der Seite konfiguriert werden. Dazu den Raster "Verhalten" anklicken und unter "Sonstige" "Als Anfang der Website benutzen" aktivieren. Dann den Raster "Zugriff" auswählen und unter "Sichtbarkeit" "Seite" deaktivieren. Nachdem die Änderungen durchgeführt wurden, müssen diese gespeichert werden. Dazu muss auf das Speicher-Symbol oben auf der Hauptseite geklickt werden. Danach erscheint ein Weltkugel-Symbol neben der soeben erzeugten Seite im linken Teil des Hauptfensters.

3.7 Erzeugung des Layouts

Um das Layout der Seite zu definieren, muss auf die soeben erzeugte Seite geklickt werden (siehe Abbildung 10).

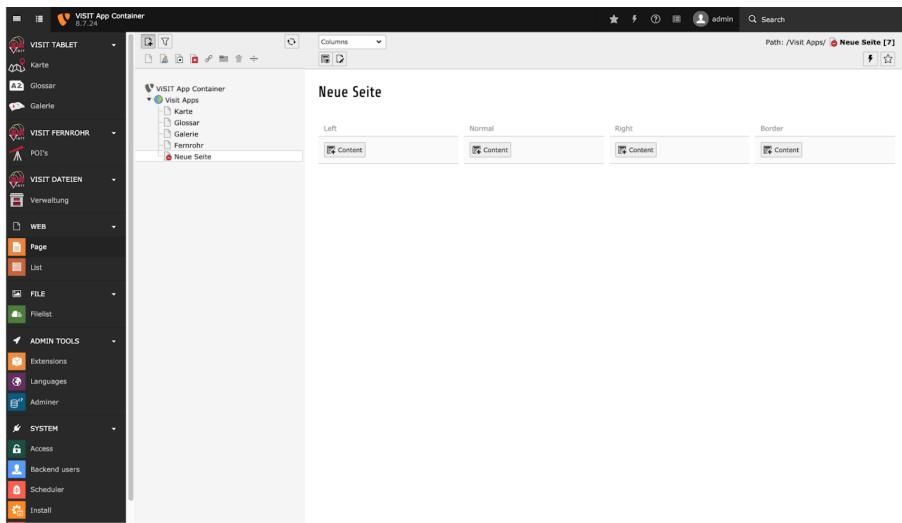


Abbildung 10: Erzeugung des Layouts der neu erstellten Seite

Im rechten Teil des Hauptfensters erscheinen vier Möglichkeiten der Inhaltspositionierung. Für die ViSIT-Applikationen wird die normale Inhaltspositionierung benötigt. Um weitere Konfiguration durchzuführen, unter “Normal” auf das das Inhalts-Symbol klicken und im Raster “Plug-Ins” auswählen, hier können die Plugins für die jeweilige ViSIT-Applikation ausgewählt werden (siehe Abbildung 11).

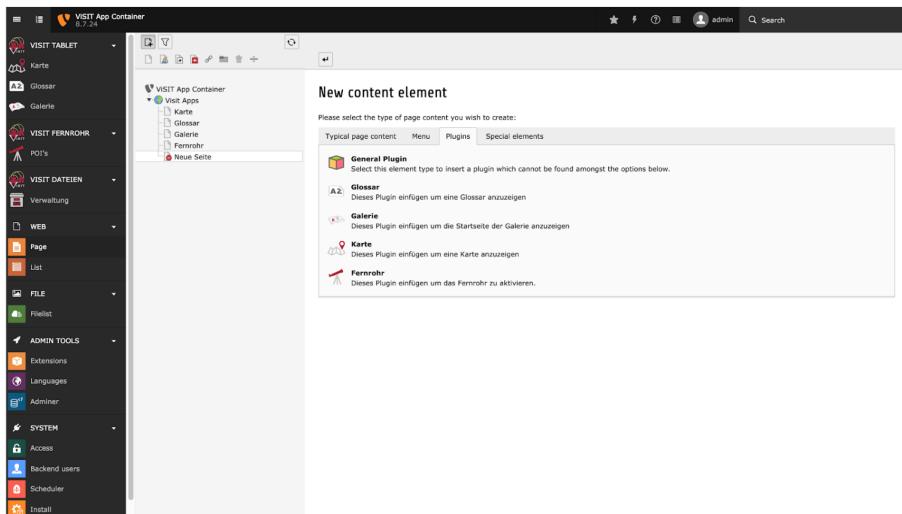


Abbildung 11: Auswahl der Plug-Ins

3.8 Das Karten-Plug-In

Im Raster “Plug-Ins” die “Karte - Dieses Plugin einfügen um eine Karte anzuzeigen” auswählen und oben auf das Speicher-Symbol klicken, damit die Änderungen gespeichert werden. Nach dem Speichern kann die Seite mit dem X-Symbol über der Überschrift geschlossen werden. Danach erscheint die Übersicht über die erzeugte Seite, hier sieht man, dass das Karten-Plugin eingebunden wurde (siehe Abbildung 12).

Wenn jetzt die soeben erstellte Seite in der linken Spalte des Hauptfensters, also da wo die Weltkugel ist, mit Rechtsklick ausgewählt, kommt ein Dropdown-Menü. Jetzt den ersten Eintrag “Ansehen” aus der Liste auswählen und die Seite kann im Browser angesehen werden.

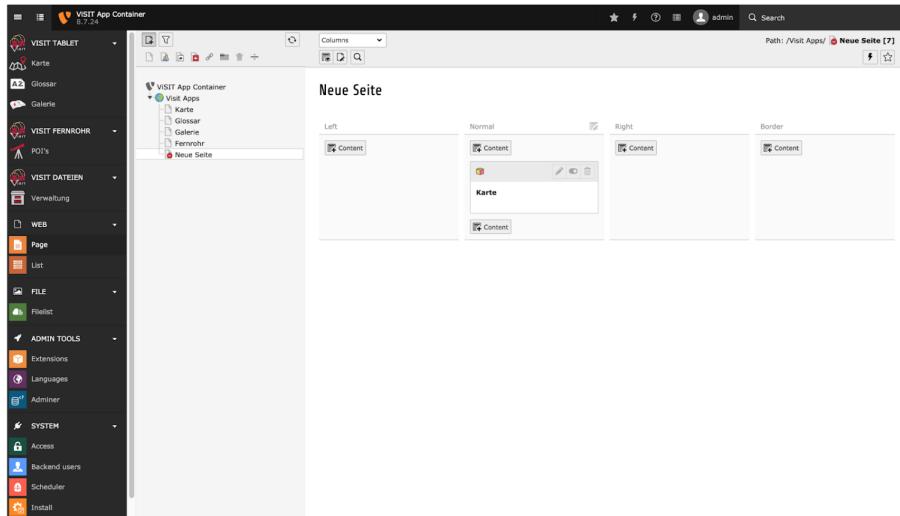


Abbildung 12: Einbindung eines Plug-Ins

3.9 Erstellung eines Templates

Wenn ein neuer Raum hinzugefügt wird, wird ein neuer Webroot benötigt, dieser wird mittels Template erzeugt und stellt den Seitenanfang der Webseite dar. Sollen mehrere gleiche Applikationen laufen, dann wird für jede einzelne Applikation ein eigenes Template benötigt. Für die Darstellung der Inhalte auf der Webseite werden Templates verwendet. Ein Template ist eine Design- und Formatierungsvorlage für ein Dokument, es ist das Grundgerüst, welches mit Inhalten gefüllt werden muss. Um ein Template in TYPO3 zu erstellen, muss im ersten Schritt unter WEB das "Template" aus der Modul-Liste auf der linken Seite ausgewählt werden. Danach erscheinen die Template-Werkzeuge in der rechten Hälfte des Hauptfensters, hier kann "Template für neue Website erstellen" ausgewählt werden. Jetzt kann in der Werkzeuleiste des Hauptbereichs das Dropdown-Feld aufgemacht und "Info/Bearbeiten" ausgewählt werden. In der Übersicht im Hauptbereich erscheinen die wichtigsten Template-Informationen. Danach "Vollständigen Template-Datensatz bearbeiten" auswählen. Hier kann im Raster "Allgemeines" der Titel des Templates hinzugefügt werden, des weiteren muss der Inhalt aus "Setup" gelöscht werden. Danach ins Raster "Enthält" wechseln, hier können verschiedene Objekte aus der rechten Spalte "Verfügbare Objekte" in die linke Spalte "Ausgewählte Objekte" verschoben werden, hier muss jedoch auf die Reihenfolge dieser Objekte geachtet werden. Hier zuerst auf "Fluid Content Elements (fluid_styled_content)" klicken, dann wandert dieses Objekt in die linke Spalte. Das gleiche mit dem Objekt "tablets (visit_tablets)". Jetzt befinden sich beide Objekte in der linken Spalte unter "Ausgewählte Objekte". Damit diese Änderungen gespeichert werden, muss wieder auf das Speichern-Symbol über der Überschrift im Hauptbereich geklickt werden. Wenn die Webseite auf dem localhost:80/ aufgerufen wird, erscheint die Karte.

4 KARTEN-APPLIKATION

4.1 Einfügen der Daten in die Karten-Applikation

Dazu aus der Modulleiste links unter der Obergruppe VISIT TABLET die Karte auswählen. Im linken Teil des Hauptfensters ist der Seitenbaum zu sehen und rechts befindet sich die Kartenübersicht. Oben links im rechten Teil des Hauptfensters befindet sich ein Menü-Button, wird dieser angeklickt, wird eine weitere dunkelblaue

Spalte zwischen dem Seitenbaum und dem Arbeitsbereich im Hauptfenster sichtbar (siehe Abbildung 13).

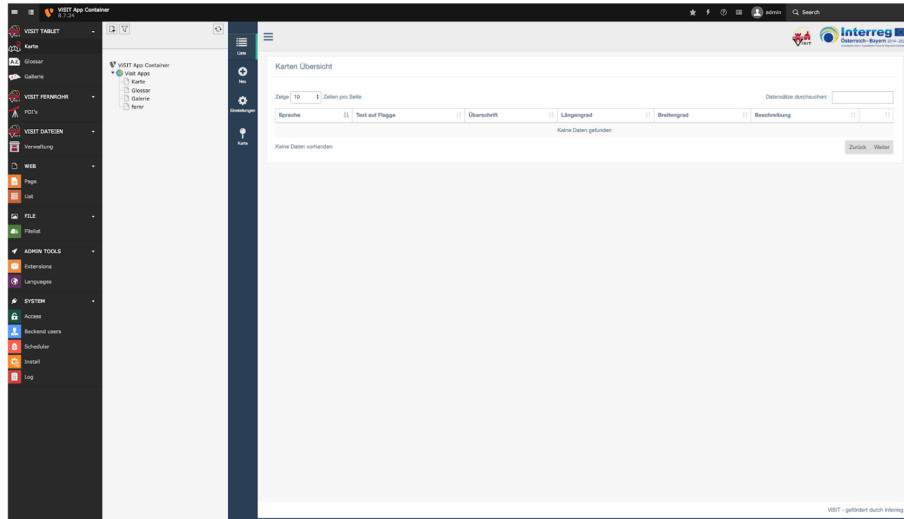


Abbildung 13: Leere Kartenübersicht

4.2 Erstellung der Startseite für die Karten-Applikation

Dazu in der dunkelblauen Leiste "Einstellungen" auswählen (siehe Abbildung 14).

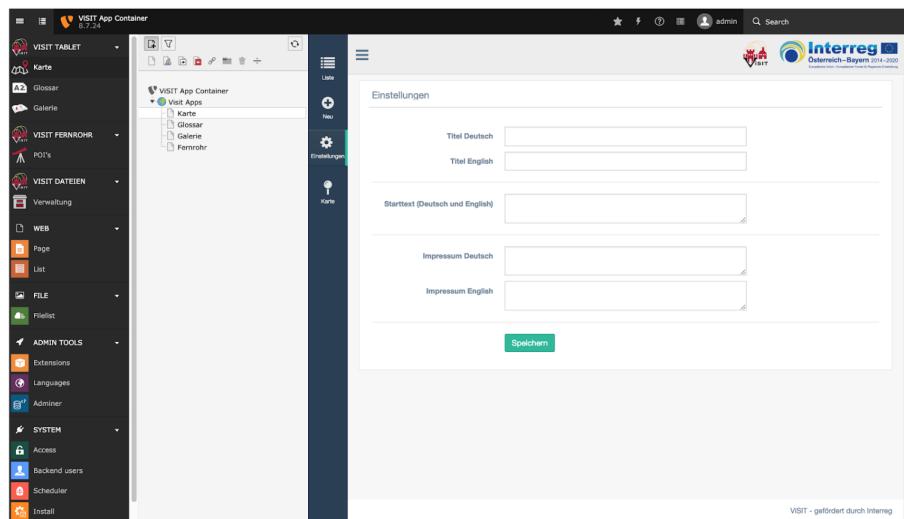


Abbildung 14: Erstellung der Startseite für die Karten-Applikation

Die Startseite wird dem Besucher als erstes angezeigt, auf dieser kann der Besucher die gewünschte Sprache auswählen. Damit das Design des Textes immer gleich aussieht, gibt es unter <https://github.com/VISIT-Dev/appbundle> in der README.md ein Beispiel für die Startseite der Tablets (siehe Abbildung 15).

Für jede Sprache wird ein Titel sowie der Impressumstext benötigt. Jetzt werden die beiden Texte aus der zuvor genannten Github-Seite benötigt (siehe Abbildung 15). Der erste Text ist der Starttext, dieser beinhaltet die HTML-Elemente Überschrift, Paragraph und Buttons über welche die gewünschte Sprache gewählt werden kann. Die einzelnen Texte in den Tags können mit dem gewünschten Text überschrieben werden (siehe Abbildung 16).

Wenn weitere Sprachen außer Deutsch und Englisch verfügbar sind, können weitere

```

Visit Apps
Typo3 Extension that contains all applications

Beispiel Startseite für Tablets

<div class="modal-content">
<div class="modal-body">
<h1 class="modal-title">Lorem ipsum dolor Headline</h1>
<h3>Lorem ipsum dolor Subline historisch</h3>
<br>
<p>
    Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris eget elit a lacus sollicitudin
    <br><br>
    Vivamus placerat aliquet posuere. Phasellus aliquet dolor arcu, non semper orci congue vitae.
</p>
</div>
</div>
<div class="modal-footer">
    <button type="button" class="btn btn-primary lang-btn btn-lg" data-dismiss="modal" onclick="initMap('DE')">DE</button>
    <button type="button" class="btn btn-primary lang-btn btn-lg" data-dismiss="modal" onclick="initMap('EN')">EN</button>
</div>
</div>

Beispiel Seite für Impressum

<div class="modal-content lang-content show-de">
<div class="modal-body">
<h1 class="modal-title">Impressum DE</h1>
<p>
    Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris eget elit a lacus sollicitudin
    <br>
    <br />
    <div class="row imprint-logos">
        <div class="col-3">
            
        <div class="col-6">
            
        <div class="col-3">
            
    </div>
    <p>
        Vivamus placerat aliquet posuere. Phasellus aliquet dolor arcu, non semper orci congue vitae.
    </p>
</div>
<div class="modal-footer">
    <button type="button" class="btn btn-primary lang-btn btn-lg" data-dismiss="modal">Schließen</button>
</div>
</div>

```

Abbildung 15: Text für die Startseite der Applikationen, zu finden auf <https://github.com/ViSIT-Dev/appbundle>

Sprachauswahl-Buttons durch das Markieren des gesamten `<button>`-Tags ausgewählt werden, dann kopieren und darunter einfügen, erstellt werden. Zwei Sachen müssen beachtet werden: einerseits muss in der `onclick='initMap('...')'`-Methode die der Sprache entsprechende ID eingegeben werden und für den Button der Pfad für die Flagge im Image-Tag angegeben werden. Dazu muss zuvor im das benötigte Flaggen-Icon vorzugsweise im PNG-Format im entsprechenden Ordner gespeichert werden und der Pfad angepasst werden
`src="/typo3/sysext/core/Resources/Public/Icons/Flags/PNG/DE.png"`.

4.3 Neues Kartenelement hinzufügen

Mit einem Klick auf "Neu" kann ein neues Kartenelement - Point of Interest - hinzugefügt werden (siehe Abbildung 17).

Ein neues Kartenelement - Point of Interest - benötigt eine Überschrift, eine Unterüberschrift ist optional, einen Text auf der Flagge und eine Beschreibung. Optional können auch weitere Medien hinzugefügt werden. Die geografische Position kann entweder über den Längen- und Breitengrad manuell eingetippt werden oder mittels setzen der Stecknadel auf die Karte, dann werden die Längen- und Breitengrade dieser Stecknadel übernommen. Ist alles vollständig ausgefüllt, kann die Eingabe mit "Anlegen" am Seitenende gespeichert werden. Nach dem Klick gelangt man zu der Kartenübersicht, wo alle eingefügten Elemente angeführt sind, jedes dieser Elemente kann sowohl nochmals bearbeitet oder auch wieder gelöscht werden.

Klickt man im Seitenbaum mit der rechten Maustaste auf Karte, dann kann man

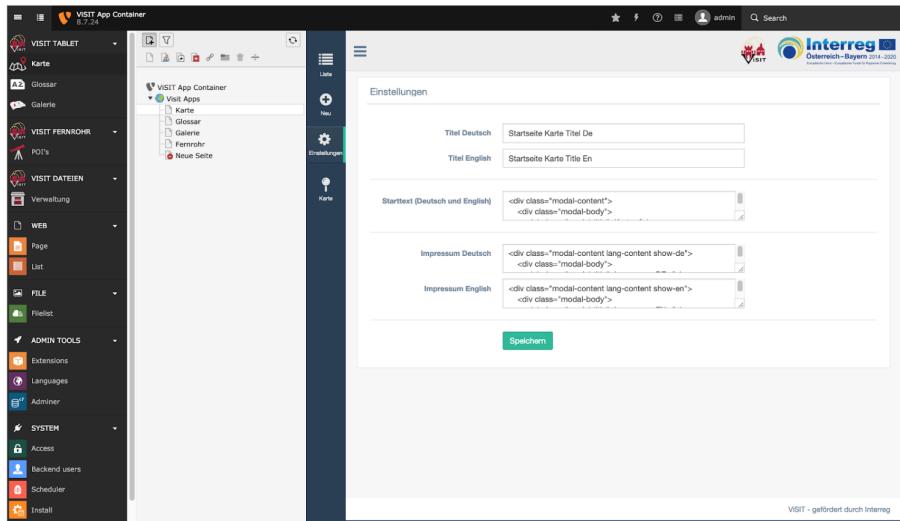


Abbildung 16: Erstellung der Startseite für die Karten-Applikation

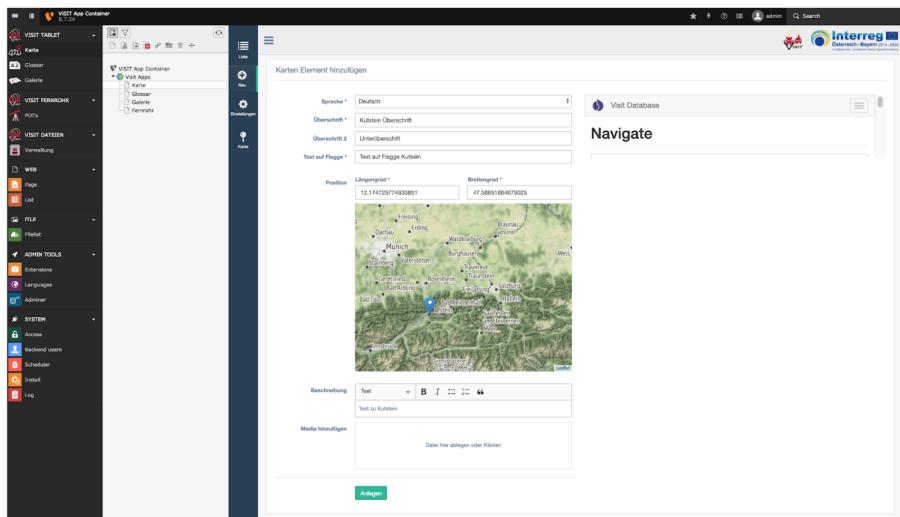


Abbildung 17: Ein neues Kartenelement hinzufügen

die angelegten Kartenelemente im Browser anzeigen lassen.
Jedes Kartenelement muss sowohl auf Deutsch als auch auf Englisch angelegt werden (siehe Abbildung 18).

Dabei wird zuerst die Startseite angezeigt, auf welcher der Besucher seine Sprache wählen kann (siehe Abbildung 19).

Nach der Auswahl der Sprache wird dem Besucher die Karte mit den einzelnen Kartenelementen - Point of Interest - auf dem Tablet angezeigt (siehe Abbildung 20).

Jetzt kann der Besucher eine Flagge auswählen, zu welcher er mehr Informationen haben möchte und via Klick öffnet sich der seitliche Infobereich auf der rechten Seite (siehe Abbildung 21).

4.4 Bearbeitung und Löschen von angelegten Kartenelementen

Die Kartenelemente können jederzeit bearbeitet oder gelöscht werden. Dies geht indem zuerst die Listenansicht in der dunkelblauen Leiste ausgewählt wird. In weiterer Folge kann jedes einzelne Element (Zeile) einzeln bearbeitet oder gelöscht werden. Zum Bearbeiten wird auf das blaue Stiftsymbol auf der rechten Seite klicken, zum Löschen des Objekts, den orangen Müllkübel.

The screenshot shows the VIST App Container interface. On the left, there's a sidebar with various menu items like 'VIST TABLET', 'Karte', 'Glossar', 'Galerie', 'VIST FERNROHR', 'POI's', 'VIST DATEIEN', 'Verwaltung', 'WEB', 'FILE', 'ADMIN TOOLS', 'SYSTEM', and 'Scheduler'. The main area is titled 'Karten Übersicht' and contains a table with five rows of data. The columns are 'Sprache' (Language), 'Text auf Flagge' (Text on Flag), 'Überschrift' (Title), 'Längengrad' (Longitude), 'Breitengrad' (Latitude), and 'Beschreibung' (Description). The data includes entries for Kufstein, Rosenheim, Saalfelden, and Saalfelden EN. At the bottom, there are buttons for 'Zurück' (Back) and 'Weiter' (Next).

Abbildung 18: Listenansicht über alle angelegten Kartenelemente

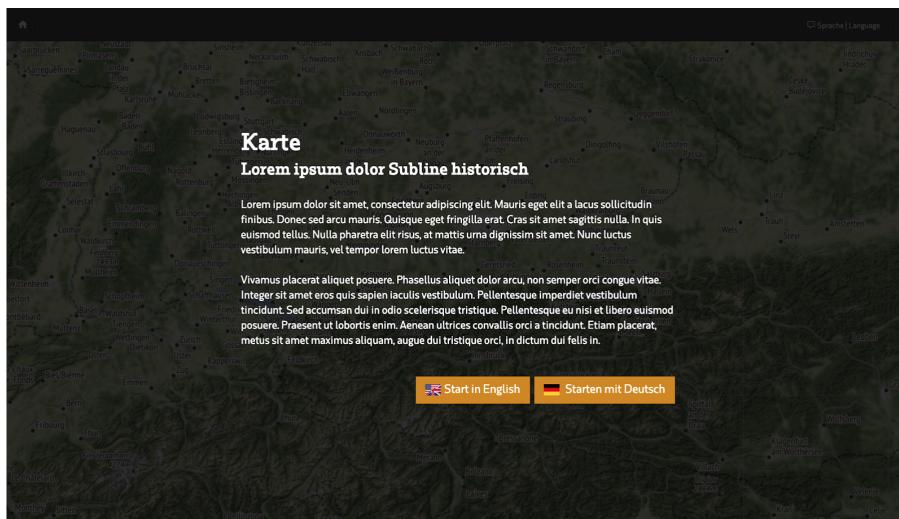


Abbildung 19: Startseite der Karten-Applikation

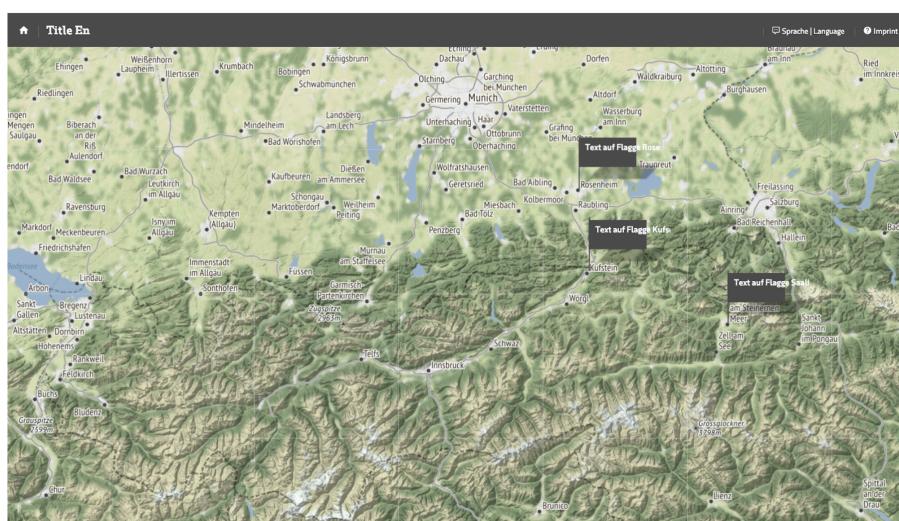


Abbildung 20: Ansicht der Karte im Browser

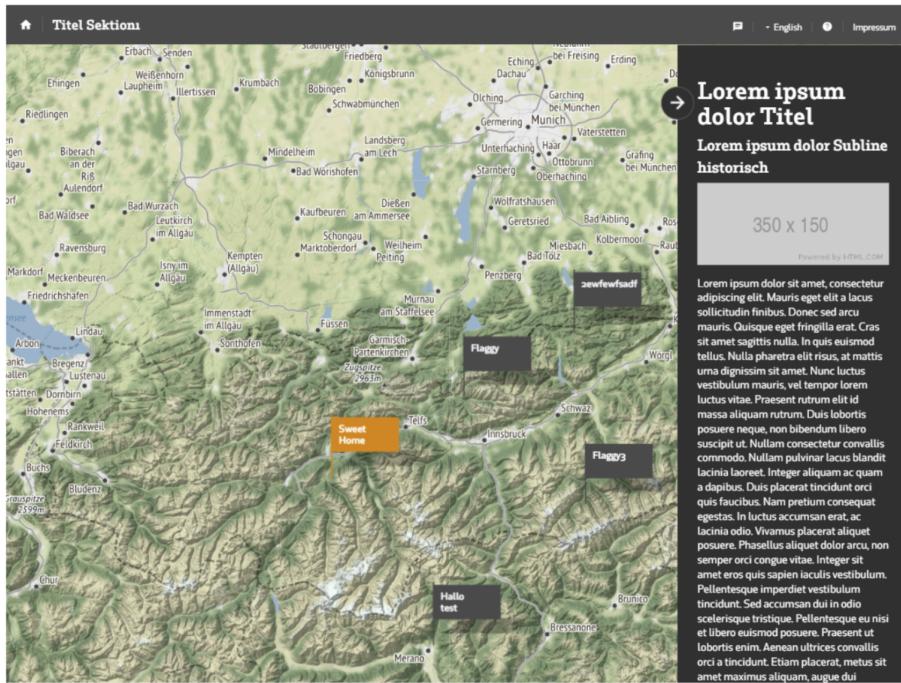


Abbildung 21: Kartenelement - Point of Interest - mit Detailinformation

5 GLOSSAR-APPLIKATION

In der Glossar-Applikation werden Insassen des Gefängnisses aufgelistet. Die Applikation ist in zwei Bereiche aufgeteilt, rechts werden die Insassen aufgelistet, wählt der Benutzer einen Namen aus dieser Liste aus, so werden die Details zu dieser Person in der rechten Spalte angezeigt. Die Auflistung der Insassen ist alphabetisch nach Vornamen beziehungsweise, wenn vorhanden, nach dem Nachnamen 22. Die Auflistung kann auch nach Ereignis, Zelle oder VIP erfolgen, dies kann der Besucher in der oberen Menüzeile auswählen.

A-Z	Ereignisse	Zellen	VIP
A	Max Mustermann, Zelle, Haftdauer		
B	Name Insasse lorem, Zelle Haftdauer		
C	Name Insasse lorem, Zelle Haftdauer		
D	Name Insasse lorem, Zelle Haftdauer		
E	Josef Zaliwski, Zelle, Haftdauer		
F	Max Mustermann, Zelle, Haftdauer		
G	Name Insasse lorem, Zelle Haftdauer		
H	Name Insasse lorem, Zelle Haftdauer		
I	Josef Zaliwski, Zelle, Haftdauer		
J	Max Mustermann, Zelle, Haftdauer		
K	Name Insasse lorem, Zelle Haftdauer		
L	Name Insasse lorem, Zelle Haftdauer		
M	Name Insasse lorem, Zelle Haftdauer		
N	Name Insasse lorem, Zelle Haftdauer		
O	Name Insasse lorem, Zelle Haftdauer		
P	Name Insasse lorem, Zelle Haftdauer		
Q	Name Insasse lorem, Zelle Haftdauer		
R	Name Insasse lorem, Zelle Haftdauer		
S	Name Insasse lorem, Zelle Haftdauer		
T	Name Insasse lorem, Zelle Haftdauer		
U	Name Insasse lorem, Zelle Haftdauer		
V	Name Insasse lorem, Zelle Haftdauer		
W	Name Insasse lorem, Zelle Haftdauer		
X	Name Insasse lorem, Zelle Haftdauer		
Y	Name Insasse lorem, Zelle Haftdauer		

Abbildung 22: Ansicht der Glossar-Applikation auf einem Tablet

5.1 Einpflegen der Daten in die Glossar-Applikation

Dazu aus der Modulleiste links unter der Obergruppe VISIT TABLET das Glossar auswählen. Im linken Teil des Hauptfensters ist der Seitenbaum zu sehen und rechts befindet sich die Insassen-Übersicht [23](#).

Sprache	Name	Geburtsdatum	Todesdatum	Inhaftiert am	Freigelassen am	Event	Zelle
Deutsch	Aa Insasse 1	01.01.1780	01.01.1810	01.01.1805	01.01.1807	Event 1 DE	Zelle 1
Deutsch	Bb Insasse 2	02.02.1600	02.02.1628	02.02.1620	02.05.1620	Event 2 DE	Zelle 2
Deutsch	Cc Insasse 3	03.03.1700		03.03.1735	03.03.1740	Event 3 DE	Zelle 1
English	Aa Prisoner 1	01.01.1780	01.01.1810	01.01.1805	01.01.1807	Event 1 DE	Zelle 1
English	Bb Prisoner 2	02.02.1600	02.02.1628	02.02.1620	02.05.1620	Event 2 DE	Zelle 2
English	Cc Prisoner 3	03.03.1700		03.03.1735	03.03.1740	Event 3 DE	Zelle 1

Abbildung 23: Übersicht über alle angelegten Insassen

5.2 Erstellung der Startseite für die Glossar-Applikation

Das Glossar kann mit einem Klick auf Glossar in der Modulleiste konfiguriert werden [24](#).

Sprache	Name	Geburtsdatum	Todesdatum	Inhaftiert am	Freigelassen am	Event	Zelle
Keine Daten gefunden							

Abbildung 24: Konfiguration der Glossar-Applikation

Die Startseite der Applikation kann unter Einstellungen in der dunkelblauen Leiste konfiguriert werden. Dafür benötigt man den Text für die Startseite <https://github.com/ViSIT-Dev/appbundle>. Dieser Text muss in die entsprechenden Inputfelder kopiert werden [25](#).

Nachdem die Startseite befüllt wurde, wird sie dem Besucher als erstes angezeigt, auf dieser kann der Besucher dann die gewünschte Sprache auswählen [26](#).

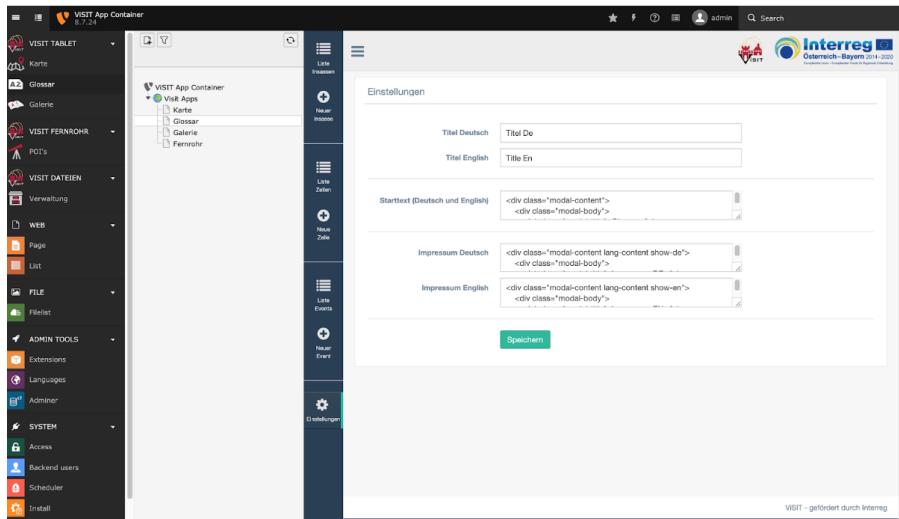


Abbildung 25: Befüllung der Startseite der Glossar-Applikation

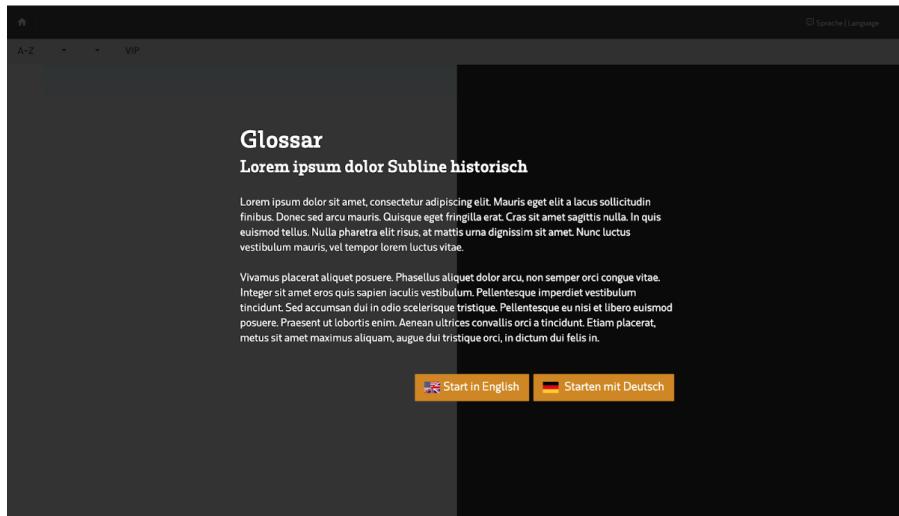


Abbildung 26: Startseite der Glossar-Applikation

5.3 Hinzufügen einer neuen Zelle

Über einen Klick auf Neue Zelle in der dunkelblauen Leiste kann eine neue Zelle angelegt werden [27](#).

Die Zelle benötigt sowohl einen deutschen als auch englischen Zellennamen. Mit einem Klick auf Anlegen, werden die eingegebenen Daten dauerhaft gespeichert.

Alle angelegten Zellen können über Liste Zellen in der dunkelblauen Leiste angegesehen werden [29](#).

5.4 Hinzufügen eines Events

Ein Event benötigt ebenfalls einen deutschen und einen englischen Namen, mit einem Klick auf Anlegen werden die eingegebenen Daten dauerhaft gespeichert [30](#). In Liste Events in der dunkelblauen Leiste können alle Events angezeigt werden.

5.5 Neuen Insassen anlegen

Über die Auswahl Neuer Insasse in der dunkelblauen Leiste kann ein neuer Insasse angelegt werden. Die Felder Geburtsdatum, Todestag, Inhaftiert sowie Freigelassen

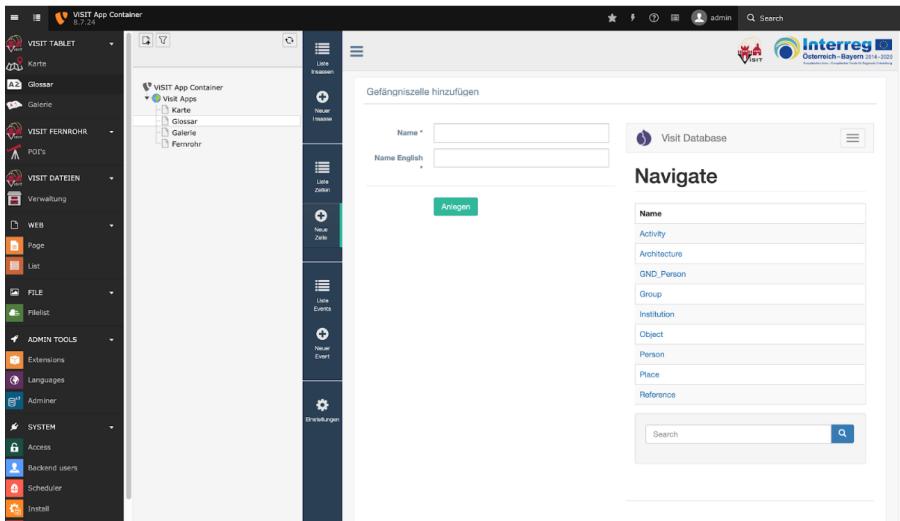


Abbildung 27: Hinzufügen einer neuen Zelle

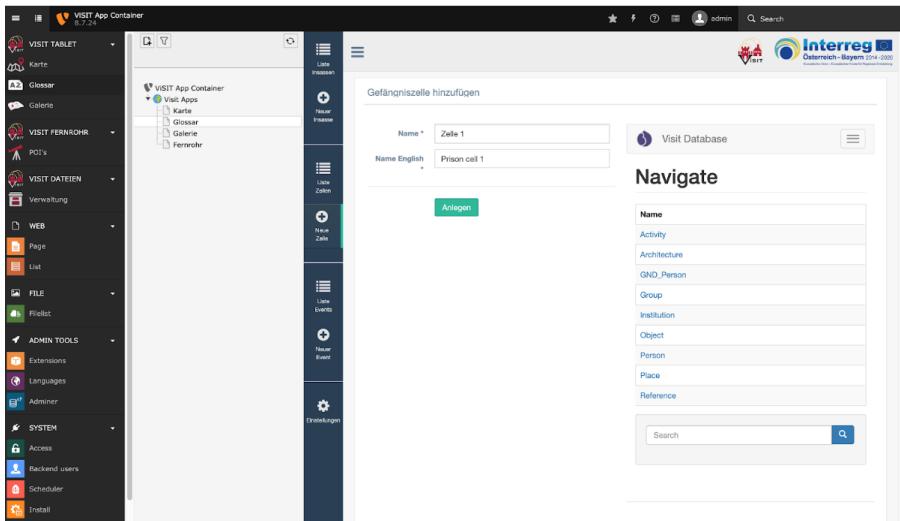


Abbildung 28: Erstellen einer neuen Zelle

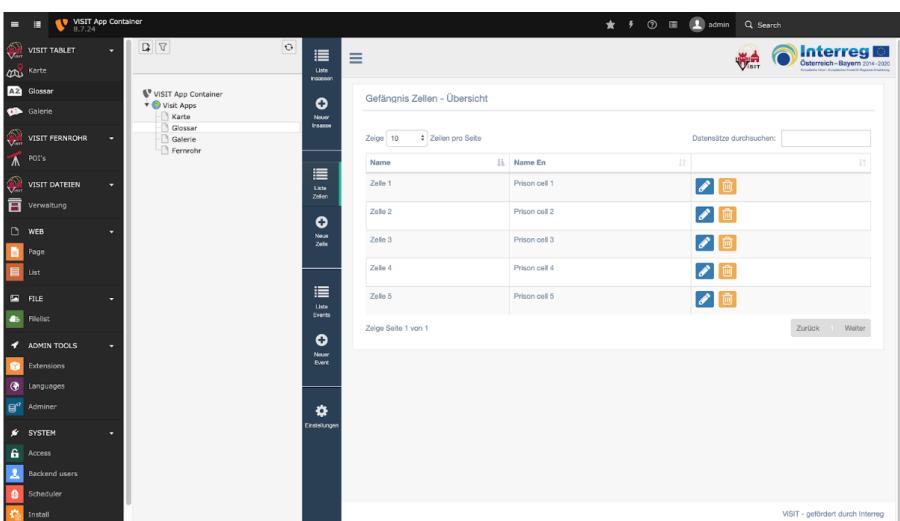


Abbildung 29: Angelegte Zellen in der Listenübersicht

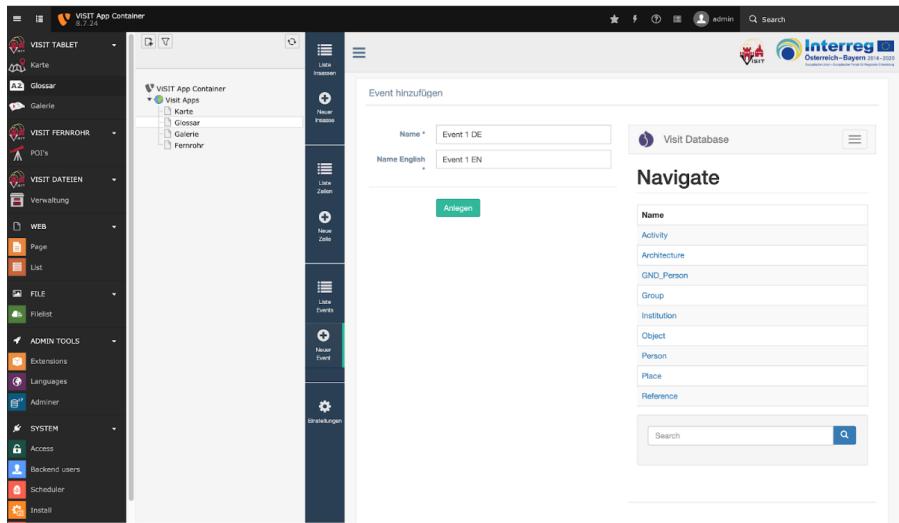


Abbildung 30: Hinzufügen eines neuen Events

Name	Name En	
Event 1 DE	Event 1 EN	
Event 2 DE	Event 2 EN	
Event 3 DE	Event 3 EN	
Event 4 DE	Event 4 EN	

Abbildung 31: Angelegte Events in der Listenansicht

sind Datumsfelder. Ist eines der Daten nicht bekannt, kann das Feld freigelassen werden [32](#). Legt man einen Insassen an, muss bei Zelle sowie Event ein bereits angelegte Zelle beziehungsweise ein angelegtes Event angegeben werden. Klickt man auf die Pfeile im Inputfeld, so öffnet sich ein Dropdown Menü und hier kann die entsprechende Zelle beziehungsweise das entsprechende Event ausgewählt werden [33](#). Aus diesem Grund müssen die dazugehörigen Zellen und Events vor dem Anlegen eines Insassens angelegt werden. Die angelegten Insassen können über Liste Insassen in der dunkelblauen Leiste angesehen werden [34](#). Klickt man mittels rechtem Mausklick auf Glossar im Seitenbaum und wählt Show aus, kann die Seite im Browser angesehen werden. Die Insassen werden im Browser, nach dem Auswählen der bevorzugten Sprache, angezeigt.

Abbildung 32: Anlegen eines neuen Insassens

Abbildung 33: Anlegen eines Insassens

Abbildung 34: Angelegte Insassen in der Listenansicht

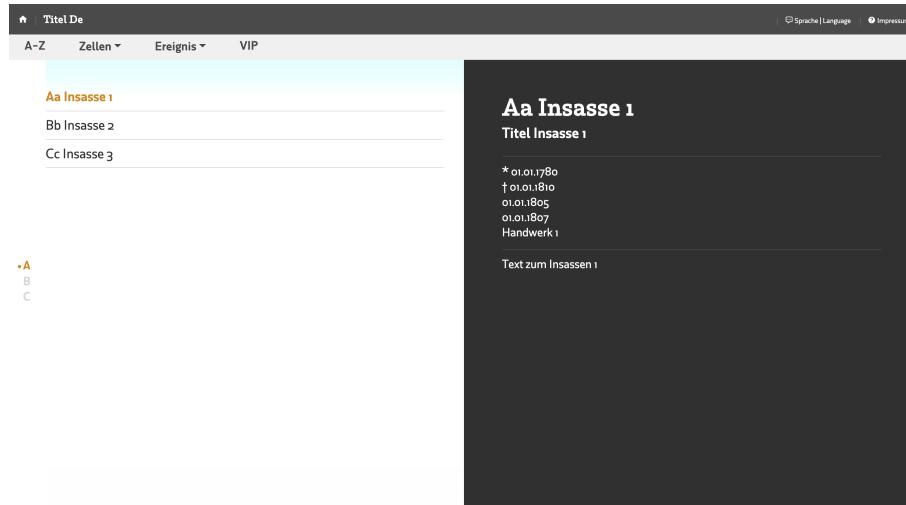


Abbildung 35: Ansicht der Insassen im Browser

5.6 Bearbeitung und Löschung von Insassen

6 GALERIE-APPLIKATION

6.1 Einpflegen der Daten in die Glossar-Applikation

6.2 Erstellung der Startseite für die Glossar-Applikation

6.3 Neuen Insassen anlegen

7 DATEIVERWALTUNG

7.1 Zugangsdaten zum Dateimanagement

Das Dateimanagement ist eine Applikation, mit der die Daten in der ViSIT Medien-Datenbank verwaltet werden können. Die Verwaltung befindet sich im TYPO3 Backend. Dafür müssen zuerst aus der Modulleiste die Extensions ausgewählt werden. In weiterer Folge kann im Hauptfenster die Visit App gefunden werden. Durch einen Klick darauf kommt man zu den Einstellungen [36](#). Diese Einstellungen sind im ganzen System gleich. Diese Login Daten sind sensibel, da sie Zugang zum Peer-to-

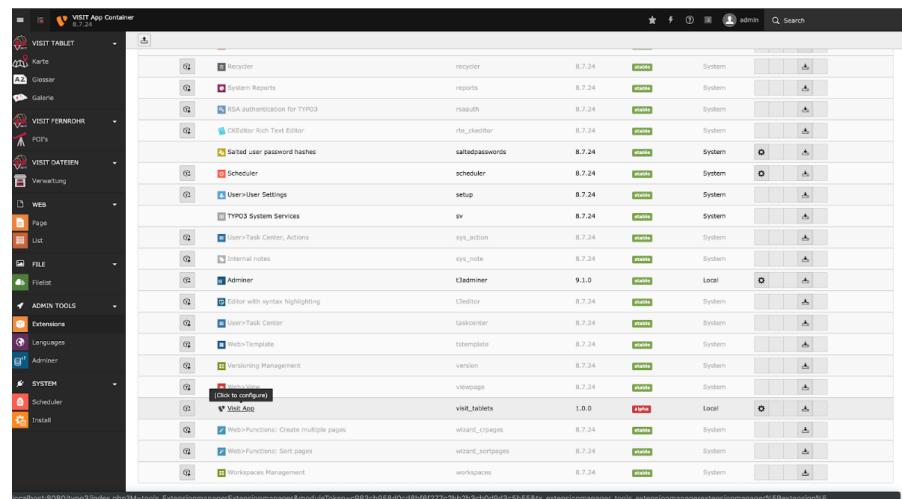


Abbildung 36: Einstellung der ViSIT App Extension

Peer-Netzwerk geben, deshalb sind sie nicht im Docker angeführt. Sie bekommen diese Zugangsdaten (API User, Password for API User sowie die Syncthing Master ID) entweder von der betreuenden Firma oder von einem anderen ViSIT-Partner [37](#).

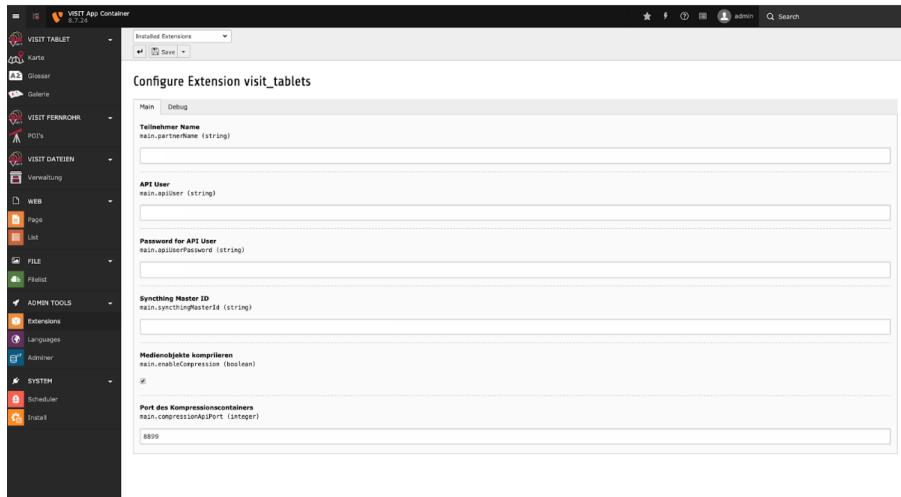


Abbildung 37: ViSIT App Extensions

Ist aber ein ViSIT-Partner nicht am Hochladen von Dateien interessiert, dann werden die Zugangsdaten zum Dateisystem nicht benötigt.

7.2 ViSIT-Partner-Liste

Um zu sehen, welche Partner Zugriff zum Peer-to-Peer-Netzwerk haben, muss zuerst aus der Modulleiste unter VISIT DATEIEN die Verwaltung ausgewählt werden. In weiterer Folge aus der dunkelblauen Leiste die Partner Liste auswählen. Hier sind alle ViSIT-Partner, die Zugang zum Dateiverwaltungssystem haben, aufgelistet.

Diese Partner haben Zugriff auf die bereits abgelegten Dateien in der Mediendatenbank im Peer-to-Peer-Netzwerk. Diese Dateien können in weiterer Folge beim Anlegen von Objekten ausgewählt werden. Ist eine benötigte Datei noch nicht vorhanden, so muss sie in die Datenbank eingepflegt werden. Dies geschieht über Verwaltung und dann Datei hochladen.

In der Abbildung [38](#) ist ersichtlich, dass es drei Benutzer gibt, die Zugriff auf die Mediendatenbank haben (K***, M*** und P***), im Feld ID ist ihre ViSIT-Zugangs-ID ebenfalls ersichtlich. Die einzelnen Partner sind im Grunde nur Verzeichnis mit Namen und der entsprechenden Partner-ID. Der Benutzer Öffentlich ist ein öffentliches Verzeichnis, er besitzt keine ID, er entspricht sozusagen einem virtuellen Benutzer, er ist nur aus organisatorischen Gründen angeführt.

7.3 Upload von 3D-Objekten und Bildern, Videos und anderen Dateien

Damit eine Datei in die Datenbank geladen werden kann, muss zuerst in der Modulleiste unter der Obergruppe VISIT DATEIEN die Verwaltung ausgewählt werden. Danach erscheint im Hauptfenster die Datei Liste über alle bereits verfügbaren Dateien [39](#). Der Zugriffsmodifikator der bereits hochgeladenen Dateien kann entweder *public*, *visit* oder *private* sein.

Zugriff *public* bedeutet, dass jeder diese Datei sehen und verwenden kann. Bei Zugriff *visit* hat jeder ViSIT-Partner Zugang zu der Datei, wohingegen Zugriff *private* nur für die eine Installation zugänglich ist.

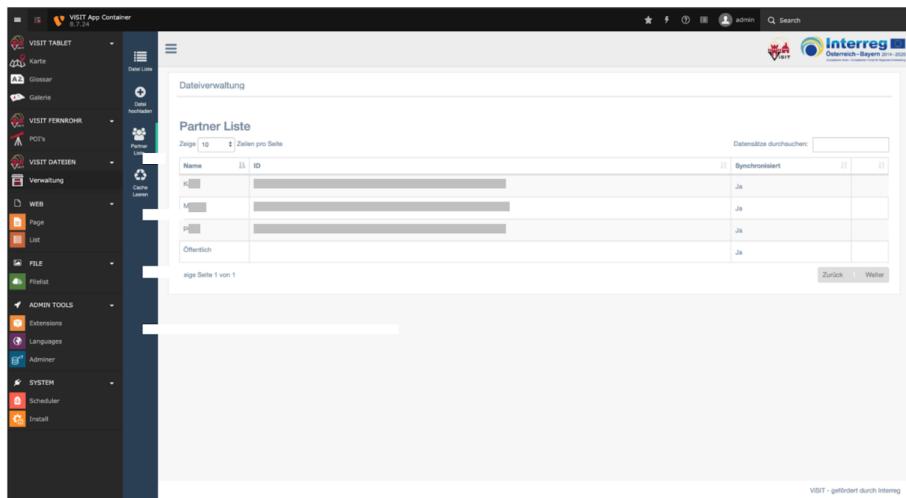


Abbildung 38: Ansicht der ViSIT-Partner mit Zugang zur Mediendatenbank im Peer-to-Peer-Netzwerk

Die Dateien mit dem Zugriffsmodifikator *public* und *visit* können mit einem Klick auf das blaue Downloadsymbol in der entsprechenden Zeile heruntergeladen werden.

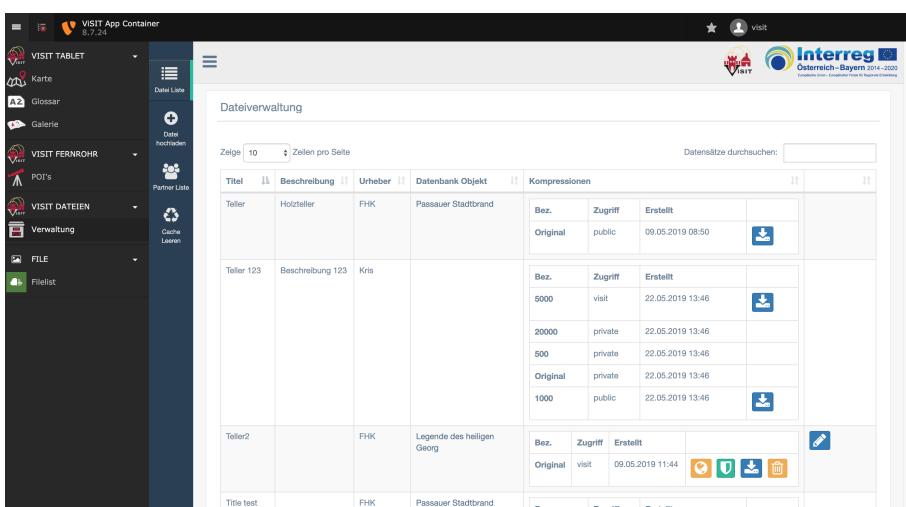


Abbildung 39: Ansicht der bereits verfügbaren Dateien in der Dateiliste

7.4 Hochladen von Dateien

Für das Hochladen eines Medienobjekts ist das Ausfüllen des Titels, des Urhebers, die ViSIT Partner ID sowie der ViSIT Partner Name zwingend erforderlich. Die Beschreibung sowie Hochgeladen von sind optional.

Im nächsten Schritt muss der Dateipfad für die Datei (entweder 3D Objekt oder Bild, Video und weitere Dateien) angegeben werden.

Als nächstes muss die Datei einem bereits angelegten Objekt (Entität) zugeordnet werden. Dies wird in der rechten Spalte des Hauptfensters gemacht. Hier sieht man die Visit Database, hier muss das entsprechende Objekt ausgewählt werden. Klickt man beispielsweise auf "Place", dann öffnen sich alle bereits angelegten Orte, aus diesen kann dann ein Ort ausgewählt werden. Wird ein Ort ausgewählt, dann erscheinen die Detailinformationen zu diesem in der gleichen Spalte.

Hat man zufällig ein falsches Objekt ausgewählt, kann mittels Klick entweder auf das Burger-Menü rechts neben Visit Database geklickt werden und dann Navigate ausgewählt werden oder unter Visit Database im Breadcrumb-Menü auf Navigate klicken, dann kommt man ebenfalls zur Übersicht.

Wurde das korrekte Objekt gefunden, dann können die Daten übernommen werden. Dies geschieht mittels Klick auf "Daten übernehmen" 40. Nach dem Klick wird das Feld oben rechts bei "Gewählte Entität*: automatisch ausgefüllt. Danach kann das Objekt zur ViSIT Datenbank hinzugefügt werden. Dies geschieht mittels Klick auf Medienobjekt zur ViSIT Datenbank hinzufügen". Danach kommen zwei Meldungen, einerseits eine Meldung, dass die Kompression des Medienobjektes wurde erfolgreich gestartet und dass die Datei erfolgreich hinzugefügt wurde 41.

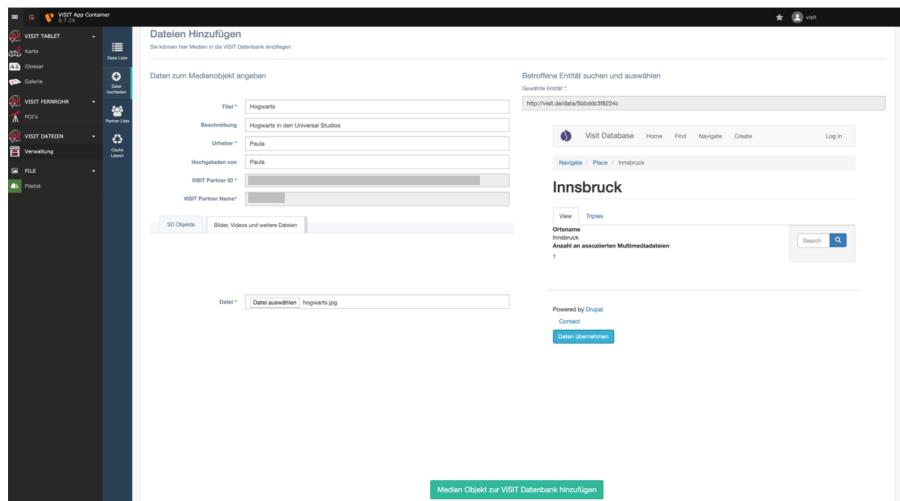


Abbildung 40: Ansicht des ausgefüllten Formulars für den Dateiupload

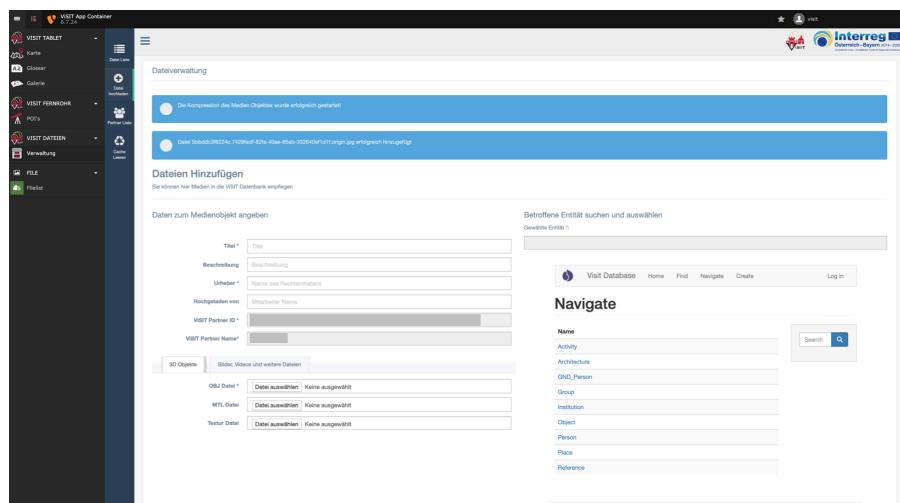


Abbildung 41: Bestätigungs Nachrichten nach einem erfolgreichen Upload

Nachdem die Datei in der ViSIT Datenbank gespeichert wurde, kann sie in der Dateiliste angesehen werden 42. Der Zugriff auf die hochgeladene Datei ist per default immer *private*. Will man die Datei verwenden, muss sie zuerst heruntergeladen werden. Dies geschieht mit einem Klick auf das blaue Download-Symbol, jetzt ist die Datei lokal gespeichert. lokale kopie, falls partner löscht

Titel	Beschreibung	Urheber	Datenbank Objekt	Kompressionen																		
Hogwarts	Hogwarts in den Universal Studios	Paula		<table border="1"> <tr> <td>Bez.</td> <td>Zugriff</td> <td>Erstellt</td> </tr> <tr> <td>Original</td> <td>private</td> <td>04.06.2019 13:34</td> </tr> </table>	Bez.	Zugriff	Erstellt	Original	private	04.06.2019 13:34												
Bez.	Zugriff	Erstellt																				
Original	private	04.06.2019 13:34																				
Teller	Hobzeller	FHK	Passauer Stadtbstrand	<table border="1"> <tr> <td>Bez.</td> <td>Zugriff</td> <td>Erstellt</td> </tr> <tr> <td>Original</td> <td>public</td> <td>09.05.2019 08:50</td> </tr> </table>	Bez.	Zugriff	Erstellt	Original	public	09.05.2019 08:50												
Bez.	Zugriff	Erstellt																				
Original	public	09.05.2019 08:50																				
Teller 123	Beschreibung 123	Kris		<table border="1"> <tr> <td>Bez.</td> <td>Zugriff</td> <td>Erstellt</td> </tr> <tr> <td>5000</td> <td>visit</td> <td>22.05.2019 13:46</td> </tr> <tr> <td>20000</td> <td>private</td> <td>22.05.2019 13:46</td> </tr> <tr> <td>500</td> <td>private</td> <td>22.05.2019 13:46</td> </tr> <tr> <td>Original</td> <td>private</td> <td>22.05.2019 13:46</td> </tr> <tr> <td>1000</td> <td>public</td> <td>22.05.2019 13:46</td> </tr> </table>	Bez.	Zugriff	Erstellt	5000	visit	22.05.2019 13:46	20000	private	22.05.2019 13:46	500	private	22.05.2019 13:46	Original	private	22.05.2019 13:46	1000	public	22.05.2019 13:46
Bez.	Zugriff	Erstellt																				
5000	visit	22.05.2019 13:46																				
20000	private	22.05.2019 13:46																				
500	private	22.05.2019 13:46																				
Original	private	22.05.2019 13:46																				
1000	public	22.05.2019 13:46																				
Teller2		FHK	Legende des heiligen Georg	<table border="1"> <tr> <td>Bez.</td> <td>Zugriff</td> <td>Erstellt</td> </tr> <tr> <td>Original</td> <td>visit</td> <td>09.05.2019 11:44</td> </tr> </table>	Bez.	Zugriff	Erstellt	Original	visit	09.05.2019 11:44												
Bez.	Zugriff	Erstellt																				
Original	visit	09.05.2019 11:44																				
Title test		FHK	Passauer Stadtbstrand																			

Abbildung 42: Hochgeladene Datei in der Listenübersicht

7.5 Veröffentlichung einer Datei im ViSIT-Netzwerk

Wenn eine Datei hochgeladen wurde, welche für alle ViSIT-Partner zur Verfügung stehen soll, dann muss diese Datei explizit freigegeben bzw. veröffentlicht werden. Dies [anno4j1]

8 TOBI – APP FRAMEWORK

TODO rename label and heading
[anno4j2]

9 KOMPRESSION

9.1 Motivation

Unabhängig davon, ob man eine Ausstellung in einem Museum gestaltet oder eine Webseite erstellt, lässt sich fast jede für Besucher oder Benutzer konzipierte Darstellung durch den Einsatz von Bildern aufwerten. Dieses lässt sich durch den Einsatz von 3D-Modellen, mit welchen der Benutzer durch Drehen, Zoomen, etc. interagieren kann, noch deutlich steigern. Abgesehen davon, dass dies in einer Ausstellung nur durch den Einsatz von Rechnern, wie beispielsweise von Tablets, möglich ist, sind damit jedoch einige technische Herausforderungen verbunden.

Einerseits erwartet der Benutzer ein möglichst realistisches Erlebnis, was nur durch hochauflöste 3D-Modelle und den damit einhergehenden großen Datenmengen möglich ist. Dennoch soll das Modell möglichst ohne Latenz angezeigt werden und flüssige Interaktion erlauben. Soll die Darstellung darüber hinaus auf einem mobilen Endgerät des Benutzers, wie beispielsweise einem Smartphone, stattfinden, das einerseits nur eingeschränkte Rechenkapazität bietet und auf das andererseits die gesamten Daten für jeden Benutzer separat übertragen werden müssen, entsteht hier ein Gegensatz, für den ein geeigneter Kompromis zu finden ist.

Dieser Kompromis besteht darin, die 3D-Modelle nicht in der höchsten verfügbaren Auflösung dem Benutzer darzustellen, sondern in einer dem Anwendungsfall und dem Endgerät angemessenen Größe. Beispielsweise ist für eine ansprechende Anzeige auf einem Smartphone-Bildschirm eine geringere Auflösung notwendig als auf der Workstation eines Museumsmitarbeiters mit entsprechend großem Bildschirm, die auch dementsprechend leistungsfähig ist, und über welche dieser Mitarbeiter das angezeigte Objekt erforschen möchte. Aus diesem Grund ist es sinnvoll, in der Medien-Datenbank die gespeicherten 3D-Modelle in verschiedenen Auflösungen bereit zu halten, weshalb diese in der Regel beim Hochladen in die Datenbank automatisch komprimiert werden.

Die eben beschriebene Problematik tritt nicht nur bei 3D-Modellen auf, sondern auch bei herkömmlichen zweidimensionalen Bildern. Da auch solche Medien in der Medien-Datenbank gespeichert werden sollen, sind auch Bilddateien zu komprimieren, was sich sehr einfach durch das Verringern der Auflösung bewerkstelligen lässt. Von jedem hochgeladenen Bild werden also komprimierte Versionen in verschiedenen Auflösungsstufen erstellt, auf welche anschließend zugegriffen werden kann.

In diesem Kapitel wird nach einer kurzen Erläuterung der theoretischen Grundlagen auf die an die ViSIT-Medien-Datenbank angebundene Kompressions-Komponente und deren Voraussetzungen, ihre Funktionsweise, die Installation und die Bedienung, die über eine Web-Oberfläche erfolgt, eingegangen. Außerdem werden die zur Verfügung stehenden Konfigurationsoptionen eingehend erläutert. Abschließend wird die Schnittstelle (API), über welche die Kompressions-Komponente unabhängig von der dafür verfügbaren Web-Oberfläche oder der ViSIT-Medien-Datenbank angesprochen werden kann, spezifiziert.

9.2 Grundlagen

Bei 3D-Modellen gilt es grundsätzlich zwischen der Geometrie, durch welche die Form der Oberfläche eines Objekts beschrieben wird, und der Textur, durch welche die Färbung der Oberfläche ausgedrückt wird, zu unterscheiden. Während die Geometrie obligatorisch für ein 3D-Modell ist, muss nicht zwangsläufig eine Textur existieren. Nicht jedes Digitalisierungsverfahren ist in der Lage, Texturdaten zu erfassen, wie beispielsweise ein Laserscanner. Im Folgenden werden Aufbau und Kompression dieser beiden Bestandteile eines 3D-Modells behandelt.

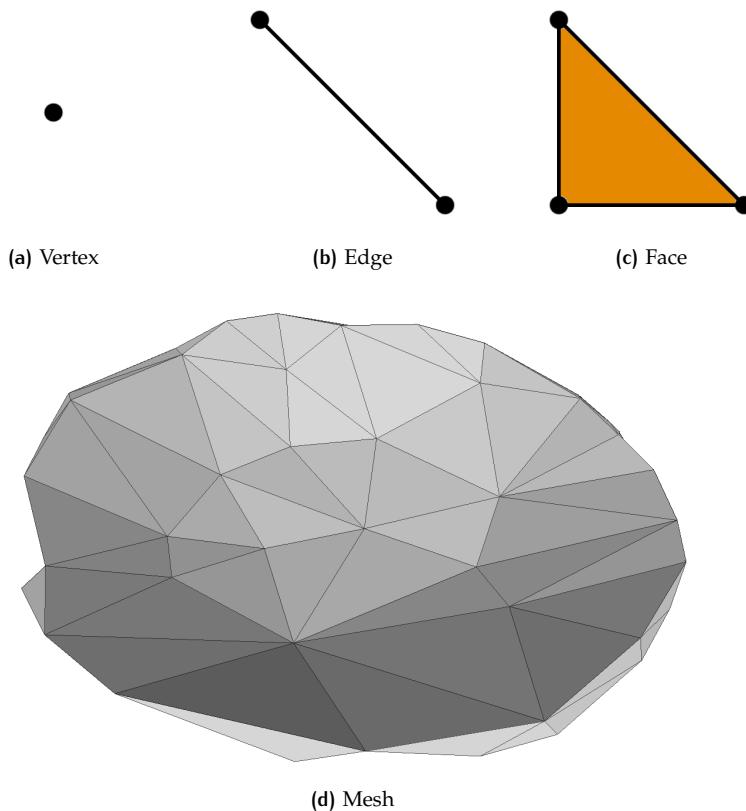


Abbildung 43: Grundlegende Elemente der Geometrie eines 3D-Modells

9.2.1 Geometrie von 3D-Modellen

Herkömmliche 3D-Modelle, die ausschließlich die Oberfläche eines Objekts und nicht dessen Inneres beschreiben, bestehen aus mehreren Punkten im dreidimensionalen Raum, die *Vertices* genannt werden. Diese Punkte werden zu Flächen, den sogenannten *Faces* verbunden, wobei es sich hier in vielen Fällen um Dreiecke handelt. Die Anzahl der Ecken eines Faces wird als dessen *Ordnung* bezeichnet. Die Kanten dieser Flächen, die Verbindungen zwischen zwei Vertices darstellen, werden *Edges* genannt. Die Gesamtheit aller Vertices und Faces eines 3D-Modells wird auch als *Mesh* bezeichnet. Die soeben genannten Begriffe werden in Abbildung 43 grafisch dargestellt.

Während bei einfachen Dateiformaten, wie beispielsweise dem STL-Format¹, für jedes Face die Koordinaten eines jeden Eckpunkts separat gespeichert werden, wird die dadurch verursachte Redundanz zum Beispiel bei OBJ-Dateien² vermieden, indem zunächst alle Vertices in Form der Koordinaten einmalig angegeben werden und anschließend bei der Definition der Faces auf diese Vertexdefinition indexbasiert zugegriffen wird.

Bei Meshes, die ein reales Objekt beschreiben, sollte dieser möglichst einer orientierbaren stetigen zweidimensionalen Mannigfaltigkeit, die in den dreidimensionalen Raum eingebettet ist, entsprechen [Bot+10, S. 3]. Dies hat zur Folge, dass für jeden Punkt im Raum eindeutig entschieden werden kann, ob er im Inneren oder im Äußeren des Objekts liegt. Dies impliziert beispielsweise, dass kein Edge Teil von mehr als zwei Faces sein kann. Jedoch sind auch an die Vertices bestimmte Bedingungen zu stellen, wobei für Details auf [Bot+10, S. 11f] verwiesen wird.

¹ http://www.fabbers.com/tech/STL_Format

² <http://www.martinreddy.net/gfx/3d/OBJ.spec>

9.2.2 Textur von 3D-Modellen

Die Textur eines 3D-Modells beschreibt dessen Färbung der Oberfläche, wodurch ihr eine äußerst wichtige Bedeutung für das visuelle Erlebnis beim Betrachten des Modells zukommt. Sie wird in der Regel getrennt von den eigentlichen Geometriedaten in einer separaten Bild-Datei gespeichert. Für jedes Face wird dann durch die sogenannten *Texturkoordinaten* festgelegt, welcher Ausschnitt des Textur-Bildes auf diesem Dreieck angezeigt wird. Pro Face werden also für jeden Eckpunkt zwei Werte gespeichert, durch welche eine eindeutige Position im Bild definiert wird. Da in der Regel bei den meisten Vertices alle angrenzenden Faces die gleichen Texturkoordinaten verwenden, wird dieses Paar von Werten beispielsweise bei OBJ-Dateien nur einmal gespeichert, worauf anschließend bei der Beschreibung der Faces indexbasiert zugegriffen wird.

Manchem Leser stellt sich hier die Frage, ob generell die Verwendung von einem Paar Texturkoordinaten pro Vertex, welches dann für alle angrenzenden Faces verwendet wird, nicht ausreichend wäre. Hier spielt jedoch die Geometrie des 3D-Modells, genauer deren *Topologie*, eine wichtige Rolle. Entspricht diese einer (verzerrten) Ebene, so wäre diese Vereinfachung in der Tat ausreichend. Betrachtet man aber beispielsweise eine Kugel, so lässt sich das zweidimensionale Bild nicht über die Kugel legen, ohne dass eine Kante entsteht, an welcher mindestens zwei Ränder des Bildes aneinandergrenzen. Genau entlang dieser Linie, dem sogenannten *Texture Seam*, befinden sich dann die Vertices, für welche je nach angrenzendem Face verschiedene Texturkoordinaten verwendet werden müssen. Unabhängig von der soeben dargelegten Notwendigkeit dieser Texture Seams lässt sich durch eine sinnvolle Unterteilung der Textur auch die Qualität erhöhen, indem für alle Bereiche des 3D-Modells eine ähnliche Auflösung verwendet wird und durch starke Streckungen oder Stauchungen verursachte Verzerrungen der Textur auf dem Modell minimiert werden.

9.2.3 Kompression der Geometrie

Eine der wichtigsten Herangehensweisen zur Kompression von 3D-Modellen ist die Reduktion von Vertices und damit einhergehend die Verringerung der Anzahl an Faces. Ausgehend von einem hoch aufgelösten Modell mit einer hohen Anzahl an Vertices ist die auf den ersten Blick einfachste Vorgehensweise das Entfernen eines möglichst unwichtigen Vertices. Das dadurch entstehende Loch im Mesh muss anschließend geschlossen werden. Dadurch entsteht jedoch im Allgemeinen ein Face mit einer höheren Ordnung, was oft unerwünscht ist. In diesem Fall muss das entstehende Loch mit Dreiecken gefüllt werden, wofür es keine eindeutige und daher auch unterschiedlich gute Lösungen gibt. Stattdessen verwenden viele Kompressionsverfahren, wie auch der in dieser Kompressions-Komponente verwendete Algorithmus, sogenannte *Edge-Collapse-/Half-Edge-Collapse*-Operationen.

Bei einer solchen Edge-Collapse-Operation, wie sie in Abbildung 44 dargestellt ist, werden zwei durch ein Edge verbundene Vertices zu einem neuen Vertex verschmolzen. Dabei degenerieren die an dieses Edge angrenzende Faces und werden entfernt. Zusätzlich lassen sich neben dem kontrahierten Edge noch weitere Edges entfernen. Entspricht der Mesh lokal einer Mannigfaltigkeit, werden durch eine Edge-Collapse-Operation also zwei Faces, drei Edges und ein Vertex entfernt. Vor dem Durchführen einer derartigen Operation muss jedoch überprüft werden, ob die in [Bot+10, S. 118f] erläuterte *Link Condition* erfüllt ist, da andernfalls die Topologie des Meshes durch die Operation verändert werden kann. Entspricht der neue Vertex einem der beiden ursprünglichen Vertices, so spricht man von einem Half-Edge-Collapse.

Es verbleibt jedoch die Frage, welche Paare von Vertices verschmolzen werden sollen. Hierfür wird in [GH97] ein Verfahren vorstellt, das mithilfe von Quadriken jeder möglichen Edge-Collapse-Operation einen Wert für die damit verbundenen Kosten zuweist. Iterativ werden nun die Vertices aus Operationen mit den gering-

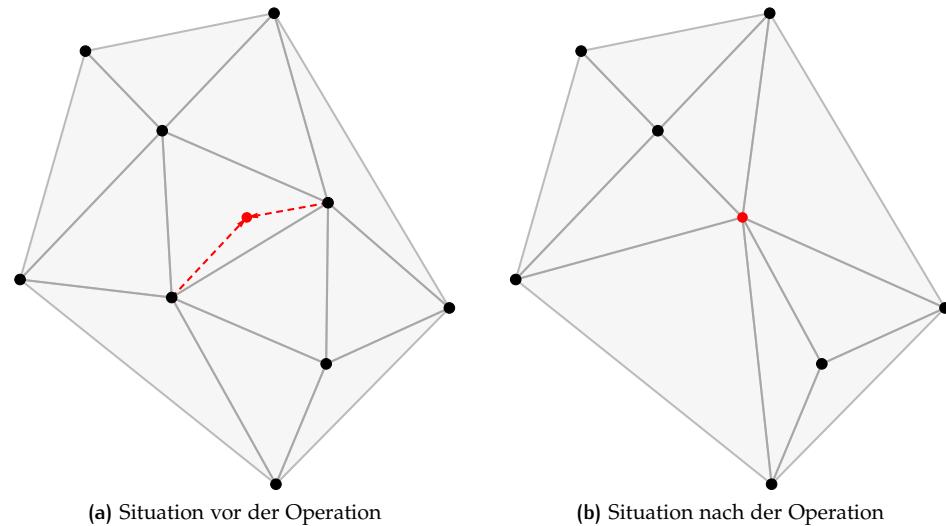


Abbildung 44: Beispiel einer Edge-Collapse-Operation. Die Zielposition der beiden an das Edge angrenzenden Vertices ist rot markiert.

sten Kosten verschmolzen, woraufhin die Kosten der Operationen, die sich auf die umliegenden Vertices beziehen, aktualisiert werden müssen.

Der Algorithmus aus [GH97] ist jedoch nur auf nicht-texturierte Meshes anwendbar. Möchte man dieses Verfahren verwenden, um 3D-Modelle mit Textur zu komprimieren, so müssen einerseits die Verschmelzungsoperationen auch auf die Texturkoordinaten angewandt werden, was insbesondere entlang von Texture Seams eine sehr sorgfältige Implementierung erfordert, andererseits müssen aber auch diese Texturkoordinaten in die Berechnung der Kosten für die jeweilige Operation miteinbezogen werden. Abweichungen in der resultierenden Oberfläche des 3D-Modells haben hier Verzerrungen in der Textur zur Folge. Dies gilt auch für Texture Seams, bei welchen es darüber hinaus nicht vermeiden lässt, dass auch Bereiche der als Textur verwendeten Bilddatei für die Einfärbung der Faces verwendet werden, die im ursprünglichen Modell überhaupt nicht parametrisiert waren und somit beliebigen Inhalt aufweisen können. Es empfiehlt sich daher, die Ränder der Bereiche in der Bilddatei mit einem ähnlichen Farbton wie die eigentliche Textur zu färben, was bei den meisten Textur erzeugenden Programmen automatisch geschieht. Die daher notwendige Erweiterung des bestehenden Kompressions-Algorithmus auf texturierte Meshes wird in [GH98] vorgenommen, wobei Texture Seams dort kaum behandelt werden. Die für die Kompressions-Komponente erstellte Implementierung behandelt jedoch auch diese Bereiche auf eine sinnvolle Art und Weise.

9.2.4 Kompression der Textur

Während die Kompression der Modellgeometrie insbesondere bei texturierten Meshes nichttrivial ist, gestaltet sich die Kompression der Texturdaten relativ einfach. Die Textur wird in einer herkömmlichen Bilddatei gespeichert, deren Dateigröße direkt von der Auflösung des darin gespeicherten Bildes abhängt. Wird die Auflösung des Bildes verringert, was der Funktionsumfang eines jeden brauchbaren Bildbearbeitungsprogramms zulässt, schlägt sich dies auch in der Größe der Texturdatei nieder. Darüber hinaus sind weitere aus der Bildverarbeitung bekannte Kompressionsverfahren einsetzbar, wie zum Beispiel die JPEG-Komprimierung³.

Allerdings muss darauf geachtet werden, dass die zusammen mit der Modellgeometrie gespeicherten Texturkoordinaten auch nach der Kompression der Textur wohldefiniert sind. Besonders angenehm ist hier jedoch die Tatsache, dass diese Texturkoordinaten sich nicht auf die Pixel beziehen, sondern stets im Intervall von

³ <https://www.ece.ucdavis.edu/cerl/reliablejpeg/compression/>

null bis eins liegen, wobei sich null auf den oberen bzw. linken Rand und eins auf den unteren bzgl. rechten Rand der Texturdatei bezieht. Aus diesem Grund sind die Texturkoordinaten gänzlich unabhängig von der Auflösung der die Textur beinhaltenden Bilddatei und diese Datei kann ohne Modifizierung der Geometriedatei komprimiert werden. Diese Aussage gilt nicht nur für die Anpassung der Auflösung, sondern auch für andere Kompressionsverfahren aus der Bildverarbeitung.

9.3 Voraussetzungen

Die Kompressions-Komponente dient zur Kompression von Mediendaten, die im Dateimanagement auf dem Lokalen Anwendungsserver registriert sind. Die Kompressions-Komponente läuft innerhalb eines separaten Docker-Containers auf dem Server, wofür die Software Docker⁴ auf dem als Host-System fungierenden Server installiert sein muss. Eine direkte Interaktion des Benutzers mit der Kompressions-Komponente findet in der Regel nicht statt. Es wird jedoch eine Web-Oberfläche zur Administration und Konfiguration des Systems bereitgestellt.

Da die zu komprimierenden Dateien auf dem Host-System gespeichert sind, worauf sowohl von der Medien-Datenbank als auch von der Kompressions-Komponente zugegriffen wird, ist es nicht möglich, das Kompressions-System unabhängig von der Medien-Datenbank auszuführen. Andernfalls würden in der Medien-Datenbank hochgeladene Dateien der Kompressions-Komponente nicht zur Verfügung stehen.

9.3.1 Hardwarevoraussetzungen

Die Anforderungen an die Ressourcen, die sich für den LAS bzw. den Docker-Container der Kompressions-Komponente ergeben, hängen in erster Linie von der Größe der zu komprimierenden Mediendateien ab. Während die Prozessorleistung primär die Dauer der Kompressions-Prozesse bedingt, beschränkt der zur Verfügung stehende Arbeitsspeicher die maximale Größe der Eingabedaten nach oben. Da der Algorithmus zur Kompression von 3D-Modellen kaum Parallelisierungs-Möglichkeiten bietet, werden die Berechnungen derzeit nicht auf mehrere Prozessorkerne oder auf die Grafikkarte verlagert. Experimente während der Implementierung haben gezeigt, dass Modelle mit ca. 8 Mio. Dreiecken auf einem aktuellen gut ausgestatteten Büreorechner (Intel-Core-i7-Prozessor, 32 GB RAM) komprimiert werden können. Da insbesondere der Arbeitsspeicher der limitierende Faktor ist, spielt jedoch der Speicherbedarf der anderen auf dem System laufenden Anwendungen eine wichtige Rolle.

9.3.2 Softwarevoraussetzungen

Die Kompressions-Komponente wurde in der Programmiersprache Java implementiert, weshalb zur Ausführung ein Java Runtime Environment (JRE) im Docker-Container installiert sein muss. Für die Kompression von Bild- und Texturdateien wird im Hintergrund die Software ImageMagick verwendet. Es ergeben sich also die folgenden beiden Software-Abhängigkeiten:

- Java Runtime Environment⁵ (getestet mit Version 1.8.0)
- ImageMagick⁶ (getestet mit Version 7.0.8)

Da diese Software-Produkte jedoch bei der Installation des entsprechenden Docker-Containers automatisch installiert werden, muss dies nicht manuell durch den Benutzer vollzogen werden.

⁴ <https://www.docker.com/>

⁵ <https://java.com/de/download/>

⁶ <https://imagemagick.org/index.php>

Dateiendung	Informationen	Optional	Notwendigkeit
.obj	Geometriedaten	Nein	Genau eine Datei erforderlich
.mtl	Materialdefinitionen	Ja	Maximal eine Datei möglich
.jpg/.jpeg/.png	Texturdaten	Ja	Maximal eine Datei, nur falls auch MTL-Datei spezifiziert wird

Tabelle 1: Für ein 3D-Modell notwendige und optionale Dateien

9.3.3 Abgrenzungskriterien

Das Absetzen eines Kompressions-Auftrags wird in der Regel von der Medien-Datenbank-Komponente angestoßen. Dennoch wird eine Benutzeroberfläche zum manuellen Absetzen von Kompressions-Aufträgen zur Verfügung gestellt, die jedoch aufgrund der Notwendigkeit der manuellen Angabe von Identifikatoren keinen hohen Bedienkomfort bietet.

An Medientypen werden Bilder und 3D-Modelle unterstützt. Andere Medientypen können mit dieser Komponente nicht komprimiert werden. An Bilddaten können alle Bildformate komprimiert werden, die von der Software ImageMagick unterstützt werden. An 3D-Modellen können texturierte und nicht-texturierte Modelle komprimiert werden, die im OBJ-Format⁷ gespeichert werden. Andere Dateien müssen zunächst manuell durch Drittsoftware in das OBJ-Format übersetzt werden. Pro Modell ist aufgrund von Limitierungen in der Medien-Datenbank maximal eine Material- und eine Texturdatei zulässig. Genauere Informationen über die für ein 3D-Modell notwendigen und möglichen Dateien sind in folgender Tabelle 1 dargestellt. Der implementierte Algorithmus wäre jedoch prinzipiell in der Lage, auch Objekte mit mehreren Texturdateien zu komprimieren.

Die in Abschnitt 9.3.1 benannten Hardware-Voraussetzungen gilt es zu beachten.

Bei 3D-Modellen wird vorausgesetzt, dass die Eingabedaten den Voraussetzungen mannigfaltigen Meshes, wie sie in Abschnitt 9.2.1 beschrieben wurden, genügen. Für Modelle, welche diese Eigenschaft nicht aufweisen, wird keine Garantie bezüglich des Ergebnisses des Kompressionsvorgangs gegeben. Mögliche Konsequenzen sind ein Fehlschlagen des Kompressionsvorgangs oder unbrauchbare Resultate, auch wenn ein Fehlschlagen in den Experimenten nicht beobachtet wurde. Das Eingabemodell sollte möglichst ausschließlich aus Dreiecken bestehen. Faces höherer Ordnung werden beim Einlesen naiv in Dreiecke umgewandelt, wobei auch hier keine Garantie bezüglich der Qualität des Ergebnisses gegeben werden kann.

Eine Darstellung der Eingangsdaten oder deren komprimierter Versionen findet innerhalb der Kompressions-Komponente nicht statt. Die Resultate des Kompressionsvorgangs werden lediglich als Datei im System abgelegt.

Die Kompressions-Komponente bietet keinerlei Benutzerauthentifizierung oder sonstige Zugriffsbeschränkungen. Die Komponente muss anderweitig vor dem Zugriff Dritter geschützt werden. Für den Zugriff auf die API und die Konfigurations- und Administrationsoberfläche kann lediglich eine Whitelist mit IP-Adressen hinterlegt werden, für welche der Zugriff exklusiv gewährt wird.

9.4 Funktionsweise

Die Kompressions-Komponente dient zum Abarbeiten von Kompressions-Aufträgen, wobei letztere die Kompression einer medialen Repräsentation eines Objekts in mehreren Auflösungsstufen umfasst. Die Medien-Dateien stehen dem Kompressions-System über das Dateisystem zur Verfügung. In der Regel wird ein Kompressions-Auftrag direkt nach dem Hochladen einer Medien-Datei in die ViSIT-Medien-Datenbank abgesetzt.

Bei Abarbeitung eines Kompressions-Auftrags werden zunächst die technischen Metadaten aus der Meta-Datenbank abgerufen. Diese technischen Metadaten wer-

⁷ <http://www.martinreddy.net/gfx/3d/OBJ.spec>

```

1  {
2      "title" : string,
3      "description" : string,
4      "objectTripleID" : string,
5      "objectTripleURL" : string,
6      "mediaTripleID" : string,
7      "mediaTripleURL" : string,
8      "MIMEType" : string
9      "createDate" : timestamp,
10     "creatorID" : string,
11     "creatorName" : string,
12     "rightholder" : string,
13     "uploader" : string,
14     "files" : {
15         "origin" : {
16             "uploadDate" : timestamp,
17             "accessLevel" : string,
18             "license" : string,
19             "fileSize" : long,
20             "paths" : [string],
21             "fileTypeSpecificMeta" : object
22         },
23         ...
24     }
25 }
```

Listing 22: Spezifikation des die technischen Metadaten beinhaltenden JSON-Objekts

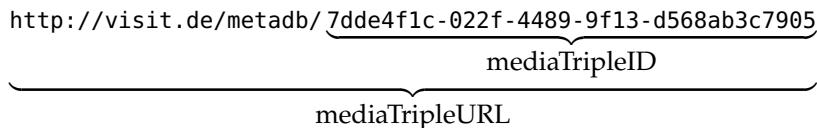


Abbildung 45: Beziehung von mediaTripleID und mediaTripleURL anhand eines Beispiels

den für jede mediale Repräsentation gespeichert und umfassen wichtige Informationen über die dazugehörigen Medien-Dateien und die bestehenden Kompressionsstufen. Sie können außerdem Kontext-Informationen über eine mediale Repräsentation beinhalten, so speichert beispielsweise die Kompressions-Komponente bei 3D-Modellen die Anzahl der Vertices und der Faces. Die technischen Metadaten liegen in der Meta-Datenbank im JSON-Format⁸ vor, wobei hierfür die in Listing 22 dargestellte Spezifikation gilt. Im Anschluss wird unterschieden, ob es sich bei der zu komprimierenden Medien-Datei um ein Bild oder ein 3D-Modell handelt. Ausschlaggebend hierfür ist der im Kompressions-Auftrag angegebene MIME-Typ.

Die beiden in diesem Objekt verwendeten URLs bezeichnen die ID des Metadatums, welches durch die mediale Repräsentation beschrieben wird, in der Meta-Datenbank (objectTripleURL) bzw. die ID der digitalen Repräsentation selbst (mediaTripleURL), wie in Abbildung Abbildung 58 veranschaulicht wird. Diese beiden Werte haben die Form einer URL und können somit nicht als Dateinamen verwendet werden, wie es in der Medien-Datenbank geschieht. Jedoch verwenden alle in diesem Kontext auftretenden URLs das gleiche Präfix, wodurch die Eindeutigkeit auch bei einer Beschränkung auf das Suffix gewährleistet bleibt, welches sich als Dateiname eignet. Das Suffix der objectTripleURL wird als objectTripleID, das der mediaTripleURL als mediaTripleID bezeichnet. Ein Beispiel hierzu ist in Abbildung 45 zu sehen.

Die Dateien werden in der Medien-Datenbank im Format

```
1 objectTripleID.mediaTripleID.compressionLevelID.Dateiendung
```

Listing 23: Schema der Dateinamen in der Medien-Datenbank, das ebenfalls durch das Kompressions-System verwendet wird

abgespeichert, worauf ebenfalls durch das Kompressions-System zugegriffen wird. Hierbei bezeichnet compressionLevelID die Bezeichnung für die jeweilige Kompre-

⁸ <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>

sions-Stufe. Für die Originaldatei wird hierbei `origin` verwendet, für 3D-Modelle die Vertexanzahl und für Bilder der in der Konfiguration festgelegte Titel für die jeweilige Auflösungsstufe.

Im folgenden wird auf die Semantik der einzelnen Werte dieses JSON-Objekts eingegangen:

- `title`: Ein vom Benutzer festgelegter Titel des Objekts
- `description`: Eine vom Benutzer festgelegte Beschreibung des Objekts
- `objectTripleID`: Wie oben beschrieben
- `objectTripleURL`: Wie oben beschrieben
- `mediaTripleID`: Wie oben beschrieben
- `mediaTripleURL`: Wie oben beschrieben
- `createDate`: Der UNIX-Timestamp, an dem die Datei in die Medien-Datenbank geladen wurde
- `creatorID`: Die Syncthing-ID des Netzwerkteilnehmers, von dem die Datei hochgeladen wurde
- `creatorName`: Der Name des Netzwerkteilnehmers, von dem die Datei hochgeladen wurde
- `rightholder`: Der Rechteinhaber an der Medien-Datei
- `uploader`: Name der Person, welche die Datei in die Medien-Datenbank geladen hat
- `MIMETYPE`: MIME-Typ der Medien-Datei (z. B. „image/png“ für ein PNG-Bild oder „text/plain“ für ein 3D-Modell)
- `files`: Ein Objekt, das für jede zur Verfügung stehende Kompressions-Stufe (inklusive der Originaldatei) weiterführende Informationen bereitstellt. Diese werden als Objekt in einer Eigenschaft gespeichert, deren Titel der Bezeichnung der jeweiligen Kompressions-Stufe entspricht, also beispielsweise `origin` für die Originaldatei. Dieses Objekt weist das folgende Schema auf
 - `uploadDate`: Der UNIX-Timestamp, an dem diese Kompressions-Stufe erstellt wurde
 - `accessLevel`: Zugriffsberechtigung für die Kompressions-Stufe in der Medien-Datenbank, kann entweder `public`, `private` oder `visit` sein
 - `license`: Lizenz, unter dem die Mediendatei in dieser Kompressions-Stufe verfügbar ist
 - `fileSize`: Größe der Mediendatei in Bytes (bei 3D-Modellen mit mehreren Dateien die Summe aller Dateien)
 - `paths`: Array mit den internen Dateinamen aller zu dieser medialen Präsentation gehöriger Dateien
 - `fileTypeSpecificMeta`: Weitere unspezifizierte Informationen zu dieser Kompressions-Stufe

BILDDATEI Zur Kompression von Bilddateien wird im Hintergrund die Software ImageMagick verwendet. Für jede der in der Konfiguration festgelegten Auflösungsstufen, die kleiner als die Originalgröße des Bildes ist, wird damit eine verkleinerte Version des Bildes erstellt. Laut den technischen Metadaten bereits existierende Auflösungsstufen werden dabei übersprungen. Die neu erstellten Auflösungsstufen werden nun den technischen Metadaten hinzugefügt und in die Meta-Datenbank überführt.

3D-MODELL Ein texturiertes 3D-Modell besteht in der Regel aus drei Dateien. Die OBJ-Datei mit den Geometriedaten verweist auf eine MTL-Datei, welche die Materialeigenschaften der Oberfläche beschreibt und insbesondere vorhandene Textur-Dateien referenziert. Beim Hochladen eines 3D-Modells in die Medien-Datenbank werden die Dateien dort automatisch umbenannt. Die Referenzen, die zwischen den Dateien bestehen, werden dabei jedoch nicht angepasst und daher ungültig. Im ersten Schritt der Kompression werden diese Verweise in den Originaldateien repariert, indem sie an die neuen Dateinamen angepasst werden. Auch die zum unkomprimierten Modell gehörenden technischen Metadaten werden angepasst.

Nachdem das unkomprimierte Modell repariert wurde, kann die mit der eigentlichen Kompression begonnen werden. Zunächst werden die Geometriedaten (OBJ-Dateien) und ggf. auch die Material-Datei komprimiert und abgespeichert, wobei auch hier laut den technischen Metadaten bereits existierende Auflösungsstufen übersprungen werden. Nachdem alle Varianten der Geometriedaten erstellt wurden, erfolgt bei texturierten Modellen die Kompression der Textur separat für jede Auflösungsstufe, erneut unter der Zuhilfenahme der Software ImageMagick. Die zuvor erstellten Material-Dateien verweisen bereits auf die nun erstellten Bilddateien. Wie bereits bei den Bilddateien werden abschließend die um die neuen Auflösungsstufen erweiterten technischen Metadaten an die Meta-Datenbank übertragen.

9.5 Installation und Steuerung

9.5.1 Installation

Zur Installation des Kompressions-Systems muss zunächst über den Befehl

```
1 docker build -t "visitapp/compression" https://github.com/VisIT-Dev/compressioncontainer.git
```

Listing 24: Befehl zum Erstellen des Docker-Images

ein Docker-Image erstellt werden. Nun muss ein Verzeichnis gewählt werden, über das von außen auf die das Kompressions-System betreffende Dateien, wie die Log-Dateien oder die Konfiguration zugegriffen wird. Hier werde beispielsweise das Verzeichnis `/etc/visit-compression` gewählt. Anschließend kann über den Befehl

```
1 docker run -d --name compression -p 1613:1613 -v visit-p2p-private:/var/www/Private -v /etc/visit-compression:/root/compression visitapp/compression
```

Listing 25: Befehl zum Erstellen des Docker-Containers

aus dem soeben erstellten Docker-Image ein ausführbarer Docker-Container generiert werden, sofern `visit-p2p-private` das Docker-Volume bezeichnet, in welchem die Medien-Dateien gespeichert sind. Die einzelnen Bestandteile dieses Befehls werden im folgenden kurz erläutert:

- `docker run`: Dies ist der Befehl zum Erstellen eines Docker-Images.
- `-d`: Der Container soll im Hintergrund ausgeführt werden.
- `-p 1613:1613`: Der Port `1613` innerhalb des Docker-Containers soll zum Host-System durchgeschleift werden. Sollen andere Ports verwendet werden, muss dieser Bestandteil entsprechend angepasst werden.
- `-v visit-p2p-private:/var/www/Private`: Das beim Einrichten der Medien-Datenbank erzeugte Docker-Volume `visit-p2p-private`, in dem die Medien-Dateien gespeichert sind, soll innerhalb des Docker-Containers unter dem Pfad `/var/www/Private` verfügbar sein.
- `-v /etc/visit-compression:/root/compression`: Auf dem Host-System soll unter dem Container-Verzeichnis `/etc/visit-compression` auf die Dateien zugegriffen werden können, welche sich im Docker-Container im Verzeichnis `/root/compression` befinden.

- `visitapp/compression`: Dies ist die gewählte Bezeichnung für das eben erstellte Docker-Image.

Nach dem Erstellen des Docker-Containers wird dieser automatisch gestartet und kann, wie im folgenden Abschnitt erläutert wird, beendet oder neu gestartet werden. Bei gestartetem Container ist die Web-Oberfläche in der Standardkonfiguration unter `http://localhost:1613` erreichbar, während die API unter `http://localhost:1613/api` zur Verfügung steht.

9.5.2 Steuerung des Docker-Containers

Der die Kompressions-Komponente beinhaltende Docker-Container kann wie jeder andere Container mit den Befehlen

```
1 docker stop compression
```

Listing 26: Kommando zum Beenden des Docker-Containers der Kompressions-Komponente

] gestoppt und mit dem Befehl

```
1 docker start compression
```

Listing 27: Kommando zum Starten des Docker-Containers der Kompressions-Komponente

wieder gestartet werden. Alle Einstellungen bleiben dabei erhalten. Über das bei der Installation angegebene Verzeichnis (standardmäßig `/etc/visit-compression`) kann auch außerhalb des Containers auf das Kompressions-System betreffende Dateien zugegriffen werden. So kann durch Editieren der Datei `config.ini` die Konfiguration angepasst werden, wobei für ein Aktivieren der aktualisierten Einstellungen ein Neustart des Docker-Containers erforderlich ist. Außerdem kann auf die Log-Datei `compression.log` zugegriffen werden, die insbesondere bei fehlgeschlagenen Kompressions-Aufträgen oder bei unerwartetem Beenden des Kompressions-Systems wichtige Informationen über die zugrunde liegende Ursache liefern kann. Da die Ausgabe dieser Informationen zusätzlich über die Standardausgabe erfolgt, besteht eine weitere Möglichkeit des Zugriffs im Ausführen des Kommandos

```
1 docker logs compression
```

Listing 28: Kommando zum Anzeigen der Ausgaben des Kompressions-Systems

Für fortgeschrittene Benutzer, welche Anpassungen direkt im Docker-Container vornehmen möchten, kann über den Befehl

```
1 docker exec -it compression /bin/bash
```

Listing 29: Befehl zum Öffnen einer Kommandozeile im Kompressions-Container

eine Kommandozeile geöffnet werden.

9.5.3 Steuerung der Kompressions-Komponente

Sämtliche Kommunikation mit der Kompressions-Komponente selbst erfolgt über die bereitgestellte API. Die zur Verfügung stehende Web-Oberfläche greift ebenfalls über diese API auf das System zu. Für Details zur Web-Oberfläche und zur API wird auf die Abschnitte [9.7](#) und [9.8](#) verwiesen.

Die Kompressions-Komponente startet automatisch nach dem Start des entsprechenden Docker-Containers. Je nach Konfiguration wird daraufhin unmittelbar mit dem Abarbeiten von Kompressions-Aufträgen begonnen.

Die Kompressions-Komponente lässt sich über entsprechende API-Aufrufe in unterschiedlichen Modi herunterfahren. Diese Modi umfassen

- das Abarbeiten aller noch ausstehender Aufträge vor dem Beenden,
- das Abarbeiten nur des sich aktuell in Verarbeitung befindlichen Auftrags und

- das sofortige Beenden des Systems ohne Rücksichtnahme auf die ausstehenden Kompressions-Aufträge.

In jedem Fall werden keine weiteren Kompressions-Aufträge angenommen. Ausstehende Kompressionsaufträge bleiben beim erneuten Starten der Komponente erhalten. Jedoch wird ein etwaiger Auftrag, der sich während des Beendens in der Verarbeitung befand, als „Fehlerhaft abgeschlossen“ markiert.

Diese Methode des Herunterfahrens erlaubt im Gegensatz zum Stoppen des Docker-Containers die kontrollierte Handhabung noch ausstehender oder sich in Verarbeitung befindlicher Kompressions-Aufträge. Allerdings wird durch das Herunterfahren der Kompressions-Komponente der sie enthaltende Docker-Container nicht automatisch mit beendet. Dies muss separat durch das Kommando in [Listing 26](#) erfolgen. Wird der Container mit diesem Kommando ohne vorheriges Herunterfahren des Kompressions-Systems ausgeführt, entspricht dies dem sofortigen Beenden des Systems ohne Rücksichtnahme auf die ausstehenden Kompressions-Aufträge.

9.6 Konfiguration

Beim Starten des Kompressions-Systems wird automatisch eine Konfigurationsdatei mit den Standardwerten unter dem Pfad

```
1 /root/compression/config.ini
```

Listing 30: Example XML

angelegt, sofern eine solche nicht bereits an dieser Position existiert. Einige der Werte lassen sich nur manuell über diese Datei anpassen. Auf die wichtigsten Konfigurationsmöglichkeiten kann jedoch auch von außen über die API und die Web-Oberfläche sowohl lesend als auch schreibend zugegriffen werden. Beim manuellen Anpassen der Konfigurationsdatei muss darauf geachtet werden, dass innerhalb der Variablenwerte die Zeichen „：“, „=” und „\“ mit einem vorgestellten Backslash versehen werden müssen, also beispielsweise durch „https\://“. Für Details diesbezüglich wird auf die entsprechende Java-Dokumentation⁹ verwiesen. [Tabelle 2](#) gibt Aufschluss über alle Konfigurationsmöglichkeiten, welche in den folgenden Abschnitten detailliert erläutert werden.

9.6.1 Verwaltung von Kompressionsaufträgen

Zur Verwaltung der eingehenden Kompressionsaufträge und für die dazu notwendigen Einstellungen des Servers stehen die folgenden Konfigurationsmöglichkeiten zur Verfügung:

ZUGRIFFSBESCHRÄNKUNG Der Wert von `accessWhiteListIps` beschreibt, über welche Rechner auf die API oder die Web-Oberfläche zugegriffen werden kann, indem die IP-Adressen dieser Rechner angegeben werden können. Generell sollte die Absicherung allerdings auf eine andere Art von außen vorgenommen werden. Die IP-Adressen werden in eckigen Klammern und durch Kommata getrennt notiert. Befindet sich ein Asterisk („*“) in dieser Auflistung, wird der Zugriff für alle Rechner autorisiert. Der Standardwert für diese Eigenschaft ist `[127.0.0.1, *]`, wodurch keine Beschränkung des Zugriffs erfolgt.

SERVERPORT Der Wert für die Variable `apiPort` muss einer Ganzzahl im Intervall von 1 bis 65535 entsprechen und legt den Port fest, über welchen sowohl auf die API als auch auf die Web-Oberfläche der Kompressions-Komponente zugegriffen werden kann. Der Standardwert für diese Variable ist 1613.

⁹ [https://docs.oracle.com/javase/7/docs/api/java/util/Properties.html#load\(java.io.Reader\)](https://docs.oracle.com/javase/7/docs/api/java/util/Properties.html#load(java.io.Reader))

Variable	Standard	Web
Verwaltung von Kompressionsaufträgen		
accessWhiteListIps	[127.0.0.1,*]	Ja
apiPort	1613	Ja
archiveDisplayLength	250	Nein
autostart	true	Ja
mediaFileRootDirectory	/var/www/private	Nein
queueMaxLength	5000	Ja
3D-Kompression		
defaultLevels	siehe Beschreibung	Ja
targetSizeBoundaryPenalty	100.0	Nein
targetSizeNormalDifferencePenalization	1000.0	Nein
targetSizeNormalDifferenceThreshold	0.5	Nein
targetSizePartitionPenalization	10.0	Nein
targetSizeQualityThreshold	0.3	Nein
textureLimits	[5000, 50000]	Ja
textureSizes	[1024, 2048, 8192]	Ja
Bildkompression		
imageCompressionLevels	siehe Beschreibung	Ja
Schnittstelle Meta-Datenbank		
metadbApiAuthString	Basic XX...XX\=\ =	Nein
metadbApiEndpointFetchUrl	https\://DOMAIN/metadb-rest-api/digrep/media	Nein
metadbApiEndpointSendUrl	https\://DOMAIN/metadb-rest-api/digrep/media	Nein
metadbApiMediaUidPrefix	http\://DOMAIN/metadb/	Nein

Tabelle 2: Überblick über alle Konfigurationsmöglichkeiten der Kompressionskomponente

LÄNGE DER ARCHIV-ANZEIGE Über den Parameter `archiveDisplayLength` lässt sich festlegen, wie viele Einträge in dem über die Web-Oberfläche zugänglichen Archiv der letzten Kompressions-Aufträge angezeigt werden sollen. Auch dieser Wert muss einer positiven Ganzzahl entsprechen und beträgt standardmäßig 250.

AUTOMATISCHER START Der Wert der Eigenschaft `autostart` kann entweder `true` oder `false` entsprechen und legt fest, ob direkt nach dem Starten der Kompressions-Komponente mit der Abarbeitung eingehender Kompressions-Aufträge begonnen werden soll, wobei dieser Fall dem Standard entspricht.

HAUPTVERZEICHNIS FÜR MEDIEN-DATEIEN Die Konfigurationsoption `mediaFileRootDirectory` legt fest, in welchem Verzeichnis innerhalb des Docker-Containers der Kompressions-Komponente die zu komprimierenden Mediendateien gespeichert sind. Etwaige Unterverzeichnisse können für jeden Kompressions-Auftrag separat angegeben werden. In ebendiesem Verzeichnis werden die komprimierten Versionen der Dateien nach der Aufführung des Auftrags auch abgelegt. Der Standardort für diese Dateien ist `/var/www/private`.

MAXIMALE LÄNGE DER AUFRAGSLISTE Mithilfe der Option `queueMaxLength` lässt sich eine Beschränkung für die Länge der Liste der unbearbeiteten Kompressions-Aufträge festlegen. Sollte dieser Wert erreicht sein, werden etwaige eingehenden Aufträge abgewiesen. Ist der Wert auf 0 festgelegt, so erfolgt keine Beschränkung der Liste. Der Standardwert beträgt 5000.

9.6.2 3D-Kompression

Folgende Optionen stehen zur Konfiguration des Kompressions-Algorithmus für 3D-Modelle zur Verfügung:

STANDARD-KOMPRESSSIONSSTUFEN Der Wert für die Option `defaultLevels` legt fest, welche Auflösungsstufen für 3D-Modelle standardmäßig erzeugt werden sollen. Jede Auflösungsstufe wird dabei durch eine Ganzzahl beschrieben, welche die gewünschte Anzahl an Vertices des komprimierten Modells festlegt. Diese Stufen werden durch Kommata voneinander getrennt und insgesamt von eckigen Klammern umrahmt, wie auch durch den in Listing 31 aufgeführten Standardwert

```
[500, 1000, 5000, 20000, 50000, 200000, 500000, 2000000, 5000000, 20000000, 50000000]
```

Listing 31: Standardwert für die Konfigurationsoption `defaultLevels`

deutlich wird. Hat das ursprüngliche Modell bereits weniger Vertices als die angestrebte Anzahl einer Auflösungsstufe, so wird diese Stufe übersprungen anstatt ein Modell mit einer größeren Anzahl an Vertices zu erzeugen.

BESTRAFUNGEN VON ABWEICHUNGEN AM RAND Der Gleitkommawert für die Eigenschaft `targetSizeBoundaryPenalty` ist nur für die Kompression von Meshes mit Rand, die also kein vollständiges Modell eines realen Objekts darstellen, relevant. Er legt fest, mit welchem Gewicht dieser Rand des ursprünglichen Modells beibehalten werden soll. Bei einem hohen Wert dieser Eigenschaft werden durch die Kompression kaum Änderungen an diesen Rändern vorgenommen, während bei einem niedrigen Wert viele Edge-Collapse-Operationen genau dort ausgeführt werden. Der Standardwert beträgt `100.0`.

BESTRAFUNGEN BEI ABWEICHUNGEN DER OBERFLÄCHENNORMALE Die Oberflächennormale ist einen Richtung, welche die Ausrichtung eines Face beschreibt und somit senkrecht zur Ebene, welche durch das Face erzeugt wird, verläuft. Über die beiden Eigenschaften `targetSizeNormalDifferencePenalization` und `targetSizeNormalDifferenceThreshold` lässt sich festlegen, in welchem Ausmaß starke Veränderungen dieser Normalen vermieden werden sollen. Der Wert von `targetSizeNormalDifferenceThreshold` beschreibt dabei, ab welcher Abweichung der durch eine Edge-Collapse-Operation verursachten Veränderung von Oberflächennormalen eine Bestrafung erfolgt, welche die Ausführung dieser Operation unwahrscheinlicher macht, indem die Kosten der Operation mit dem Faktor `targetSizeNormalDifferencePenalization` multipliziert werden. Perfekt übereinstimmende Normalen entsprechen dabei dem Wert `1.0`, während zueinander senkrechte Normalen durch den Wert `0.0` beschrieben werden. Entsteht durch eine Edge-Collapse-Operation ein Face, dessen Oberflächennormale eine Übereinstimmung mit der ursprünglichen Normale hat, die unter dem Wert von `targetSizeNormalDifferenceThreshold` liegt, so wird dieser Bestrafungsfaktor aktiviert.

BESTRAFUNGEN ENTLANG VON TEXTURE SEAMS Wie in Abschnitt 9.2.3 erläutert wurde, sorgen Edge-Collapse-Operationen entlang von Texture Seams nicht nur zu Verzerrungen in der Textur, sondern können auch die Darstellung von eigentlich nicht parametrisierten Bereichen in der Texturdatei zur Folge haben, was es möglichst zu vermeiden gilt. Aus diesem Grund werden Kompressionsoperationen entlang von Texture Seams mit einem Faktor bestraft, der sich im Wesentlichen aus dem Produkt des Quadrats der an die kollabierende Kante angrenzenden Texturpartitionen und des Werts der Eigenschaft `targetSizePartitionPenalization` ergibt, wobei letzterer standardmäßig auf `10.0` festgelegt ist.

SCHWELLWERT FÜR DIE BEGÜNSTIGUNG WOHLGEFORMTER FACES Bei der Erzeugung von Meshes werden in der Regel möglichst gleichmäßige Faces angestrebt, während hingegen sehr lange aber dünne Dreiecke unerwünscht sind. Als Maß für diese „Schönheit“ eines Faces wird der Quotient aus Fläche und maximaler Seitenlänge betrachtet. Die Kosten einer Edge-Collapse-Operation werden durch die minimale Qualität der durch diese Operation entstehenden Faces dividiert, wodurch allerdings bei sehr wohlgeformten Faces und demzufolge hoher Qualität die Ko-

Vertexanzahl	Texturauflösung
1000	1024 x 1024
5000	2048 x 2048
10000	2048 x 2048
75000	8192 x 8192

Tabelle 3: Beispiele für die sich bei unterschiedlichen Vertexanzahlen ergebenden Texturauflösungen bei der Standardkonfiguration.

sten der gesamten Operation unerwünscht stark sinken können. Aus diesem Grund lässt sich durch den Parameter `targetSizeQualityThreshold` der Divisor nach oben beschränken, wobei der Standardwert 0.3 beträgt und somit auch nicht perfekt geformte Dreiecke ohne Erhöhung der Kosten zulässt.

TEXTURKOMPRESSION Durch die Textur eines 3D-Modells kann ein relevanter Anteil des insgesamt notwendigen Speicherbedarfs verursacht werden, weshalb es auch diesen Bestandteil zu komprimieren gilt. Dies sollte je nach gewählter Vertexanzahl in einem ähnlichen Ausmaß geschehen. Allerdings können viele Programme nur Texturdateien mit einer Zweierpotenz als Seitenlänge verarbeiten oder erweitern die gegebene Textur durch Padding zu einer solchen Größe. Aus diesem Grund ist eine pixelgenaue Wahl der Texturauflösung nur bedingt sinnvoll. Stattdessen kann einem bestimmten Intervall an Vertexanzahlen eine bestimmte Texturauflösung zugewiesen werden. Dies lässt sich über die beiden Parameter `textureLimits` und `textureSizes` konfigurieren, wobei beide Parameter Listen mit ganzzahligen Einträgen als Werte akzeptieren. Die Einträge in diesen Listen werden wie bereits bei anderen Konfigurationsoptionen durch Kommata getrennt, während die ganze Liste durch eckige Klammern eingefasst wird. Zu beachten ist jedoch, dass die Anzahl an Einträgen in `textureSizes` stets um genau eins größer sein muss, als die Anzahl der Einträge in `textureLimits`.

Hat ein Modell nun weniger Vertices, als der erste Eintrag in der Schwellwertliste `textureLimits`, so wird eine Textur erzeugt, deren Auflösung dem ersten Eintrag in `textureSizes` entspricht. Hat es stattdessen mindestens so viele Vertices wie der erste Eintrag in `textureLimits`, jedoch weniger Vertices als der zweite Eintrag in `textureLimits` vorgibt, so wird eine Textur mit einer Seitenlänge entsprechend des zweiten Eintrags in `textureSizes` erzeugt. Diese Regel gilt entsprechend für jeden Eintrag in `textureLimits`. Standardmäßig sind die Schwellwerte `textureLimits` festgelegt durch [5000, 50000], während die dazugehörigen Auflösungen durch [1024, 2048, 8192] gegeben sind. Bei Bedarf lässt sich diese Konfiguration feiner gestalten und dabei insbesondere auch ein Intervall festlegen, das Texturdateien mit einer Seitenlänge von 4096 Pixeln erzeugt. [Tabelle 3](#) veranschaulicht die resultierenden Texturauflösungen abhängig von unterschiedlichen Vertexanzahlen für die Standardkonfiguration.

Ist die Auflösung der ursprünglichen Texturdatei jedoch kleiner als die sich bei der Kompression ergebende Größe, so wird die Textur nicht vergrößert, sondern es wird die Datei in der ursprünglichen Auflösung unverändert übernommen.

9.6.3 Bildkompression

Ähnlich wie bei den 3D-Modellen sollen auch bei Bildern unterschiedliche Auflösungen vorgehalten werden, um für verschiedene Anwendungsfälle eine passende Größe zur Verfügung zu haben. Welche Auflösungen bei der Kompression einer Bilddatei erstellt werden sollen, wird durch die Konfigurationsoption `imageCompressionLevels` festgelegt. Aufgrund der Komplexität dieses Parameters muss dieser im JSON-Format¹⁰ angegeben werden. Der Wert der Konfigurationsoption

¹⁰ <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>

Name	Wertebereich	Beispiel
maxWidth	Positive Ganzzahl	1920
maxHeight	Positive Ganzzahl	1080
title	A bis Z, a bis z, Binde-, Unterstriche	„FullHD“

Tabelle 4: Pro Kompressionsstufe notwendige Parameter bei der Bildkompression

muss einem Array aus Objekten entsprechen, wobei jedes Objekt die in Tabelle 4 dargestellten Eigenschaften aufzuweisen hat. Jeder Eintrag des Arrays definiert auf diese Art eine Kompressions-Stufe für ein Bild. Ist das Bild in mindestens einer Dimension kleiner als eine bestimmte Kompressionsstufe, so wird die Erzeugung einer Version mit dieser Auflösung komplett übersprungen, es wird also im Gegensatz zur Texturkompression nicht die ursprüngliche Auflösung verwendet. Der Standardwert für diesen Parameter ist in Listing 32 dargestellt.

```
1  [{"maxWidth":3840,"maxHeight":2160,"title":"UHD"}, {"maxWidth":1920,"maxHeight":1080,"title":"FullHD"}, {"maxWidth":800,"maxHeight":600,"title":"Mittel"}, {"maxWidth":120,"maxHeight":120,"title":"Klein"}]
```

Listing 32: Standardwert für die Konfigurationsoption `imageCompressionLevels`

9.6.4 Schnittstelle Meta-Datenbank

Um die während des Kompressionsvorgangs erstellen Modelle im ViSIT-System zu registrieren, müssen die zu der Mediendatei gehörigen technischen Metadaten, die in Listing 22 spezifiziert wurden, in der Meta-Datenbank aktualisiert werden. Hierzu muss auf diese Datenbank zugegriffen werden können, wofür die nachfolgend erläuterten Parameter anzugeben sind. Alle in diesem Abschnitt angegebenen Konfigurations-Optionen müssen nach der Installation angepasst werden, um die Integration in das Gesamtsystem zu ermöglichen. In sämtlichen Standardwerten ist `DOMAIN` durch die jeweilige Domain, über welche auf die Meta-Datenbank zugegriffen werden kann, zu ersetzen.

AUTHORIZIERUNG ZUM ZUGRIFF AUF DIE METADATEN Um die Meta-Datenbank durch unbefugten Zugriff zu schützen, müssen sich zugelassene Benutzer oder Systeme authentifizieren. Diese Authentifizierung erfolgt durch das *HTTP Basic Authentication*-Verfahren¹¹. Der dafür notwendige Base64-codierte Authentifizierungs-String, dem die Zeichenfolge `Basic` vorausgeht, muss in der Konfigurationsoption `metadbApiAuthString` angegeben werden, welche nach der Installation der Kompressions-Komponente auf einen gültigen Wert zu setzen ist. Der (ungültige) Standardwert ist in Listing 33 dargestellt.

```
1  Basic XXXXXXXXXXXXXXXXXXXXXXXX\=\\=
```

Listing 33: Standardwert für die Konfigurationsoption `metadbApiAuthString`

ENDPUNKT ZUM ABRUFEN DER METADATEN In der Konfigurationsoption `metadbApiEndpointFetchUrl` kann der Endpunkt der API zur Meta-Datenbank spezifiziert werden, über den technische Metadaten abgerufen werden können. Dieser Wert ist standardmäßig festgelegt auf `https://DOMAIN/metadb-rest-api/digrep/media`.

ENDPUNKT ZUM SCHREIBEN DER METADATEN Mithilfe der Konfigurationsoption `metadbApiEndpointSendUrl` kann der Endpunkt der API zur Meta-Datenbank spezifiziert werden, über den technische Metadaten gespeichert werden können. Auch hier wird der Wert `https://DOMAIN/metadb-rest-api/digrep/media` als Standard verwendet, da er in der Regel mit dem Wert für `metadbApiEndpointFetchUrl` übereinstimmt.

¹¹ <https://tools.ietf.org/html/rfc2617>

PRÄFIX DER MEDIEN-UIDS Jede in der Meta-Datenbank registrierte mediale Repräsentation wird durch eine eindeutige sogenannte UID identifiziert. Jede UID beginnt mit einem allen Mediendateien gemeinsamen Präfix, auf welches ein für das Objekt spezifischer alphanumerischer Identifikator folgt. Da zum Starten eines Kompressionsvorgangs durch die Medien-Datenbank nur das alphanumerische Suffix übermittelt wird, muss in der Konfigurationsoption `metadbApiMediaUidPrefix` das konstante Präfix festgelegt werden. Der Standardwert, der gegeben ist durch `http://DOMAIN/metadb/`, muss vor dem ersten Start der Kompressionskomponente geeignet angepasst werden.

9.7 Zugriff über die Web-Oberfläche

Auf die Web-Oberfläche kann über jeden Browser zugegriffen werden, indem eine Verbindung mit dem Docker-Container auf dem in der Konfiguration festgelegten Port aufgebaut wird. Auf dem Server kann beispielsweise in der Standardkonfiguration, und sofern der Port durch die Docker-Konfiguration nicht umgeleitet wird, durch die Eingabe der in [Listing 34](#) dargestellten Zeichenfolge die Startseite der Kompressions-Komponente aufgerufen werden.

```
1 http://localhost:1613
```

[Listing 34](#): Zugriff auf die Web-Oberfläche der Kompressions-Komponente

Die Web-Oberfläche bietet vier verschiedene Ansichten, welche im Folgenden erläutert werden. Zwischen diesen Ansichten kann über die Navigation in der linken Seitenleiste bzw. über das Aufklapp-Menü auf der linken Seite gewechselt werden.

9.7.1 Startseite

Die Startseite hat zwei wichtige Funktionen. Zum einen erlaubt sie das Pausieren oder Fortführen der Auftragsverarbeitung, zum anderen gibt sie einen Überblick über die aktuell ausstehenden Kompressions-Aufträge, die sich dort auch abbrechen lassen. Wie in [Abbildung 46](#) deutlich wird, werden zu jedem ausstehenden Auftrag mehrere Informationen angezeigt. Diese umfassen neben dem aktuellen Status des Auftrags der Titel, UID und Dateityp des Objekts, die gewünschten Kompressionsstufen, sowie die Zeitpunkte der Absetzung des Auftrags und der letzten Statusänderung. Über die rote Schaltfläche im rechten oberen Bereich eines Auftrags wird dieser nach dem Bestätigen einer Sicherheitsabfrage abgebrochen, sofern sich dieser noch nicht in Verarbeitung befindet. Kompressionsaufträge, die bereits ausgeführt werden, lassen sich nicht abbrechen. Die Auflistung der ausstehenden Kompressionsaufträge erfolgt aufsteigend nach dem Zeitpunkt der Absetzung.

9.7.2 Neuer Auftrag

Über diese Seite, welche in [Abbildung 47](#) dargestellt ist, lassen sich manuelle Kompressionsaufträge absetzen, wobei dies normalerweise direkt beim Hochladen der Medien-Datei von der Medien-Datenbank übernommen wird. Soll dennoch manuell ein Auftrag abgesetzt werden, so müssen in dieser Ansicht die UID des Metadatums (Objekt-Identifikator, `objectId`) und die UID der digitalen Repräsentation (Medien-Identifikator, `mediaID`), deren Beziehung in Abbildung [Abbildung 58](#) veranschaulicht wird, angegeben werden. Zusätzlich kann ein Unterverzeichnis angegeben werden (Basis-Pfad), in dem sich die zu komprimierende Datei befindet, wobei dieser Pfad stets in Bezug auf das in der Konfiguration angegebene `mediaFileRootDirectory` interpretiert wird. Außerdem muss ein Titel für die Datei angegeben werden, der beliebig gewählt werden kann und lediglich zur leichteren Identifizierung des Auftrags innerhalb des Kompressions-Systems dient. Des Weiteren ist der Dateityp zu spezifizieren, wobei hier die Optionen PNG-Bild, JPEG-Bild und OBJ-3D-Modell zur Auswahl stehen.

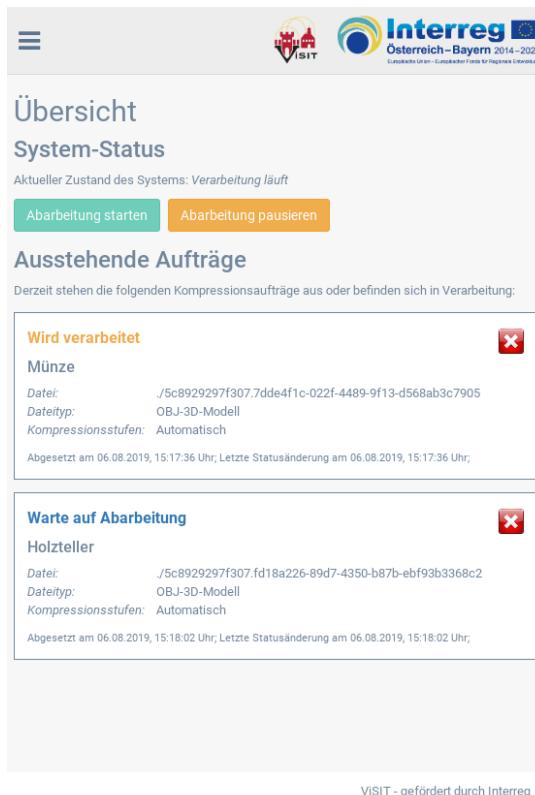


Abbildung 46: Startseite, unter anderem mit einem Überblick über die laufenden und anstehenden Kompressions-Aufträge

Für den Fall, dass ein 3D-Modell komprimiert wird, können im darauffolgenden Abschnitt die gewünschten Kompressionsstufen festgelegt werden. Zum Einen steht die Option „Automatisch“ zur Verfügung, die standardmäßig ausgewählt ist und alle in der Konfiguration spezifizierten Standard-Kompressionsstufen umfasst. Weitere oder alternative Auflösungsstufen können durch die Wahl des Eintrags „Feste Größe“ und die Angabe der gewünschten Vertex-Anzahl hinzugefügt werden. Sämtliche derzeit angegebenen Auflösungsstufen werden aufgelistet und lassen sich mit Klick auf das Mülltonnen-Symbol wieder entfernen.

9.7.3 Archiv

Über diese Ansicht lässt sich ein Überblick über die zuletzt abgearbeiteten Kompressions-Aufträge erhalten, unabhängig davon, ob die Ausführung erfolgreich war oder fehlgeschlagen ist. Die Einträge werden nach der letzten Statusänderung absteigend sortiert, wobei die Anzahl der dargestellten Einträge in der Konfiguration festgelegt werden kann. Ein Beispiel für diese Ansicht ist in Abbildung 48 zu sehen.

9.7.4 Einstellungen

Auf dieser Seite lassen sich die wichtigsten Konfigurations-Optionen anpassen, wie in Abbildung 49 zu sehen ist. Alle Einstellungen, die auf dieser Seite nicht verfügbar sind, müssen manuell in der Konfigurationsdatei angepasst werden. Für Details zu den einzelnen Optionen wird auf Abschnitt 9.6 verwiesen.

Die über die Web-Oberfläche verfügbaren Konfigurationsmöglichkeiten werden im Folgenden aufgeführt, wobei auch die Entsprechung in der Konfigurationsdatei benannt wird. Sämtliche Angaben werden jedoch erst durch das Betätigen der Schaltfläche „Einstellungen speichern“ am Ende der Seite zum Server übertragen und übernommen.

Basisdaten

- Basis-Pfad: [leeres Feld]
- Objekt-Identifikator (UID): 5c8929297f307
- Medien-Identifikator (UID): 7dde4f1c-022f-4489-9f13-d568ab3c7905
- Datei-Titel: Münze
- Dateityp: OBJ-3D-Modell

Kompressionsstufen

- Automatisch
- 12345

Weitere Kompressionsstufe:

- Feste Größe

Anzahl der Vertices:

- 12345

Auftrag absetzen

Abbildung 47: Ansicht zum Absetzen eines neuen Kompressions-Auftrags

Erfolgreich abgeschlossen
Münze
Datei: /5c8929297f307.7dde4f1c-022f-4489-9f13-d568ab3c7905
Dateityp: OBJ-3D-Modell
Kompressionsstufen: Automatisch
Abgesetzt am 06.08.2019, 15:17:36 Uhr; Letzte Statusänderung am 06.08.2019, 15:19:27 Uhr;

Fehlerhaft abgeschlossen
Holzteller
Datei: /5c8929297f307.fd18a226-89d7-4350-b87b-ebf93b3368c2
Dateityp: OBJ-3D-Modell
Kompressionsstufen: Automatisch
Abgesetzt am 06.08.2019, 15:18:02 Uhr; Letzte Statusänderung am 06.08.2019, 15:19:27 Uhr;

Erfolgreich abgeschlossen

Abbildung 48: Archiv-Ansicht mit einem Überblick über die ausgeführten Kompressions-Aufträge

- *Portnummer der API-Schnittstelle:* `apiPort`
- *Maximale Länge der Auftragsliste:* `queueMaxLength`
- *Abarbeitung automatisch beim Starten des Servers beginnen:* `autostart`
- *API-Zugriffsberechtigte IP-Adressen:* `accessWhiteListIps`. Die berechtigten IP-Adressen sind einzeln anzugeben und über die Schaltfläche „IP-Adresse hinzufügen“ zu bestätigen. Durch einen Klick auf das Mülltonnen-Symbol können Einträge wieder entfernt werden. Auch hier entspricht die Angabe eines Asterisks („*“) einer Zugriffsgenehmigung für alle Hosts.
- *Standard-Kompressionsstufen für 3D-Modelle:* `defaultLevels`. Die Anzahl der Vertices der komprimierten Modelle, die bei der Kompression von 3D-Modellen standardmäßig erstellt werden sollen, sind an dieser Stelle einzeln anzugeben und über die Schaltfläche „Kompressionsstufe hinzufügen“ hinzuzufügen. Auch hier können einzelne Kompressionsstufen durch Betätigen des Mülleimer-Symbols entfernt werden.
- *Kompression der Textur von 3D-Modellen:* `textureLimits`, `textureSizes`. Die Anzahl der Schwellwerte und demzufolge unterschiedlicher Textur-Auflösungen lässt sich über die beiden Schaltflächen „Unterscheidung hinzufügen“ bzw. „Unterscheidung entfernen“ kontrollieren. Die Texturgröße ist als Anzahl der Pixel pro Dimension zu verstehen.
- *Aktionen für die Kompression von Bildern:* `imageCompressionLevels`. Die für eingehende Bild-Kompressions-Aufträge zu erstellenden Auflösungsstufen sind hier aufgelistet, wobei sich einzelne Einträge durch das Betätigen des Mülleimer-Symbols entfernen lassen. Weitere Kompressionsstufen lassen sich durch die Angabe eines beliebigen Titels, der nur aus Groß- oder Kleinbuchstaben, Ziffern und Binde- oder Unterstrichen bestehen darf, sowie der maximalen Breite und maximalen Höhe des komprimierten Bildes in Pixeln und anschließendes Betätigen der Schaltfläche „Kompressionsstufe hinzufügen“ hinzufügen.

9.8 Zugriff über die API

Sämtliche Funktionen der API stehen bei der standardmäßigen Konfiguration unter dem Basispfad `http://localhost:1613/api` zur Verfügung. Die API bietet unterschiedliche Module an, auf deren Funktionen in diesem Abschnitt eingegangen wird. POST-Parameter sind stets als JSON-Objekt übergeben, ebenso erfolgt die Antwort in Form eines JSON-Objekts.

9.8.1 Aktuelle Kompressions-Aufträge

Für Informationen zu den IDs, dem MIME-Typen oder den Kompressions-Stufen wird auf Abschnitt 9.4 verwiesen. Beim Absetzen wird jedem Kompressions-Auftrag eine ID zugewiesen, welche zum Löschen des Auftrags angegeben werden muss. Der Status eines Auftrags kann einen der folgenden Werte annehmen:

- **ENQUEUED:** Der Auftrag wurde abgesetzt, mit der Abarbeitung wurde jedoch noch nicht begonnen.
- **PROCESSING:** Der Auftrag wurde abgesetzt und befindet sich derzeit in Verarbeitung.
- **ERROR:** Die Verarbeitung des Auftrags wurde fehlerhaft abgeschlossen.
- **COMPLETED:** Die Verarbeitung des Auftrags wurde erfolgreich abgeschlossen.

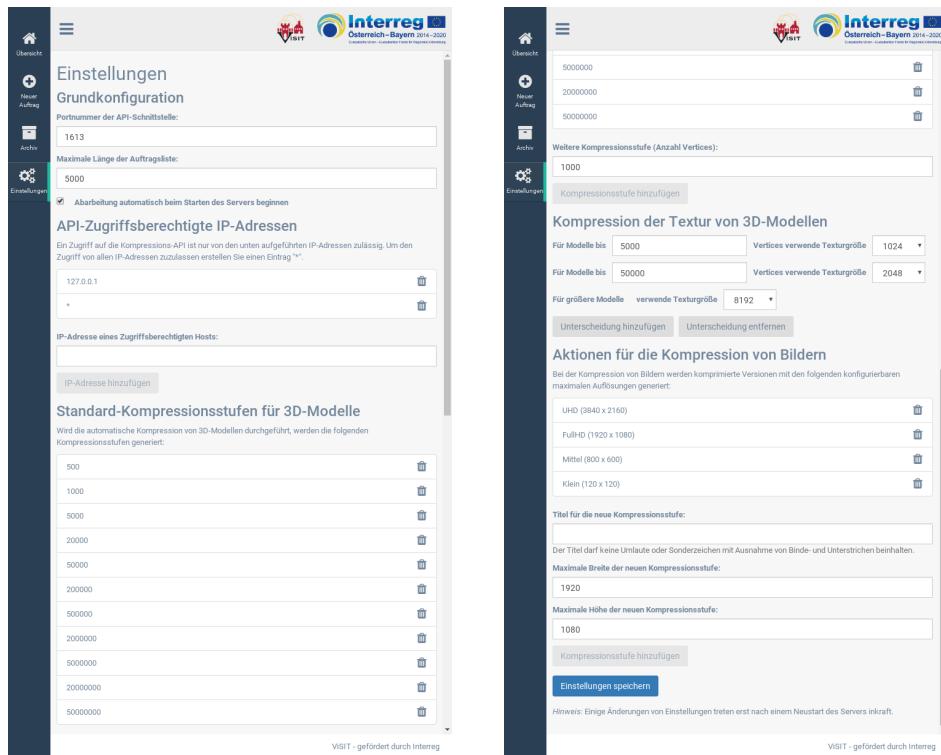


Abbildung 49: Einstellungsmöglichkeiten über die Web-Oberfläche

Funktion: Absetzen eines Kompressions-Auftrags

HTTP-Methode: POST

Pfad: /jobs/dispatch

POST-Paramter:

```

1 {
2   basePath : string, /* Unterverzeichnis, in welchem die Datei abgelegt ist, ansonsten "" */
3   objectUid : string, /* objectTripleID (nicht objectTripleURL) der Medien-Datei */
4   mediaUid : string, /* mediaTripleID (nicht mediaTripleURL) der zu komprimierenden Medien-Datei */
5   title : string, /* Beliebiger Titel zur Identifizierung des Kompressions-Auftrags */
6   mimeType : string, /* MIME-Typ der zu komprimierenden Datei */
7   levels : [string], /* Array mit den Bezeichnern aller gewünschten Kompressions-Stufen */
8 }
```

Listing 35: POST-Parameter zum Absetzen eines Kompressions-Auftrags

Antwort:

```

1 {
2   success : boolean,
3   message : string
4 }
```

Listing 36: Antwort auf das Absetzen eines Kompressions-Auftrags

Funktion: Auflisten aller noch nicht abgeschlossenen Kompressions-Aufträge

HTTP-Methode: GET

Pfad: /jobs/dispatch

Antwort:

```

1 {
2   success : boolean,
3   message : string,
4   items : [
5     job : {
6       basePath : string, /* Unterverzeichnis, in welchem die Datei abgelegt ist, ansonsten "" */
7       objectUid : string, /* objectTripleID (nicht objectTripleURL) der Medien-Datei */
8       mediaUid : string, /* mediaTripleID (nicht mediaTripleURL) der Medien-Datei */
9       title : string, /* Beliebiger Titel zur Identifizierung des Kompressions-Auftrags */
10      mimeType : string, /* MIME-Typ der zu komprimierenden Datei */
```

```

11     levels : [string], /* Array mit den Bezeichnern aller gewuenschten Kompressions-Stufen */
12 },
13 receivedOn : timestamp, /* UNIX-Timestamp, an dem der Auftrag abgesetzt wurde */
14 id : int, /* Vom Kompressions-System vergebene ID fuer den Auftrag */
15 state : string, /* Aktueller Status des Auftrags, kann entweder ENQUEUED,
16 PROCESSING, ERROR oder COMPLETED sein */
17 lastStateChange : timestamp /* UNIX-Timestamp der letzten Statusaenderung des Auftrags */
18 }
}

```

Listing 37: Antwort auf das Auflisten nicht abgeschlossener Kompressions-Aufträge

Funktion: Löschen eines Kompressions-Auftrags**HTTP-Methode:** DELETE**Pfad:** /jobs/cancel/{ID}, mit {ID} der ID des Kompressions-Auftrags**Antwort:**

```

1 {
2   success : boolean,
3   message : string
4 }

```

Listing 38: Antwort auf das Löschen eines Kompressions-Auftrags

9.8.2 Abgeschlossene Kompressions-Aufträge

Funktion: Auflisten der letzten abgeschlossenen Kompressions-Aufträge**HTTP-Methode:** GET**Pfad:** /archive/jobs/**Antwort:**

```

1 {
2   success : boolean,
3   message : string,
4   items : [
5     job : {
6       basePath : string, /* Unterverzeichnis, in welchem die Datei abgelegt ist, ansonsten "" */
7       objectUid : string, /* objectTripleID (nicht objectTripleURL) der Medien-Datei */
8       mediaUid : string, /* mediaTripleID (nicht mediaTripleURL) der Medien-Datei */
9       title : string, /* Beliebiger Titel zur Identifizierung des Kompressions-Auftrags */
10      mimeType : string, /* MIME-Typ der zu komprimierenden Datei */
11      levels : [string], /* Array mit den Bezeichnern aller gewuenschten Kompressions-Stufen */
12    },
13    receivedOn : timestamp, /* UNIX-Timestamp, an dem der Auftrag abgesetzt wurde */
14    id : integer, /* Vom Kompressions-System vergebene ID fuer den Auftrag */
15    state : string, /* Aktueller Status des Auftrags, kann entweder ENQUEUED,
16    PROCESSING, ERROR oder COMPLETED sein */
17    lastStateChange : timestamp /* UNIX-Timestamp der letzten Statusaenderung des Auftrags */
18  ]
19 }

```

Listing 39: Antwort auf das Auflisten abgeschlossener Kompressions-Aufträge

9.8.3 Status des Kompressions-Systems

Der Zustand des Kompressions-Systems kann einen der folgenden Werte annehmen:

- **STARTUP:** Das System wird hochgefahren, mit der Abarbeitung von Kompressions-Aufträgen wurde noch nicht begonnen.
- **RUNNING:** Das System ist hochgefahren und bearbeitet Kompressions-Aufträge oder ist bereit dazu.
- **PAUSED:** Das System ist hochgefahren, die Abarbeitung von Kompressions-Aufträgen wurde jedoch pausiert.
- **SHUTTINGDOWN:** Das System wird heruntergefahren, weshalb das Absetzen weiterer Aufträge nicht möglich ist. Jedoch werden noch Kompressions-Aufträge verarbeitet.

- **SHUTDOWN:** Das System wurde heruntergefahren. Es können keine weiteren Aufträge abgesetzt oder verarbeitet werden.

Zum Setzen des Zustands des Kompressions-Systems kann einer der folgenden Werte verwendet werden:

- **RUN:** Setze den Zustand auf **RUNNING** und beginne mit der Bearbeitung von Kompressions-Aufträgen. Dieses Kommando ist nur in den Zuständen **PAUSED** oder **STARTUP** zulässig.
- **PAUSE:** Setze den Zustand auf **PAUSED** und pausiere damit die Abarbeitung der Kompressions-Aufträge nach dem Abschließen des aktuellen Auftrags. Dieses Kommando ist nur im Zustand **PAUSED** zulässig.
- **SHUTDOWN_PROCESS_QUEUE:** Setze den Zustand auf **SHUTTINGDOWN**, verarbeite aber vor dem Herunterfahren die gesamte Auftragsliste. Dieses Kommando ist nur im Zustand **STARTUP**, **RUNNING** oder **PAUSED** zulässig.
- **SHUTDOWN_IMMEDIATELY:** Setze den Zustand auf **SHUTTINGDOWN**, schließe aber vor dem Herunterfahren den sich aktuell in Verarbeitung befindlichen Auftrag ab. Dieses Kommando ist nur im Zustand **STARTUP**, **RUNNING** oder **PAUSED** zulässig.
- **KILL:** Setze den Zustand auf **SHUTDOWN**, fahre das System herunter ohne Rücksicht auf ausstehende oder sich in Verarbeitung befindliche Kompressions-Aufträge.

Funktion: Abrufen des Systemzustands

HTTP-Methode: GET

Pfad: /control/state

Antwort:

```

1 {
2   success : boolean,
3   message : string,
4   state : string /* {STARTUP | RUNNING | PAUSED | SHUTTINGDOWN | SHUTDOWN} */
5 }
```

Listing 40: Antwort auf das Abrufen des Systemzustands

Funktion: Setzen des Systemzustands

HTTP-Methode: PUT

Pfad: /control/state

POST-Paramter:

```

1 {
2   state : string /* {RUN | PAUSE | SHUTDOWN_PROCESS_QUEUE | SHUTDOWN_IMMEDIATELY | KILL} */
3 }
```

Listing 41: POST-Parameter zum Setzen des Systemzustands

Antwort:

```

1 {
2   success : boolean,
3   message : string
4 }
```

Listing 42: Antwort auf das Setzen des Systemzustands

9.8.4 Konfiguration des Kompressions-Systems

Für Details zu den einzelnen Konfigurationsoptionen wird auf Abschnitt 9.6 verwiesen.

Funktion: Abrufen der Systemkonfiguration

HTTP-Methode: GET

Pfad: /settings/config

Antwort:

```

1  {
2      success : boolean,
3      message : string,
4      config : {
5          apiPort : integer,
6          apiAccessWhitelist : [string],
7          autostart : boolean,
8          queueMaxLength : integer,
9          defaultLevels : [string],
10         textureLevelLimits : [integer],
11         textureLevelSizes : [integer]
12         imageCompressionLevels : [
13             maxWidth : integer,
14             maxHeight : integer,
15             title : string
16         ]
17     }
18 }
19 }
```

Listing 43: Antwort auf das Abrufen der Systemkonfiguration**Funktion:** Setzen der Systemkonfiguration**HTTP-Methode:** PUT**Pfad:** /settings/config**POST-Paramter:**

```

1  {
2      apiPort : integer,
3      apiAccessWhitelist : [string],
4      autostart : boolean,
5      queueMaxLength : integer,
6      defaultLevels : [string],
7      textureLevelLimits : [integer],
8      textureLevelSizes : [integer]
9      imageCompressionLevels : [
10         maxWidth : integer,
11         maxHeight : integer,
12         title : string
13     ]
14 }
```

Listing 44: POST-Parameter zum Setzen der Systemkonfiguration**Antwort:**

```

1  {
2      success : boolean,
3      message : string
4 }
```

Listing 45: Antwort auf das Setzen der Systemkonfiguration

10 VISIT METADATEN UND DIE SEMANTISCHE DATENBANK

Brainstorm, things to write about:

- theoretischer background: rdf daten, CIDOC, Vismo
- datenbank: infrastruktur (hosting, allgemeiner zugriff von aussen), drupal, wisski (allgemein), grundfunktionalität
- wisski: rdf daten, pfade, konfiguration
- REST API: allgemeine beschreibung
- zusatzfeatures: copy and paste, excel import

10.1 Theoretische Grundlagen für die Semantische Datenbank

Dieses Unterkapitel gibt Einblicke in Teilbereiche des Semantic Webs, um eine theoretische Grundlage für die folgenden technischen Entwicklungen zu geben. Nachdem diese erläutert wurden, wird ebenfalls auf eine spezielle Ausprägung eines Metadatenmodells eingegangen, welches die Struktur für die im ViSIT Projekt verwendeten Metadaten vorgibt: das ViSIT Model **VisMo**.

Die hier angeführten Ausführungen beschränken sich jedoch nur auf jeweilige Grundlagen der Themenkomplexe, welche an manchen Stellen um weiterführende Informationen erweitert werden, wenn dies für den weiteren Verlauf von Nöten ist. Dennoch, falls angestrebt, verweisen wir für ein tieferes Verständnis auf weitere Fachliteratur, wie z.B. [Hit+07].

SEMANTIC WEB UND RDF DATEN Das Semantic Web ist eine Art Erweiterung zum eigentlichen World Wide Web, wie wir es aktuell kennen. Dieses ist primär für Menschen ausgelegt, die durch Homepages browsen und dabei entsprechende Informationen durch betrachten und lesen der Homepages erlangen. Diese Informationen sind dadurch jedoch nur für Menschen vorhanden, Maschinen oder Computer können auf die Informationen nicht zugreifen, um mit den entsprechenden Daten arbeiten zu können. Genau hier setzt das Semantic Web an, welches Standardisierungen, Regeln und Prozesse vorgibt, um Homepages und Applikationen so anzupassen, dass eben genau eine (semi-) automatische Informationsverarbeitung für Maschinen möglich wird.

Eine dieser Standardisierungen ist das Resource Description Framework **RDF** [MMo4], welches der de-facto Standart im Semantic Web ist, um Metadaten zu beschreiben. Daten in RDF werden als Graph modelliert und persistiert, welcher aus Knoten und Kanten besteht. Dabei entsteht eine Wissensbasis gefüllt an Informationen. Die Knoten sind hierbei die "Akteure", also diejenigen Entitäten, Sachen, Objekte, Dinge etc., ausgehend vom jeweiligen Anwendungsfall, auf die sich die im Graphen enthaltenen Informationen beziehen (diese Dinge werden im Folgenden weiterhin als "Metadatenentität" bezeichnet). Die Kanten im Graphen beschreiben Beziehungen zwischen den gegebenen Knoten und Eigenschaften der Knoten. Weiterhin sind die Knoten und Kanten durch das Grundprinzip eines **Statements** verbunden, welches eine Kapselung einer elementaren Aussage darstellt. Das Statement ist, ähnlich dem deutschen Satzbau, immer bestehend aus drei Teilen:

SUBJEKT Die Metadatenentität repräsentiert als ein Knoten im Graphen, von der die Aussage - und damit das Prädikat - des Statements ausgeht.

PRÄDIKAT Die Semantik oder die Bedeutung der Aussage.

OBJEKT Zweierlei Konzepte können das Objekt des Statements bilden: ein weiterer Knoten im Graphen, um das Ziel der Aussage und damit des Prädikats, um eine Relation zwischen zwei Metadatenentitäten/Knoten darzustellen, oder ein fester Wert, um eine Eigenschaft einer Metadatenentitäten/eines Knotens zu charakterisieren.

Zur Verständlichkeit für die Thematik der Aussagen und Statements im Semantic Web Kontext, soll hier ein kurzes, erfundenes Beispiel erläutert werden. Folgende Aussagen bilden die Wissensbasis:

- Peter ist vom Beruf Baumeister.
- Peter ist 40 Jahre alt.
- Peter war am Bau des Steinschlosses beteiligt.
- Das Steinschloss besteht aus Stein.
- Das Steinschloss ist 10 Jahre alt.

Wie oben beschrieben, bestehen die Aussagen jeweils aus Subjekt, Prädikat und Objekt. Als Subjekte agieren die beiden Metadatenentitäten "Peter" und das "Steinschloss", während die Objekte der Aussagen der Beruf "Baumeister", das Material "Stein", zwei "Altersangaben", sowie das "Steinschloss" selbst sind. Semantisch sind die Subjekte und Objekte über die Beziehungen bzw. Eigenschaften einer "Berufszuordnung", zwei "Alterszuordnungen", einer "Materialzuweisung" sowie der "Erbauung" eines Objekts verbunden.

Diese Aussagen können nun in einen Graphen zusammengefasst werden, dessen high-level Illustration in [Abbildung 50](#) zu sehen ist.

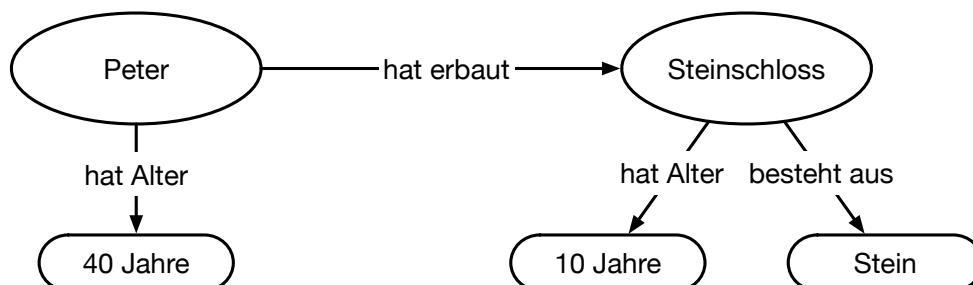


Abbildung 50: Informationen aus obigen Aussagen, kombiniert als Graph.

LINKED OPEN DATA GEDANKE Ein weiterer Eckpfeiler des Semantic Web ist ein weiteres Konzept, das unter dem Namen **Linked Open Data - LOD** bekannt ist. Oft wird dieser Name ebenfalls für das Semantic Web selbst benutzt, die punktgenauen Definitionen überschneiden und ergänzen sich.

Einfach übersetzt zielt LOD auf öffentlich zugängliche Daten ab, die untereinander vernetzt und verlinkt sind. Somit soll es möglich sein, verteilte Datenbanken mit ihren eigenen entsprechenden Wissensbasen, miteinander zu verbinden, um so jedem Beteiligten mehr Informationen zur Verfügung zu stellen, da durch die Verlinkung einzelner Graphen ein großer Gesamtgraph entsteht. Auf diese Weise macht es Sinn, dass jede Wissensbasis ihren eigenen spezialisierten Kontext besitzt. Sollte eine Wissensbasis weitere Informationen aus einem anderen Kontext benötigen, müssen diese Daten nicht auf eigene Hand erforscht und aufbereitet werden, da eine LOD Verbindung zu einer anderen Wissensbasis hergestellt werden kann. Zur weiteren Veranschaulichung dieser Thematik und dessen Vorteile, zeigt der folgende Paragraph zwei Anwendungsfälle im geschichtswissenschaftlichen Kontext.

ZWEI ANWENDUNGSFÄLLE FÜR RDF IM GESCHICHTSWISSENSCHAFTLICHEN KONTEXT Ein erster Anwendungsfall, von dem geisteswissenschaftliche Wissensbasen profitieren können, ist oben bereits kurz angedeutet worden: das Verbinden einer eigenen Wissensbasis mit externen, bereits bestehenden Wissensbasen. Das Erforschen und Erkunden von Wissen benötigt generell in jeglichem Kontext sehr viel Zeit und ebenfalls Pflege der Daten. Daher kommt diesem Anwendungsfall der LOD Gedanke entgegen, da bereits erstellte Wissensbasen und deren Datenbanken öffentlich zugänglich sind.

Gerade generelle Themen oder Kontexte wie Personen, Städte oder Orte werden in vielen geschichtswissenschaftlichen Projekten benötigt, und gerade diese sind in öffentlichen Datenbanken zugänglich. Daher ist es für diese Anwendungsfälle sinnvoll, den eigens entwickelten Anwendungsfall an diese Datenbanken zu knüpfen. Dadurch wird der eigene Zeitaufwand erheblich reduziert und die angebundenen Daten genießen in der Regel außerdem einen hohen Standard, da bereits viele potenzielle Reviews von anderen Nutzern bestehen.

Ein zweiter großer Vorteil davon, geschichtswissenschaftliche Daten in Form von Metadaten und RDF zu persistieren, ist das mögliche Erschließen von vorher nicht bekannten oder erforschten Zusammenhängen der persistierten Objekte. Dazu folgendes (frei erfundenes) Beispiel: Ausgehend von der eigenen Wissensbasis, die die Daten aus [Abbildung 50](#) enthält, sollen nun zwei weitere Wissensbasen angekoppelt werden, welche auf der einen Seite weitere Informationen über Personen und vor allem deren familiärer Beziehungen beinhaltet, und auf der anderen Seite eine Wissensbasis, die mehr Informationen über Gebäude und deren Geschichte beinhaltet. Dies ist in [Abbildung 51](#) visualisiert.

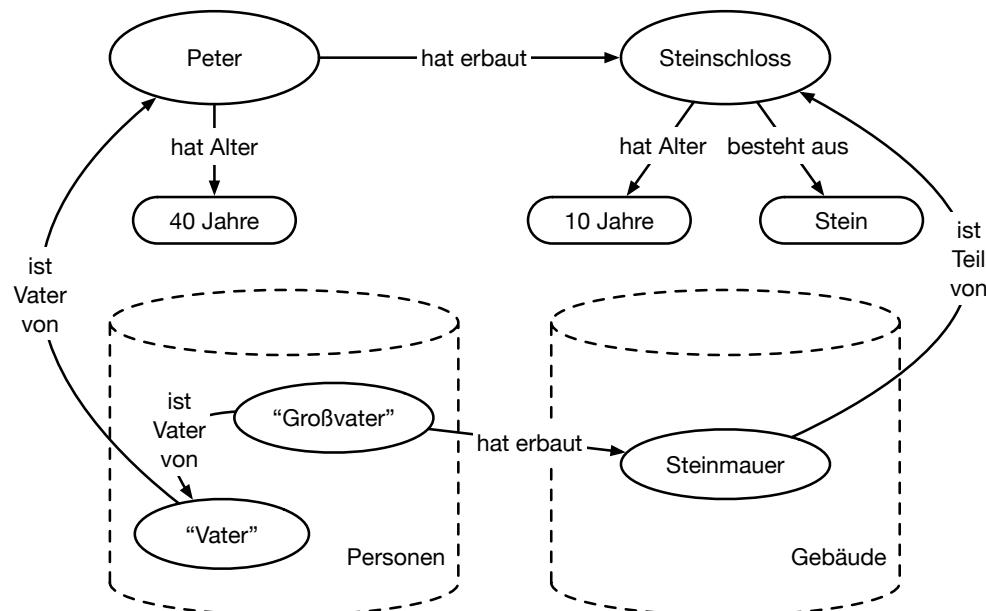


Abbildung 51: Grundlegende eigene Wissensbasis (oben), erweitert um zwei externe Wissensbasen (unten).

In dem Beispiel beinhaltet die eigene Wissensbasis Informationen über "Peter" und das "Steinschloss". Durch die beiden hinzugenommenen Wissensbasen wird Peter aus dem Anwendungsbeispiel mit seinem "Vater", und dieser wiederum mit seinem "Großvater" verbunden (die Namen sind hier zur Einfachheit ersetzt). Zudem wird die "Steinmauer" als ein Teil des Steinschlosses deklariert. Die beiden neuen Wissensbasen enthalten darüber hinaus bereits implizit eine eigene Verbindung, die semantisch besagt, dass der "Großvater" am Bau der Steinmauer beteiligt ist.

Dadurch erweitern die beiden externen Wissensbasen die eigenen Informationen durch die neu erstellten Relationen. Darüber hinaus jedoch lässt sich so ebenfalls eine neue Erkenntnis in den Daten schliessen: sowohl "Peter" als auch dessen "Großvater" sind direkt oder indirekt am Bau des "Steinschlosses" beteiligt.

DAS CONTEXTUAL REFERENCE MODEL – CIDOC CRM Bisher war die technische Beschreibung der semantischen Daten im ViSIT Kontext aus Gründen der Einfachheit sehr flach gehalten. Gemäß den Semantic Web Standards basieren die Metadaten jedoch auf einem Datenmodell, um die Anforderungen des Semantic Webs zu genügen und ebenfalls technische Verarbeitbarkeit zu gewährleisten.

In ViSIT ist die Wahl hierbei auf das **Contextual Reference Model CIDOC CRM** [Doeo3] gefallen, da dies eine der bekanntesten und vorherrschendsten Ontologien im Bereich des kulturellen Erbes ist. Diese Ontologie wird als Basis benutzt, die im folgenden Paragraphen erweitert für den ViSIT Kontext beschrieben wird. Der größte Vorteil dieser Ontologie ist, dass sie sich nicht auf einen speziellen Bereich des kulturellen Erbes fokussiert ist, sondern auf generische Weise komplexe Zusammenhänge und verschiedene Themengebiete abbildet. Zudem solle es möglich sein, andere Ontologien oder Modelle aus dem selben Bereich in diese Ontologie zu überführen, um eine gemeinsam verständliche Wissensbasis zu kreieren.

Das CIDOC CRM wird seit mittlerweile über 10 Jahren von der CIDOC Documentation Standards Working Group¹² und der CIDOC CRM SIG¹³ entwickelt, welche beide Arbeitsgruppen von CIDOC¹⁴ sind. Das CIDOC CRM ist 2000 als "Working Draft" bei der ISO/TC46/SC4¹⁵ akzeptiert worden, welcher 2006 schliesslich auch als offizieller Standard [Cida] akzeptiert wurde, und 2014 in eine überarbeitete Version [Cidb] überführt wurde.

In der aktuellen Hauptversion 6.2¹⁶, die im Mai 2015 veröffentlicht wurde, enthält die Ontologie 89 RDF Klassen und 149 einzigartige Relationen und Eigenschaften, die sich in einer mehrfach ineinander- sowie auseinander verzweigenden Struktur einordnen. Laufend werden ebenfalls Nebenversionen veröffentlicht - die aktuellste Versionsnummer lautet 6.2.3¹⁷.

DAS VISIT MODEL – VISMO Aufbauend auf dem CIDOC CRM wurde eine Ontologie entwickelt, die den kompletten Anwendungsfall des ViSIT Projekt abbilden kann: das **ViSIT Model VisMo**. Der Fokus liegt dabei auf der Darstellung von Architektur-Objekten und Ausstellungsobjekten, die mit Personen oder Gruppen von Personen, Orten sowie zeitlichen Events in Verbindung gesetzt werden, um eine Wissensbasis zu kreieren.

Diesbezüglich sind die Hauptentitäten, die in der ViSIT Datenbank angelegt werden können, die folgenden:

EREIGNIS (ACTIVITY): Diese Entität umfasst alle vergangenen und zukünftigen Vorgänge und Geschehnisse in kulturellen, sozialen und physischen Systemen, analog zum "E5_Event"¹⁸ des CIDOC CRM.

BAUWERK (ARCHITECTURE): Diese Entität bezeichnet alle Arten von Bauten, die wie die Objekte als Informationsträger (vgl. "E84_Information_Carrier"¹⁹ des CIDOC CRM) betrachtet werden.

GRUPPE (GROUP): Diese Entität bezeichnet mehrere Personen, die sich zu einer Gruppierung zusammengeschlossen haben und durch eine gleiche oder ähn-

¹² <http://network.icom.museum/cidoc/working-groups/overview/>

¹³ <http://network.icom.museum/cidoc/working-groups/crm-special-interest-group/>

¹⁴ <http://network.icom.museum/cidoc/>

¹⁵ <https://www.iso.org/committee/48798.html>

¹⁶ <http://www.cidoc-crm.org/Version/version-6.2>

¹⁷ <http://www.cidoc-crm.org/Version/version-6.2.3>

¹⁸ <http://www.cidoc-crm.org/Entity/E5-Event/Version-6.2>

¹⁹ <http://www.cidoc-crm.org/Entity/E84-Information-Carrier/Version-6.2>

liche Tätigkeit miteinander verbunden sind (vgl. "E74_Group"²⁰ des CIDOC CRM).

INSTITUTION (INSTITUTION): Diese Entität bezeichnet hier alle Arten von organisierten Einrichtungen, die keine natürlichen Personen oder Personengruppen sind, z.B. Museen, Archive, Bibliotheken, Universitäten usw.

OBJEKT (OBJECT): Diese Entität bezeichnet alle Arten von Ausstellungsobjekten aus den Sammlungen von musealen oder museumsähnlichen Institutionen, Archiven etc., die wie Bauwerke als Informationsträger betrachtet werden (vgl. "E84_Information_Carrier"²¹ des CIDOC CRM).

PERSON (PERSON): Diese Entität bezeichnet analog zur Definition im CIDOC CRM der "E21_Person"²² alle natürlichen Personen, die leben oder bereits verstorben sind sowie Personen, von denen angenommen wird, dass sie lebten. Dazu zählen historische Persönlichkeiten als auch Personen aus Legenden, Mythen und Sagen.

ORT (PLACE): Diese Entität bezeichnet hier ausschließlich über Koordinaten lokalisierbare verschwundene und bestehende Dörfer und Städte.

LITERATUR (REFERENCE): Diese Entität bezeichnet alle Arten von niedergeschriebenen und veröffentlichten Texten.

Dabei erfüllt VisMo genau den Zweck, den sich das CIDOC CRM als Ziel gesetzt hat: als eine semantische "Erweiterung" des CIDOC CRM ist der Inhalt, der für VisMo produziert wird, direkt zum größten Teil verständlich und Leser oder Benutzer des Modells können dies intuitiver, auf der Basis der Beschreibungen des CIDOC CRM, verstehen, lesen und benutzen. Dies ist dadurch begründet, dass alle Klassen und viele der Relationen und Eigenschaften durch Vererbung speziellere Konzepte der CIDOC CRM Klassen und Relationen/Eigenschaften sind. Nur einzelne Teile des VisMo sind speziell für die Ontologie hinzugefügt worden, immer wenn kein Konzept aus dem CIDOC CRM passend für eine Vererbung war. [Abbildung 52](#) visualisiert den Entwicklungsprozess hinter VisMo.

Der erste Schritt bestand dabei in der Sammlung der Kernthemen, die in ViSIT behandelt werden. Aus diesen konnte dann im nächsten Schritt ein grobes Konzept entwickelt werden, welches anschließend in RDF übertragen werden konnte. Wie oben beschrieben, wurde hierbei von CIDOC CRM Grundklassen und Relationen bzw. Eigenschaften ausgegangen, welche dann für den ViSIT Kontext erweitert und angepasst wurden. Als nächstes konnten dann die erstmals groben Konzepte und Entitäten mit benötigten Metadaten bzw. dessen Anforderungen erweitert werden. Die Ergebnisse der vorherigen Schritte konnten dann letztendlich in dem Ontologie-Editor [protege](#)²³ zusammengeführt werden, um eine RDF/OWL Ontologie zu erstellen. Diese ist in ihrer letzten offiziellen Version in [Listing 49](#) im Appendix zu sehen.

Ebenfalls ist in [Abbildung 52](#) visualisiert, wie und an welcher Stelle die VisMo Ontologie technisch zum Einsatz kommt: sie dient als Input für das sogenannte WissKI Modul, um aus der Ontologie Ein- sowie Ausgabemasken zu generieren, welche letztendlich vom Endnutzer des ViSIT Systems benutzt werden, um einerseits Daten in die semantische Datenbank einzutragen und diese dann auch wieder auszulesen und anzuzeigen. Der große Vorteil an diesem Prozess ist, dass der Endnutzer keinerlei Wissen über das Semantic Web und seine Technologien benötigt, da der oben beschriebene Prozess davon abstrahiert. Damit schreiben und lesen die Endnutzer im Endeffekt RDF, ohne davon zu wissen. Technische Details zu diesem Prozess sowie WissKI werden in folgenden Unterkapiteln gegeben.

²⁰ <http://www.cidoc-crm.org/Entity/E74-Group/Version-6.2>

²¹ <http://www.cidoc-crm.org/Entity/E84-Information-Carrier/Version-6.2>

²² <http://www.cidoc-crm.org/Entity/E21-Person/Version-6.2>

²³ <https://protege.stanford.edu/>

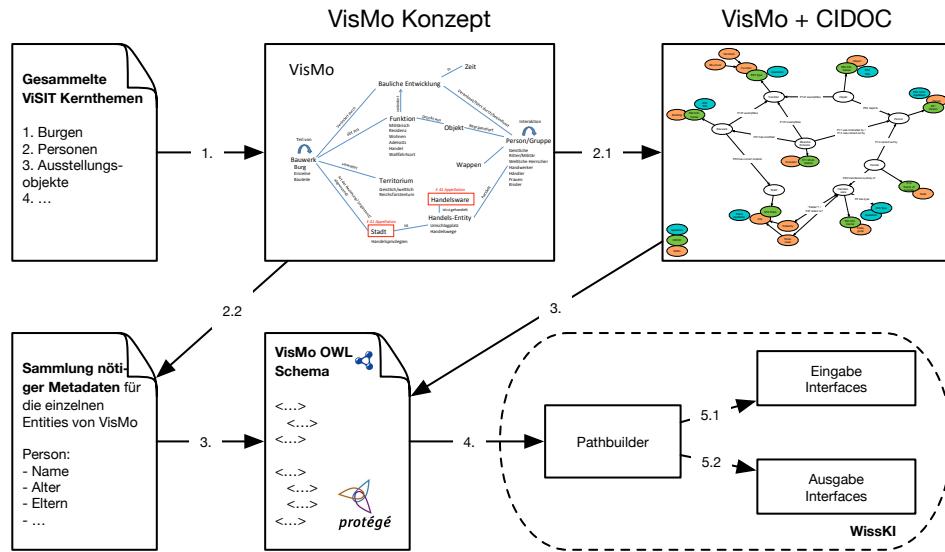


Abbildung 52: Arbeitsprozess hinter der Entwicklung des ViSIT Modells.

10.2 Technische Details zur Semantischen Datenbank

Nachdem [Unterabschnitt 10.1](#) die theoretische Grundlage für die Semantische Datenbank beschrieben hat, fokussiert sich dieses Unterkapitel auf die technischen Aspekte der Datenbank. Dazu zählt in erster Linie die **allgemeine Infrastruktur**, das **Hosting** an der Universität Passau, der **allgemeine Zugriff auf die Datenbank**, getroffene Entscheidungen bezüglich **Security und Zertifizierungen**, sowie die anschließende Beschreibung einzelner Komponenten: dem **CMS Drupal**, dessen Modul **WissKI**, die **ViSIT REST API**, der unterliegende RDF Triplestore **RDF4J** und dessen generelle Funktionalität.

Für die semantische Datenbank wurde zur Projektlaufzeit aus Testzwecken ebenfalls eine Testinstanz ins Leben gerufen, welche eine komplette Spiegelung des damals aktuellen Systems ist. Die beiden Haupt-URLs der Server sind:

- <https://database.visit.uni-passau.de/>
- <https://database-test.visit.uni-passau.de/>

Von diesen beiden Base-URLs ausgehend sind die weiteren Komponenten über folgende URL-Zusätze zu erreichen:

- **Drupal/WissKI**: Base URL + /drupal
- **RDF4J**: Base URL + /rdf4j-workbench
- **Tomcat**: Base URL (ohne Zusatz)
- **ViSIT REST API**: Base URL + /metadb-rest-api
- **API Beschreibung**: Base URL + /metadb-test-api/swagger-ui.html

INFRASTRUKTUR Die Semantische Datenbank des ViSIT Projekts ist auf einem virtuellen Server an der Universität Passau installiert. Die allgemeine Infrastruktur ist in [Abbildung 53](#) zu sehen.

Dessen Hauptkomponenten mit Beschreibung oder Verweis auf das ausführliche Unterkapitel sind die folgenden:

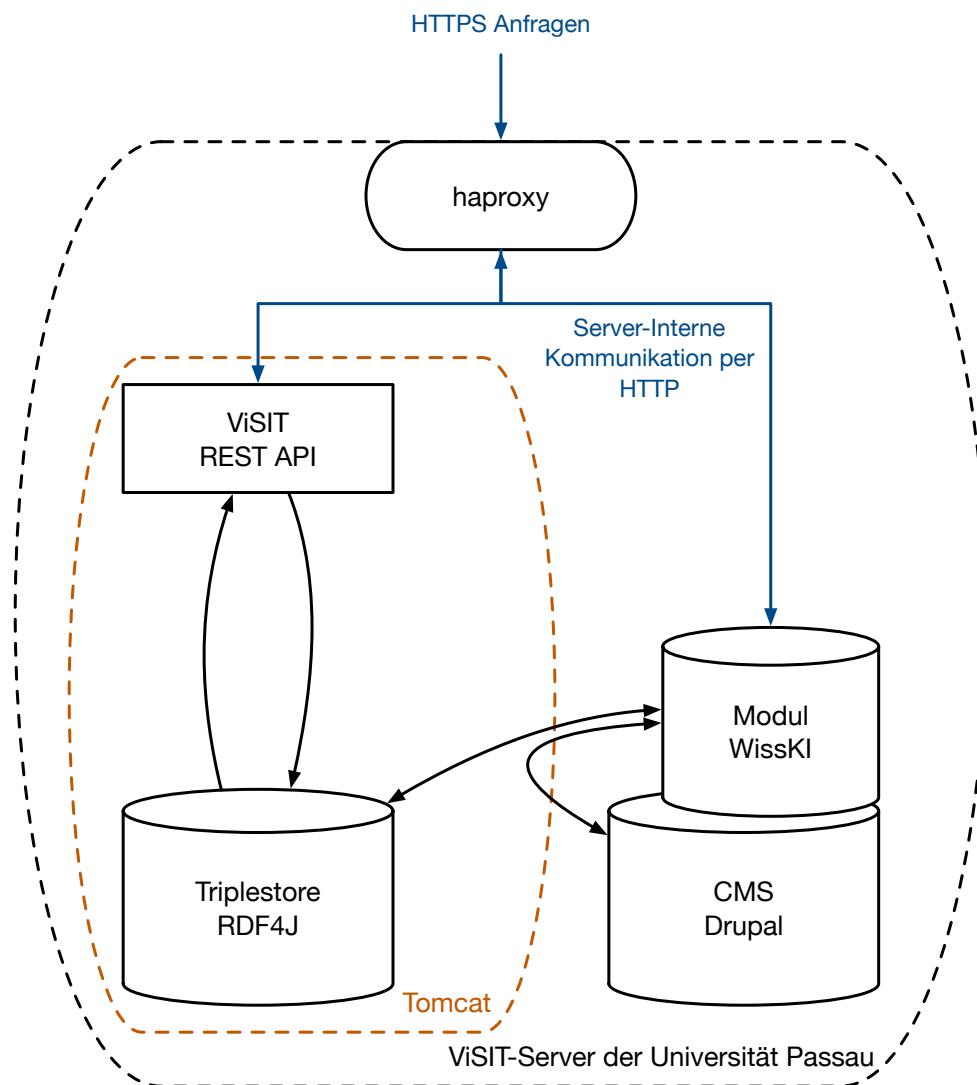


Abbildung 53: Technische Infrastruktur der Semantischen Datenbank des ViSIT Projekts.

HAPROXY Dem virtuellen Server für die ViSIT Infrastruktur ist ein **haproxy**²⁴ vorgeschaltet. Dieser ist dafür da, die per HTTPS verschlüsselten Anfragen von aussen an den Server entgegen zu nehmen, und intern an die richtigen Komponenten weiterzuleiten. Prinzipiell kann dieser haproxy ebenfalls Anfragen per HTTP entgegen nehmen, leitet diese dann aber automatisch auf den Port für HTTPS weiter. Damit ist sicher gestellt, dass nach aussen nur verschlüsselte Daten versandt werden. Dem haproxy sind für die benötigte Funktionalität zwei backends bekannt: eines für den Tomcat (ViSIT REST API und Triplestore RDF4J) und eines für Drupal bzw. WissKI (welche hintergrundig auf einem Apache laufen). Die Verschlüsselung ist durch ein SSL Zertifikat der Universität gewährleistet. Die Konfiguration zum haproxy ist am Server zu finden unter /etc/haproxy.

TOMCAT Zur Installation weiterer Komponenten am Server, ist ein **Apache Tomcat**²⁵ in der Version 8.5.24 installiert. Am Server ist dieser zu finden unter /opt/tomcat8/apache-tomcat-8.5.24.

VISIT REST API Diese API wurde eigens für ViSIT entwickelt, um eine Abstraktionsschicht für die unterliegenden Metadaten zu bieten. Während WissKI die-

²⁴ <http://www.haproxy.org/>

²⁵ <http://tomcat.apache.org/>

se Abstraktion für die wissenschaftlichen Benutzer der Metadaten bildet, ist die API für die technische Anbindung der restlichen Komponenten des ViSIT Projekts zuständig. Die API bietet in erster Linie die Möglichkeit, die RDF Metadaten aus dem Triplestore zu lesen. Zurückgegeben wird das Ergebnis im JSON Format, um eine möglichst breite Verständnis und damit direkte Verwendbarkeit zu gewährleisten. Der zweite große Teil behandelt das Schreiben, Auslesen und Updaten der sogenannten technischen Metadaten: Metadaten, die Informationen zu einem Medien-Objekt im ViSIT Kontext geben. Die REST API ist als eigenständiges Java-Projekt implementiert, welches auf dem ViSIT Server bzw. in dessen Tomcat Installation deployed wird. Eine ausführliche Beschreibung wird in [Unterabschnitt 10.4](#) gegeben.

TRIPLESTORE RDF4J Der Triplestore ist für die Persistierung der RDF Daten zuständig. Im ViSIT Kontext ist die Wahl hierfür auf die **RDF4J²⁶** Datenbank gefallen, dieser ist jedoch durch jeglichen gleichwertigen Triplestore ersetzbar. Wie bei der REST API beschrieben, wird im ViSIT Kontext weitestgehend möglich von den RDF Daten abstrahiert. Dies wird sowohl durch die REST API und dem WissKI Modul bewerkstelligt. Der RDF4J Triplestore ist am Server bzw. in dessen Tomcat Installation deployed.

DRUPAL UND WISSKI Die letzte Komponente der Infrastruktur der Semantischen Datenbank ist eine Kombination aus dem Content Management System **Drupal²⁷** und dessen Modul **WissKI - Wissenschaftliche KommunikationsInfrastruktur²⁸**. Als Modul baut WissKI auf der Implementierung von Drupal auf und benutzt dessen Funktionalität zum Persistieren von Entities als Inhalt. Zudem, da WissKI aber ebenfalls mit RDF Daten arbeitet, wird ein Triplestore benötigt - wie oben beschrieben. WissKI übernimmt dabei die Synchronisation zwischen den Entities in Drupal und den RDF Daten, sowohl beim Speichern als auch beim Auslesen von Daten. Weiterhin bietet WissKI die Möglichkeit, ein eigenes Datenmodell zu definieren, welches den gesamten Datenfluss eine Struktur vorgibt. Aus diesem werden ebenfalls einfache Interfaces generiert, um auf der einen Seite die Daten anzulegen, und auf der anderen Seite auf eine einfache Weise darzustellen. Weitere Details hierzu werden in [Unterabschnitt 10.3](#) beschrieben.

Im Zusammenspiel der obig genannten Komponenten erlaubt das gesamte System der Semantischen Datenbank das Management der semantischen Daten, die für den Kontext des ViSIT Projekts benötigt werden. Der Workflow der Datenbank sieht dabei in etwa wie folgt aus:

- Über die einfachen WissKI Eingabe-Interfaces geben die Kuratoren bzw. Geisteswissenschaftler Informationen und Metadaten in das Gesamtsystem ein.
- Diese Metadaten werden mit der WissKI Funktionalität automatisch ebenfalls in RDF Daten übersetzt, die dem Schema entsprechen, welches bei Installation und Konfiguration des Gesamtsystems erstellt wird (siehe [Unterabschnitt 10.1](#) für das Metadatenmodell CIDOC + VisMo, [Unterabschnitt 10.3](#) für die Konfiguration von WissKI).
- Für Forschungszwecke können diese angelegten Metadaten dann mit den WissKI Ausgabe-Interfaces betrachtet werden, was ebenfalls die Graphstrukturen hinter den Daten hervorhebt, da in den Interfaces zwischen den einzelnen Entitäten navigiert werden kann.
- Die ViSIT REST API dient zur technischen Anbindung weiterer ViSIT Komponenten, indem die Metadaten auf standardisierte Weise abgefragt werden können.

²⁶ <http://rdf4j.org/>

²⁷ <https://www.drupal.org/>

²⁸ <http://wiss-ki.eu/>

10.3 WissKI – Wissenschaftliche Kommunikationsinfrastruktur

Das WissKI Modul bietet die Möglichkeiten, sowohl RDF Daten zu lesen und zu schreiben - aber auf eine einfache Weise über simpel gehaltene Eingabe- und Ausgabeinterfaces, um den Zugang für Forscher und die Geisteswissenschaftler im ViSIT Kontext zu gewährleisten. Um dies jedoch zu bewerkstelligen, benötigt das Modul verschiedene Konfigurationen und Einstellungen. Unter anderem das wichtigste ist das Definieren der semantischen Struktur der Daten, wie es bereits oben beschrieben wurde.

Für den ViSIT Anwendungsfall ist das technische System der semantischen Datenbank, beschrieben in [Abschnitt 10](#), vollständig konfiguriert und betriebsbereit. Nichtsdestotrotz werden in den folgenden Unterabschnitten die Einstellungen für WissKI erläutert, um für potenziell zukünftige Änderungen eine grundlegende Beschreibung zu geben. Diese Beschreibungen können jedoch nie eine Tiefe und Genauigkeit erreichen, wie sie von den WissKI Entwicklern gegeben werden kann. Deswegen sei hier ebenfalls auf <http://wiss-ki.eu/> verwiesen.

WISSKI SALZ ADAPTER Wie ebenfalls bereits in [Unterabschnitt 10.2](#) beschrieben, regelt das WissKI System das Persistieren und Auslesen der im Gesamtsystem angewandten semantischen Daten. Als ein Modul für das CMS Drupal, werden die Daten auf der einen Seite im CMS als Entitäten gespeichert, auf der anderen Seite - da die Daten auf Semantic Web Standards basieren sollen - als RDF Daten in einem Triplestore. WissKI führt hier automatisch die Konvertierung zwischen den beiden Datenbanken durch, ohne dass der Nutzer hier aktiv werden müsste.

Die Verbindung mit dem Drupal CMS geschieht automatisch mit der Installation des WissKI Moduls. Was jedoch konfiguriert werden muss ist die Verbindung des Moduls zum zu verwendenden Triplestore. Dies passiert im sogenannten **WissKI Salz Adapter**.

Wenn das Menü zum bearbeiten der Adapter geöffnet wird, erscheint eine Liste der aktuell definierten Adapter. Für das ViSIT Projekt ist bereits ein Adapter eingerichtet mit dem Namen `visittestrepo`. Grundsätzlich reicht für einen Anwendungsfall wie ViSIT ein Adapter, es können aber natürlich beliebig viele Adapter definiert werden. [Abbildung 62](#) und [Abbildung 63](#) im Appendix zeigen die Konfigurationsmöglichkeiten eines WissKI Salz Adapters, bzw. die Einstellungen die für ViSIT getätigten wurden.

Die wichtigsten Endpunkte bzw. Konfigurationsmöglichkeiten sind die folgenden (die hier nicht erwähnten Punkte können in der Regel auf der Standardkonfiguration bzw. leer gelassen werden):

ADAPTER NAME: Der Name des Adapters, mit dem dieser eindeutig identifiziert werden kann.

WRITEABLE UND PREFERRED LOCAL STORE: Diese beiden Checkboxen sollten in der Regel immer gesetzt sein, wenn es sich um den Adapter bzw. Triplestore handelt, der hauptsächlich mit dem System arbeiten soll. "Writeable" bedeutet, dass Daten auf dem Triplestore geschrieben werden dürfen, "Preferred Local Store" weist das System an, diesen entsprechenden Adapter als Hauptadapter zu benutzen, falls mehrere definiert sein sollten.

READ UND WRITE URL: Dies sind die beiden Einstellungen, die WissKI mit dem Triplestore verbinden. Es sind die beiden URLs des entsprechenden Triplestores, auf die bei diesem lesend bzw. schreibend zugegriffen werden kann. Nur wenn diese beide gesetzt sind, kann das System richtig in Betrieb genommen werden. Die beiden URLs, die in den Bildern gesetzt sind, zeigen also auf den Triplestore, der in der Infrastruktur für die semantische Datenbank installiert wurde. (Zusätzliche Hintergrundinformation: die URLs zeigen hier auf "[http://localhost:8081/...](http://localhost:8081/)" und damit auf eine lokale Installation, da

sowohl das WissKI /CMS System und der Triplestore auf dem selben Server installiert ist. Die beiden Komponenten kommunizieren lokal miteinander.)

DEFAULT GRAPH URI: Für RDF Daten werden eindeutige URI Bezeichner für die Knoten und Kanten des RDF Graphen benötigt. In der Regel erhalten die Knoten, wenn sie für Instanzen bzw. Entitäten stehen, eine zufällig generierte Zeichenkette als URI. Die Default Graph URI wird dann verwendet, um vor diese Zeichenkette gesetzt zu werden. Somit entstehen URI Bezeichner, die auf die Semantic Web Standards passen und auch den eigenen Anwendungsfall besser repräsentieren: so wie im Beispiel für das ViSIT Projekt mit "<http://visit.de/data>". Eine beispielhafte URI wäre also "<http://visit.de/data/5c62c9aab4666>".

REITER COMPUTE TYPE AND PROPERTY HIERARCHY: Ein weiterer wichtiger Punkt im Bezug auf das Modell und damit die Struktur der semantischen Daten befindet sich im Reiter mit dem Namen "Compute Type and Property Hierarchy and Domains and Ranges". Öffnet man den Reiter, erhält man die Möglichkeit (nachdem die Checkbox "Re-Compute results" betätigt wurde), durch den Button "Start Reasoning" einen sogenannten Reasoning Prozess zu starten. Einfach formuliert betrachtet dieser die aktuell definierten Modelle des Systems, um potenziell zusätzliche Informationen hinzuzufügen. Dadurch kann das System auf schnellere Weise arbeiten, da diese Informationen nicht erst im produktiv laufenden Zustand des Systems hinzugefügt werden müssen. Diesen Prozess zu starten ist sehr wichtig, wenn ein **Update oder eine Änderung des Metadatenmodells passiert** ist. Der Prozess kann einige Minuten in Anspruch nehmen, bis er vollständig durchgeführt wurde.

WISSKI ONTOLOGY In diesem Teil der Konfiguration kann die unterliegende Ontologie bzw. das Metadatenmodell für das System definiert werden. Dazu wird zunächst in einem Drop-Down Menu der Adapter ausgewählt, für den dies getan werden soll. Weiterhin muss dann eine RDFS oder OWL Schema Datei in das WissKI System hochgeladen werden. Dazu ist ein entsprechender Button vorgesehen.

Wenn bereits eine Ontologie für einen Adapter existiert, wird diese bzw. vielmehr dessen enthaltene Namensräume angezeigt. Zusätzlich gibt es dann die Möglichkeit, die aktuelle Ontologie zu löschen, womit wieder zum ursprünglichen Zustand - einer nicht vorhandenen Ontologie samt Upload Button - zurückgekehrt wird.

Auf diese Weise kann eine Ontologie ausgetauscht werden. Vorsicht jedoch hier: Beim Austauschen einer Ontologie sollte darauf geachtet werden, dass die alte Ontologie in der neuen Ontologie enthalten ist, damit die aktuell definierten Pfade (siehe nächsten Unterabschnitt) nicht invalidiert werden.

Ein Beispiel für das VisMo Modell, welches für das ViSIT Projekt im entsprechenden WissKI Modul definiert ist, ist in [Abbildung 64](#) im Appendix zu sehen.

PATHBUILDERS Die Pfade des WissKI Moduls bilden das eigentliche Herzstück, da ausgehend von diesen Pfaden alle weiteren Komponenten, wie zum Beispiel die Eingabe- sowie Ausgabeinterfaces, generiert werden. Dieser Unterabschnitt wird einen Einblick in die Konfiguration des ViSIT Projekts geben. Wie eingangs zu diesem Kapitel jedoch bereits erwähnt, kann dieser Einblick nie alle Details des Moduls abdecken. Deswegen sei hier nochmals auf <http://wiss-ki.eu/> für detailliertere und ausführlichere Erklärungen verwiesen.

Die erste Übersicht beim Navigieren auf das Pathbuilders Menü listet alle vorhandenen Pathbuilder auf, die aktuell im entsprechenden WissKI System definiert sind. Genauso wie für die Salz Adapter und die Ontologie genügt es aber auch hier, einen Pathbuilder zu benutzen. Der für das ViSIT Projekt konfigurierte Pathbuilder trägt den Namen "visittestrepo_paths".

Wird dieser editiert, gelangt man in die Übersicht aller in diesem Pathbuilder befindlichen Pfade. Dies ist beispielhaft in [Abbildung 65](#) zu sehen.

Ein WissKI Pfad kann intern eine Gruppe (zur Gruppierung mehrerer Pfade für das selbe Objekt) oder ein wirklicher Pfad sein und besteht prinzipiell aus drei wichtigen Komponenten:

ID Eindeutiger Identifikator für den gesamten Pfad innerhalb des WissKI Systems.

PFAD Einzelner Knoten für eine Gruppe, oder eine Folge von RDF Knoten, Relationen und Eigenschaften, die den Pfad im RDF Graphen widerspiegeln sollen.

DATENTYP Definition des Werts des jeweiligen Pfades. Bei primitiven Datentypen endet der Pfad in einer RDF Eigenschaft, während eine sogenannte "entity reference" eine Relation, also einer Verbindung zu einem weiteren Knoten bzw. einer WissKI Entität entspricht.

Drei Beispiele sollen diesen Sachverhalt weiter erklären. [Abbildung 54](#) zeigt zwei Pfade, wie sie für das ViSIT Projekt definiert wurden: der obere "Pfad" ist eine Gruppe für das allgemeine Museumsobjekt. Dessen ID ist "Object", während der Pfad nur auf "<http://visit.de/ontologies/vismo/Object>" gesetzt ist. Dies lässt sich dadurch erklären, dass die Gruppe für den Ursprungsknoten eines dieser Entitäten steht, welcher den RDF Typen "<http://visit.de/ontologies/vismo/Object>" besitzen soll. Weiterhin benötigt eine Gruppe keinen Datentypen.

	Object	Group [http://visit.de/ontologies/vismo/Object]	<input checked="" type="checkbox"/>	Unlimited	<button>Edit</button>
	Object_identifiedBy_Title	http://visit.de/ontologies/vismo/Object -> ecrm:P1_is_identified_by -> ecrm:E35_Title	<input checked="" type="checkbox"/>	Text (plain)	Unlimited

Abbildung 54: Zwei Beispiel-Pfade aus der WissKI Konfiguration des ViSIT Projekts.

Der zweite Pfad in [Abbildung 54](#) ist nun ein wirklicher Pfad und steht high-level für den bezeichnenden Titel eines Ausstellungsobjekts. Die ID des Pfads ist "Object_identifiedBy_Title", während der Pfad aus zwei Knoten und einer Relation besteht:

- Da sich der Pfad bzw. der zugehörige Titel auf ein Ausstellungsobjekt beziehen soll, ist der erste Knoten von dem der Pfad ausgeht "<http://visit.de/ontologies/vismo/Object>".
- An diesen Knoten schließt sich dann die Relation "ecrm:P1_is_identified_by" an.
- Der Zielknoten dieses Pfads ist ein weiterer Knoten: "ecrm:E35_Title".

Was für den zweiten Pfad noch fehlt ist der zugehörige Datentyp, welcher in der Ansicht in [Abbildung 54](#) ebenfalls zu sehen ist: da ein Titel angegeben werden soll, definiert der Pfad eine Eigenschaft am Ende des Pfades (nicht in der Übersicht zu sehen) und benötigt damit einen primitiven Datentypen "text/plain". Editiert man diesen Pfad, öffnet sich die Maske, die in [Abbildung 55](#) zu sehen ist. Dort können viele Einstellungen zum Pfad editiert werden und zusätzlich ist die Angabe der letztendlichen Eigenschaft durch die Eingabe "Datatype Property" möglich. In diesem Beispiel ist der letzte Teil des Pfads somit "http://erlangen-crm.org/170309/P3_has_note".

Wird die aktuelle Einstellung des Pfads gespeichert, öffnet sich die zweite Maske zur Konfiguration eines WissKI Pfads. Ausgehend von dem aktuellen Beispiel eines Objekt-Titels ist dies in [Abbildung 56](#) zu sehen. Die ersten vier Einstellungen werden automatisch vom System gesetzt, und sollten in der Regel nicht angepasst werden müssen. Wichtig sind die vier unteren Einstellungen, welche den Datentypen des aktuell betrachteten Pfads definieren, indem zuerst der allgemeine Datentyp gesetzt wird und in den folgenden drei Einstellungen lässt sich einstellen, wie dieser Datentyp später in der Eingabemaske des WissKI Systems aussehen soll. In unserem Beispiel kann ein Text in einem einfachen Textfeld angelegt werden. Zusätzlich dazu kann die Kardinalität für das entsprechende Feld festgelegt werden.

Name *

Machine name: object_identifiedby_title

Path Type

Path

Is this Path a group?

► REASONER HAS RUN. CACHE IS PREPARED

STEP	EDIT
http://visit.de/ontologies/vismo/Object	-
http://erlangen-crm.org/170309/P1_is_identified_by	-
http://erlangen-crm.org/170309/E35_Title	-
please select	-

Datatype Property

http://erlangen-crm.org/170309/P3_has_note

Disambiguation Point

Concept 2: http://erlangen-crm.org/170309/E35_Title

Save

Abbildung 55: Erste Maske zum Editieren eines WissKI Pfads.

Abbildung 57 zeigt weiterhin ein Beispiel für einen Pfad, welcher eine Relation zwischen zwei Datenbankobjekten definiert. In diesem Falle handelt es sich semantisch um eine Beziehung zwischen Teilobjekten, also dass ein Objekt ein Teil eines zweiten Objekts ist. Um dies zu tun ist der Datentyp "Entity reference" nötig, und dass der Pfad in der entsprechend zu referenzierenden Klasse endet: wie in diesem Fall "http://visit.de/ontologies/vismo/Object".

Zwei weitere wichtige Punkte der WissKI Konfiguration sind in obigen Beispielen zu sehen:

ANORDNUNG DER PFADE Die Anordnung der Pfade spielt eine wichtige Rolle in der Konfiguration eines WissKI Systems. Dies ist auf der linken Seite in Abbildung 65 und der Übersicht der Pfade eines Pathbuilders zu sehen. Die Pfade können dort (durch drag&drop der "Kreuzchen" links neben einer Pfad-ID) in verschiedene Reihenfolgen bzw. Gruppierungen gebracht werden, um so mit die Zugehörigkeit eines Pfads zu einer Gruppe, bzw. sogar einer Gruppe zu einer Obergruppe zu definieren. Dies wird getan, indem ein Pfad "unter" und dann "eine Ebene nach rechts" geschoben wird, wie dies im Beispiel für die ersten beiden Pfade zu sehen ist (ebenfalls in Abbildung 54 zu sehen). Eine Untergruppe zum Objekt bildet zum Beispiel der Pfad mit der ID "Object_Dating", welche ihrerseits wieder vier Pfade unter sich zusammen führt. Dies ist wichtig, da das Setzen eines dieser Pfade nicht jeweils einen eigenen Zwischenknoten (mit dem RDF Typen "http://visit.de/ontologies/vismo/Dating") erzeugen soll, sondern alle vier Pfade den selben Zwischenknoten benutzen sollen.

DISAMBIGUIERUNG Die Disambiguierung bezieht sich - seicht formuliert - ähnlich wie die oben beschriebene Anordnung und Untergruppen indirekt auf die korrekte Zuweisung von Knoten zu ihren entsprechenden Objekten. Die Disambiguierung ist in den Pfaden durch die rot geschriebenen Teile zu sehen. Sie weist das hinterliegende System an, dass alles was ab diesem Punkt im Pfad definiert ist, einzigartig im System gespeichert werden soll. Dadurch können keine Duplikate mit genau der selben Konfiguration entstehen. Dies lässt sich einfach am obigen Beispiel erläutern: die Disambiguierung für den Objekttitle beginnt ab dem "ecrm:E35_Title" Knoten, der weiterhin nur die RDF Eigenschaft "http://erlangen-crm.org/170309/P3_has_note" besitzt. Durch die Disambiguierung an dieser Stelle wird das System keinen zweiten Knoten

Pathbuilder *
visittestrepo_paths
Name of the pathbuilder.

Path *
object_identifiedby_title
Name of the path.

CHOOSE FIELD

Titel (f961978b85ac335b73e52837f090be9d)
Select an existing field from bundle Object (b15d6693d3ba884f733c2cce4cd6b015)

f961978b85ac335b73e52837f090be9d
ID of the mapped Field.

Type of the field that should be generated.*
Text (plain)

Type for the Field (Textfield, Image, ...)

Type of form display for field
Textfield

Widget for the Field – If there is any.

Type of formatter for field
Plain text

Formatter for the field – If there is any.

Cardinality
Unlimited

Save **Delete**

Abbildung 56: Zweite Maske zum Editieren eines WissKI Pfads.



Abbildung 57: Beispiel-Pfad aus der WissKI Konfiguration des ViSIT Projekts für eine Relation zwischen zwei Entitäten der Datenbank.

erstellen, der den selben Titel in der Notiz-Eigenschaft besitzt. Damit wird sichergestellt, dass zum Beispiel keine zwei Objekte mit dem Titel “Mona Lisa” erstellt werden können. Technisch verweist das System im Hintergrund bei Verweis auf diesen Knoten somit immer auf genau diesen einen Knoten – wie beschrieben werden *keine* Duplikate davon erstellt. Die Disambiguierung eines Pfads kann in der Maske mit dem Feld “Disambiguation Point” vorgenommen werden, die in Abbildung 55 zu sehen ist.

Die Informationen und die Konfiguration des Pathbuilders wird vom WissKI System weiterhin genutzt, um Ein- sowie Ausgabeinterfaces für die Hauptobjekte des Pathbuilders zu erzeugen. Über die Create-Seite des WissKI Systems (zu sehen in Abbildung 66 im Appendix) ist unter anderem das Eingabe-Interface für die Ausstellungsobjekte (Object) zu erreichen, dieses ist in Abbildung 67 im Appendix zu sehen. Nachdem Objekte über dieses erstellt wurden, können diese per Navigate- und Find-Seite des WissKI Systems eingesehen werden. Ein Beispiel für das Ausgabeinterface einer Partisane aus dem ViSIT Projekt ist in Abbildung 68 im Appendix zu sehen.

10.4 Technischer Zugang zu den Metadaten – die ViSIT REST API

Das oben beschriebene WissKI System ist entworfen, um im ViSIT Kontext Zugang für Geisteswissenschaftler, Museumsmitarbeiter und allgemein forschenden Personen zu schaffen. Weiterhin war es im ViSIT Projekt aber ebenfalls nötig, einen technischen Zugang zu den über WissKI erzeugten Daten zu gewährleisten. Diesen technischen Zugang verwenden weiter verarbeitende Komponenten des Projekts, beschrieben in den Kapiteln 4 bis 8. Ziel ist es ebenfalls, die Informationen der RDF

Daten zu vermitteln, ohne dass der Empfänger mit RDF oder anderen semantischen Technologien in Berührung kommen muss.

Um dies auf standardisierte Weise durchzuführen, ist die Wahl auf eine serverseitige REST API gefallen, die mit den weiteren technischen Komponenten via HTTP kommunizieren kann. Die REST API ist in Java geschrieben und mit dem Spring Framework²⁹ umgesetzt. Die Entwicklung ist in einem Repository im allgemeinen ViSIT Projekt Bitbucket: <https://bitbucket.org/visit2016/metadb-rest-api/>. Eine direkte Kommunikation der aktuell installierten Version der REST API ist stets (sowohl auf dem produktiven Server als auch dem Testsystem) unter der URL Endung „.../metadb-rest-api/swagger-ui.html“ zu finden.

Dieses Kapitel beschreibt weiterhin die allgemeine Umsetzung der API, dessen Hintergründe, sowie technische Eigenheiten wie Datenmodelle usw. Jedoch sind die wichtigsten technischen Details zur Verwendung der ViSIT REST API in [Unterabschnitt 10.5](#) beschrieben. Dort werden unter anderem wichtige Skripte zur Inbetriebnahme der API erklärt, sowie der Prozess des Deployments erläutert.

Thematisch lässt sich die REST API in folgende Teile aufteilen, die unterschiedliche Aufgabenbereiche übernehmen:

DIGITAL REPRESENTATIONS: Die Digital Representations - zu deutsch digitale Repräsentationen - stellen die Verbindungen zwischen Metadaten und zugehörigen Mediendaten dar. Hierzu sind sie ein Eintrag in der Metadatenbank, die zu entsprechenden Entitäten hinzugefügt werden, und dabei verschiedene technische Details zu den Mediendaten beinhaltet. Die REST API bietet hierfür die Möglichkeiten, über HTTP entsprechende Digital Representations zu erzeugen, schreiben, bearbeiten, und zu löschen. Den Repräsentationen liegt folgendes (RDF) Modell, aufgezeigt in [Abbildung 58](#), zugrunde:

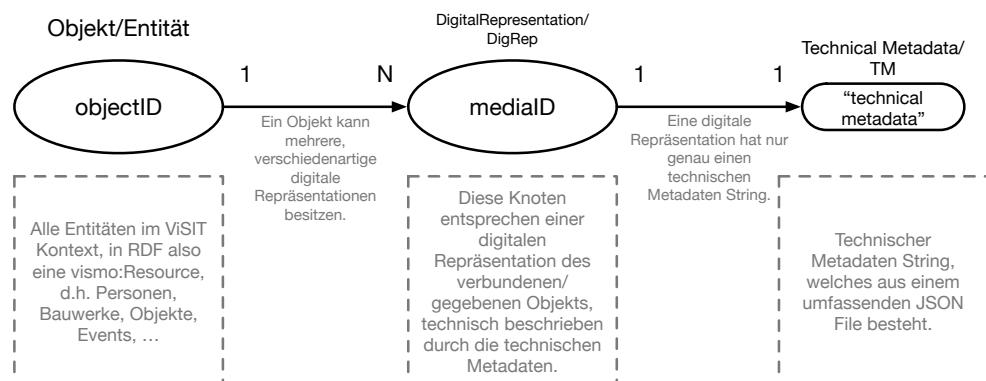


Abbildung 58: Modell der digitalen Repräsentationen.

Die möglichen API Funktionen sind, in [Tabelle 5](#) die folgenden (die API Pfade sind relativ zur jeweiligen REST API angegeben):

OBJEKTE: Dieser Bereich der API dient hauptsächlich zum Auslesen der Datenbankobjekte bzw. Entitäten, die über WissKI in der Metadatenbank angelegt wurden. Diese API Schnittstelle erfüllt die Anforderung, die bereits am Anfang dieses Unterkapitels beschrieben wurde: die Gewährleistung des technischen Auslesens der Metadaten in einem *nicht*-Semantic Web Format, in diesem Falle JSON. Dieses JSON beinhaltet alle Informationen, die entsprechend als RDF Metadaten auf der Datenbank vorhanden sind. [Listing 50](#) im Appendix gibt eine Art Schema für diesen Output an. Dort werden alle möglichen Felder mit ID, Datentyp und den potenziell referenzierten Typen einer Referenz angegeben. Zum Auslesen der Objekte bietet die API zwei mögliche API Funktionen aus [Tabelle 6](#):

²⁹ <https://spring.io/>

HTTP	API Pfad	Request Params	Kurzbeschreibung
GET	/digrep/media	id	Gibt TM String des mit der "id" spezifizierten DigRep Knotens zurück.
PUT	/digrep/media	id, newData	Updated den TM String durch "newData" des per "id" spezifizierten DigRep Knotens.
DELETE	/digrep/media	id	Löscht den durch die "id" spezifizierten DigRep Knoten.
GET	/digrep/object	id	Gibt alle TM Strings und deren DigRep Knoten IDs zurück.
POST	/digrep/object	id	Legt einen neuen DigRep Knoten für das Objekt mit der gegebenen "id" an.
DELETE	/digrep/object	mediaId, objectId	Löscht den DigRep Knoten mit der ID "mediaID", zugehörig zum Objekt mit der ID "objectId".

Tabelle 5: API Funktionen für die Digital Representations.

HTTP	API Pfad	Request Params	Kurzbeschreibung
GET	/object	id	Gibt die JSON Repräsentation des Objekts mit der ID "id" zurück.
GET	/wisskiobject	wisskipath	Gibt die JSON Repräsentation des Objekts zurück, welches dem WissKI View Path "wisskipath" entspricht.

Tabelle 6: API Funktionen zum Auslesen von Objekten.

Wichtig zu wissen beim Auslesen der Objekte ist, dass die angesprochene ID der API die RDF ID des Knotens der entsprechenden Entität ist (z.B. "<http://visit.de/data/5c4ae02a>" während der sogenannte "WissKI View Path" derjenige URL Pfad ist, der angezeigt wird, wenn eine Entität per WissKI Oberfläche betrachtet wird (z.B. "<https://database.visit.uni-passau.de/drupal/wisski/navigate/405/view>").

Weiterhin bietet dieser Bereich der API noch ein zusätzliches Quality of Life Feature: das Zurückgeben der Anzahl an vorhandenen Digital Representations und damit vorhandenen Mediendaten, bezogen auf ein Objekt. Dieses Objekt kann der API, wie beim Auslesen, per RDF ID oder per WissKI View Pfad übergeben werden.

(**UPLOAD UND DOWNLOAD FÜR EXCEL:)** TODO add?

Für die REST API ist eine Verschlüsselung umgesetzt, die dem Konzept der "basic authentication"³⁰ folgt. Mit dieser werden die sogenannten "unsicheren" Operationen der API, also diejenigen, die Daten verändern, erzeugen, und löschen dürfen, von unzulässigem Zugriff geschützt. Diese sind im ViSIT Kontext diejenigen Operationen, die sich auf die DigitalRepresentations beziehen. Alle rein lesenden Zugriffe auf die Metadatenbank sind gemäß dem Linked Open Data Gedanken offen nach aussen.

³⁰ Erklärt z.B. auf <https://swagger.io/docs/specification/authentication/basic-authentication/>

Für die gesicherten Zugänge der API sind auf dem ViSIT Server verschiedene "User" samt Passwort angelegt, welche auf die jeweils verarbeitenden Applikationen gemünzt werden, dementsprechend die Tablet und Fernrohr Apps, sowie die Kompressions-Komponente. Diese Liste an Usern und Passwort sind am entsprechenden Server im eigenen Bereich des "visit" Accounts zu finden und nur für entsprechende Administratoren zugänglich. Die Datei hat den Namen "users.csv" und muss **vor dem Start bzw. dem Deployment** der REST API vorhanden sein, damit diese darauf Zugriff hat.

10.5 Wichtige Technische Charakteristika der Entwicklung und den Betrieb der Semantischen Datenbank

In diesem Unterabschnitt werden wichtige technische Details zum Betrieb der ViSIT REST API erläutert. Diese sind wichtig für die Nutzung der API.

PYTHON SCRIPT ZUM GENERIEREN DER SPARQL QUERY TEMPLATES WICHTIG:
Das im Folgenden beschriebene Script muss **vor Deployment** der REST API ausgeführt werden.

Für den Betrieb der REST API, vor allem zum Auslesen der unterliegenden Metadaten, sind sogenannte SPARQL Query Templates von Nöten. Diese können als Vorlagen für die Abfrage aus der Metadatenbank verstanden werden. Diese Query Templates werden automatisch aufbauend auf dem in WissKI definierten Paths erstellt. So kann, wenn sich die WissKI Pfad Konfiguration ändert, das Script erneut angestoßen werden, um auf die Veränderung zu reagieren und damit die neu hinzugefügten Pfade mit in die Anfragen aufgenommen werden. Dieser Prozess ist in einem Phyton Script gekapselt, welches auf dem ViSIT Server ausgeführt werden kann.

Wichtigste Information die das Script benötigt sind die Pfad Konfigurationen des WissKI Systems. Diese können als Datei direkt im WissKI System, im Pathbuilder downgeloadet werden. [Abbildung 59](#) zeigt diese Funktionalität mit dem "Create Exportfile" Button.

Nach Betätigen des Buttons wird eine aktuelle Datei in die darüber stehende Liste aufgenommen. Durch Drücken dieser kann die Datei anschließend gespeichert werden.

Das Python Script befindet sich auf dem Server im Account "visit" im Ordner "python" und heißt "CreateSPARQLTemplatesFromPathsXML.py". **Die Pfad-Datei muss umbenannt werden zu "paths.xml" und muss sich im selben Ordner befinden.** Das Ausführen des Scripts kann mit folgendem Befehl ausgeführt werden:

```
1 | sudo python3 CreateSPARQLTemplatesFromPathsXML.py
```

Listing 46: Befehl zum Starten des Python Script zum Erstellen der Query Templates der ViSIT Metadatenbank.

Die Ergebnisse des Scripts werden automatisch in den Ordner "Templates" im Hauptordner des Accounts "visit" gespeichert, und automatisiert von der REST API benutzt.

EINSTELLUNGEN ZUM DEPLOYMENT DER REST API Wie oben bereits beschrieben, ist die REST API in einem Repository³¹ des ViSIT Haupt-Repositories angelegt.

In der Regel ist das entsprechende Projekt in einem "offline Teststatus" bezüglich seiner Konfiguration. Dies bedeutet, dass lokale Tests ausgeführt werden können, um die Funktionen der API lokal zu überprüfen. Sollte die REST API produktiv auf einem Server deployed werden, müssen folgende Änderungen in der Konfiguration durchgeführt werden:

³¹ <https://bitbucket.org/visit2016/metadb-rest-api/src/master/>

- Einbinden der springframework Dependency: In der “pom.xml” Maven Konfigurationsdatei des Projekts muss die Dependency für das Artefakt “spring-boot-starter-tomcat” auf den scope “provided” gesetzt werden. Die entsprechende Einstellung ist in Abbildung 60 zu sehen. Wichtig ist, dass Zeile 4 nicht kommentiert ist.
- Verbindung zum entsprechenden Triplestore: Weiterhin ist es nötig, die REST API mit dem Triplestore zu verbinden, mit dem das aktuell verwendete System, im speziellen das WissKI System, arbeitet. Diese Verbindung wird in der REST API in der Konfigurationsdatei “application.properties” gesetzt. Dabei muss der query und der update Endpunkt URLs des jeweiligen Triplestores angegeben werden. Im offline Modus sind die beiden URLs auf “none” gesetzt, während sie für den Produktiv-Modus der API auf den entsprechenden Triplestore eingestellt werden müssen. In Abbildung 61 ist die Konfiguration für den offline Modus zu sehen, während ebenfalls zwei vordefinierte URL Pärchen zu sehen sind, welche für den ViSIT Test- und Produktiv-Server stehen.

10.6 Semantische Datenbank – FAQ und häufig auftretende Probleme

Nachdem bis jetzt alle Details bezüglich der Metadatenbank und dessen Umgang erläutert wurden, führt dieses Unterkapitel eine Zusammenfassung von bekannten “Problemen” in Kombination mit Lösungen an, die im produktiven Betrieb der Datenbank entstehen können. Bei Auftreten eines dieser Probleme sollte die Datenbank durch Einsatz der aufgeführten Lösung wieder in einen funktionierenden Zustand überführt werden können.

WISSKI CACHE FLUSH Bei manchen Konfigurations-Änderungen kann es passieren, dass diese erst “online” geschaltet werden, wenn der Cache des Drupal Systems geleert wird. Dies ist zum Beispiel der Fall, wenn ein Thumbnail für ein Objekt in der Metadatenbank hochgeladen wird. Der produktive Server ist so konfiguriert, dass einmal am Tag ein solcher Cache Flush durchgeführt wird. Sollten die jeweiligen Informationen durch die Konfiguration jedoch sofort benötigt werden, kann ein Cache Flush auch per Hand am Server ausgeführt werden. Dazu muss in das Verzeichnis der Drupal Installation gewechselt werden (`/var/www/html/drupal`) und folgender Befehl ausgeführt werden:

```
| drush cr
```

Listing 47: Befehl für einen Cache Flush eines Drupal Systems.

Sollte dieser Befehl aus irgendwelchen Gründen nicht funktionieren, kann die selbe Funktionalität über andere Wege durchgeführt werden. Hierzu folgender Link: <https://www.drupal.org/docs/7/administering-drupal-7-site/clearing-or-rebuilding-drupals-cache>

RECOMPUTE HIERARCHY IN WISSKI Wenn die dem WissKI System unterliegende Ontologie ausgetauscht oder upgedated wird, kann es sein, dass das System über diese Ontologie eine neue Hierarchie generieren muss. Das neue Berechnen der Ontologie kann in der Übersicht des SALZ Adapters (siehe Unterabschnitt 10.3, Funktionalität ganz unten aufklappbar) angestoßen werden.

UPDATES VON DRUPAL, MAINTENANCE MODE Von Zeit zu Zeit wird eine neue Version von Drupal released. Dies ist für die Metadatenbank theoretisch irrelevant, wenn es nur ein Minor Update ist (diese sollten aber im Bezug auf aktuelle Software trotzdem eingespielt werden). Ein Major Update führt in der Regel jedoch dazu, dass das Drupal System in den Maintenance Mode versetzt wird - ein Status in dem das System aus Sicherheitsgründen offline geschaltet wird. Deswegen ist hier ein Update durchzuführen.

Updates des Drupal Systems können (in seltenen Fällen) entweder im Drupal Interface selbst durchgeführt werden (Reiter “Extend”, dann auf Link für “available updates” drücken). In der Regel führt man die Updates (auch von anderen Modulen) direkt am Server durch. Dafür in das Verzeichnis der Drupal Installation wechseln (`/var/www/html/drupal`) und folgenden Befehl ausführen:

```
| sudo composer update --with-all-dependencies
```

Listing 48: Befehl zum Update einer Drupal Installation.

Möglicherweise müssen im Konflikt stehende Abhängigkeiten von Hand gelöst werden. Dazu Schritt für Schritt die einzelnen Dependencies updaten und am Ende nochmals obigen Befehl ausführen.

RDF4J WORKBENCH ACCESS Um direkt auf dem unterliegenden Triplestore zu arbeiten, kann auf die RDF4J Workbench zugegriffen werden (URL: <https://database.visit.uni-passau.de/rdf4j-workbench>). In manchen Fällen leitet das Aufrufen der Homepage zu einem Prompt, der nach der URL, sowie Benutzer und Passwort verlangt. Für den Server ist kein User angelegt, deswegen können die beiden Felder für Benutzer und Passwort leer gelassen werden. Jedoch ist manchmal die angegebene URL inkorrekt, da “http” anstatt “https” in der URL steht. Dieses muss angepasst werden, dann kann auf den Triplestore zugegriffen werden.

MAVEN ABHÄNGIGKEITEN IM REST API PROJEKT Im Java Projekt für die ViSIT REST API passiert ab und an der Fall, dass die Abhängigkeiten des Projekts nicht richtig interpretiert werden, was dazu führt, dass einige Packages in den implementierten Klassen als “nicht vorhanden” angezeigt werden. Durch ein erneutes Downloaden der Abhängigkeiten (durch das Kommando “Reimport” oder falls dies nicht hilft “Update Maven Indices” in der obersten pom.xml des Projekts) sollte sich dieses Problem lösen lassen.

visittestrepo_pathes M

Name of the Pathbuilder-Tree.

visittestrepo ▼

Which adapter does this Pathbuilder belong to?

IMPORT TEMPLATES

Pathbuilder Definition Import

Path to a pathbuilder definition file.

Set default mode to

Keep settings from import file ▼

What should the fields and groups mode be set to?

Import

EXPORT TEMPLATES

- rdf4jpathbuildertest_20180315T204613
- visittestrepo_pathes_20180808T170845
- visittestrepo_pathes_20180905T182510
- visittestrepo_pathes_20181128T104132
- visittestrepo_pathes_20190107T110029
- visittestrepo_pathes_20190109T143959
- visittestrepo_pathes_20190123T151843
- visittestrepo_pathes_20190131T143855

visittestrepo_20180320T05420

Create Exportfile

Save and generate bundles and fields ▼ [Delete](#)

Abbildung 59: WissKI Pathbuilder Ansicht, die den Download der Pfad-Datei anbietet.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-tomcat</artifactId>
    <scope>provided</scope>
    <!-- Uncomment above setting to produce a productive instance -->
</dependency>
```

Abbildung 60: pom.xml Konfiguration zum produktiven Deployment der REST API.

```
# -----
# Properties for the Anno4j, whether to connect or not connect to the database
# -----
# Use these to connect to the PRODUCTIVE DB
#visit.rest.sparql.endpoint.query=https://database.visit.uni-passau.de/rdf4j-server/repositories/visittestrepo
#visit.rest.sparql.endpoint.update=https://database.visit.uni-passau.de/rdf4j-server/repositories/visittestrepo/statements

# Use these to connect to the test DB
#visit.rest.sparql.endpoint.query=https://database-test.visit.uni-passau.de/rdf4j-server/repositories/visittestrepo
#visit.rest.sparql.endpoint.update=https://database-test.visit.uni-passau.de/rdf4j-server/repositories/visittestrepo/statements

# Use these to NOT connect to the DB
visit.rest.sparql.endpoint.query=none
visit.rest.sparql.endpoint.update=none
```

Abbildung 61: Einstellungen zum Verbinden des Triplestores. Hier gezeigt: lokale Konfiguration, während die Konfiguration für Produktiv- und Testsystem oben vorhanden aber auskommentiert ist.

11 SCHLUSS

blub

12 APPENDIX

```

1 <?xml version="1.0"?>
2 <rdf:RDF xmlns="http://visit.de/ontologies/vismo/"
3   xmlns:base="http://visit.de/ontologies/vismo/"
4   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
5   xmlns:ns="http://www.w3.org/2003/06/sw-vocab-status/ns#"
6   xmlns:owl="http://www.w3.org/2002/07/owl#"
7   xmlns:xml="http://www.w3.org/XML/1998/namespace"
8   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
9   xmlns:skos="http://www.w3.org/2004/02/skos/core#"
10  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
11  xmlns:wot="http://xmlns.com/wot/0.1/"
12  xmlns:foaf="http://xmlns.com/foaf/0.1/"
13  xmlns:dc="http://purl.org/dc/elements/1.1/"/>
14 <owl:Ontology rdf:about="http://visit.de/ontologies/vismo/">
15   <owl:versionIRI rdf:resource="http://visit.de/ontologies/vismo/0.4.5/" />
16   <owl:imports rdf:resource="http://erlangen-crm.org/170309/" />
17   <owl:imports rdf:resource="http://xmlns.com/foaf/0.1/" />
18 </owl:Ontology>
19
20
21
22 <!--
23 //////////////////////////////////////////////////////////////////
24 //
25 // Object Properties
26 //
27 //////////////////////////////////////////////////////////////////
28 -->
29
30
31
32 <!-- http://visit.de/ontologies/vismo/containsEntry -->
33
34 <owl:ObjectProperty rdf:about="http://visit.de/ontologies/vismo/containsEntry">
35   <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topObjectProperty"/>
36   <owl:inverseOf rdf:resource="http://visit.de/ontologies/vismo/isEntryIn"/>
37   <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Reference"/>
38   <rdfs:range rdf:resource="http://visit.de/ontologies/vismo/ReferenceEntry"/>
39   <rdfs:comment>Reference from a vismo:Reference to a contained vismo:ReferenceEntry.</
40   rdfs:comment>
41   <rdfs:label>contains entry</rdfs:label>
42 </owl:ObjectProperty>
43
44
45 <!-- http://visit.de/ontologies/vismo/employsTraderoute -->
46
47 <owl:ObjectProperty rdf:about="http://visit.de/ontologies/vismo/employsTraderoute">
48   <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topObjectProperty"/>
49   <owl:inverseOf rdf:resource="http://visit.de/ontologies/vismo/forTrade"/>
50   <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Trade"/>
51   <rdfs:range rdf:resource="http://visit.de/ontologies/vismo/Traderoute"/>
52   <rdfs:comment>Refers from a vismo:Trade to the vismo:TradeRoute that the trade is fulfilled on
53 .</rdfs:comment>
54   <rdfs:label>employs traderoute</rdfs:label>
55 </owl:ObjectProperty>
56
57
58 <!-- http://visit.de/ontologies/vismo/endLocation -->
59
60 <owl:ObjectProperty rdf:about="http://visit.de/ontologies/vismo/endLocation">
61   <rdfs:subPropertyOf rdf:resource="http://visit.de/ontologies/vismo/routeLocation"/>
62   <rdfs:comment>Refers from a traderoute to the vismo:City that represents the ending point for
63 the route.</rdfs:comment>
64   <rdfs:label>end location</rdfs:label>
65 </owl:ObjectProperty>
66
67
68 <!-- http://visit.de/ontologies/vismo/entryIsAbout -->
69
70 <owl:ObjectProperty rdf:about="http://visit.de/ontologies/vismo/entryIsAbout">
71   <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topObjectProperty"/>
72   <owl:inverseOf rdf:resource="http://visit.de/ontologies/vismo/referencedByEntry"/>
73   <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/ReferenceEntry"/>
74   <rdfs:range rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
75   <rdfs:comment>Reference from a vismo:ReferenceEntry to a vismo:Resource, associating the
76 describing nature of the associated vismo:Reference.</rdfs:comment>
```

```

77      <rdfs:label>entry is about</rdfs:label>
78  </owl:ObjectProperty>
79
80
81
82  <!-- http://visit.de/ontologies/vismo/forTrade -->
83
84  <owl:ObjectProperty rdf:about="http://visit.de/ontologies/vismo/forTrade">
85    <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topObjectProperty"/>
86    <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Traderoute"/>
87    <rdfs:range rdf:resource="http://visit.de/ontologies/vismo/Trade"/>
88    <rdfs:comment>Refers from a vismo:TradeRoute to the vismo:Trade resource that illustrates the
89    trading on the given route.</rdfs:comment>
90    <rdfs:label>for trade</rdfs:label>
91  </owl:ObjectProperty>
92
93
94  <!-- http://visit.de/ontologies/vismo/hasDigitalRepresentation -->
95
96  <owl:ObjectProperty rdf:about="http://visit.de/ontologies/vismo/hasDigitalRepresentation">
97    <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topObjectProperty"/>
98    <owl:inverseOf rdf:resource="http://visit.de/ontologies/vismo/representsDigitally"/>
99    <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
100   <rdfs:range rdf:resource="http://visit.de/ontologies/vismo/DigitalRepresentation"/>
101   <rdfs:comment>Links from a vismo:Resource (so an object that can be further specified in the
102   ViSIT context) to a digital representation of it, e.g. a picture that shows the respective
103   resource, a 3D model, etc.</rdfs:comment>
104   <rdfs:label>has digital representation</rdfs:label>
105 </owl:ObjectProperty>
106
107
108  <!-- http://visit.de/ontologies/vismo/interactionSource -->
109
110 <owl:ObjectProperty rdf:about="http://visit.de/ontologies/vismo/interactionSource">
111   <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topObjectProperty"/>
112   <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/MiscellaneousInteraction"/>
113   <rdfs:range rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
114   <rdfs:label>interaction source</rdfs:label>
115 </owl:ObjectProperty>
116
117
118  <!-- http://visit.de/ontologies/vismo/interactionTarget -->
119
120 <owl:ObjectProperty rdf:about="http://visit.de/ontologies/vismo/interactionTarget">
121   <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topObjectProperty"/>
122   <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
123   <rdfs:range rdf:resource="http://visit.de/ontologies/vismo/MiscellaneousInteraction"/>
124   <rdfs:label>interaction target</rdfs:label>
125 </owl:ObjectProperty>
126
127
128
129  <!-- http://visit.de/ontologies/vismo/interstation -->
130
131 <owl:ObjectProperty rdf:about="http://visit.de/ontologies/vismo/interstation">
132   <rdfs:subPropertyOf rdf:resource="http://visit.de/ontologies/vismo/routeLocation"/>
133   <rdfs:comment>Refers from a traderoute to the vismo:City that represents a interstation for the
134   route.</rdfs:comment>
135   <rdfs:label>interstation</rdfs:label>
136 </owl:ObjectProperty>
137
138
139  <!-- http://visit.de/ontologies/vismo/isEntryIn -->
140
141 <owl:ObjectProperty rdf:about="http://visit.de/ontologies/vismo/isEntryIn">
142   <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topObjectProperty"/>
143   <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/ReferenceEntry"/>
144   <rdfs:range rdf:resource="http://visit.de/ontologies/vismo/Reference"/>
145   <rdfs:comment>Reference from a vismo:ReferenceEntry to its encompassing vismo:Resource.</
146   rdfs:comment>
147   <rdfs:label>is entry in</rdfs:label>
148 </owl:ObjectProperty>
149
150
151  <!-- http://visit.de/ontologies/vismo/partOfTradeRoute -->
152
153 <owl:ObjectProperty rdf:about="http://visit.de/ontologies/vismo/partOfTradeRoute">
154   <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topObjectProperty"/>
155   <owl:inverseOf rdf:resource="http://visit.de/ontologies/vismo/routeLocation"/>
```

```

156      <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Place"/>
157      <rdfs:range rdf:resource="http://visit.de/ontologies/vismo/Traderoute"/>
158      <rdfs:comment>Refers from a vismo:City to a/multiple vismo:TradeRoute resource, indicating the
159      given city is part of a trade route and therefore its associated trade.</rdfs:comment>
160      <rdfs:label>part of trade route</rdfs:label>
161      </owl:ObjectProperty>

162
163
164      <!-- http://visit.de/ontologies/vismo/reference -->
165
166      <owl:ObjectProperty rdf:about="http://visit.de/ontologies/vismo/reference">
167          <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topObjectProperty"/>
168          <owl:inverseOf rdf:resource="http://visit.de/ontologies/vismo/referencedBy"/>
169          <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
170          <rdfs:range rdf:resource="http://visit.de/ontologies/vismo/Reference"/>
171          <rdfs:comment>Issues that the referenced vismo:Reference contains further and descriptive
172          information about the given vismo:Resource entity.</rdfs:comment>
173          <rdfs:label>reference</rdfs:label>
174      </owl:ObjectProperty>

175
176
177      <!-- http://visit.de/ontologies/vismo/referencedBy -->
178
179      <owl:ObjectProperty rdf:about="http://visit.de/ontologies/vismo/referencedBy">
180          <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topObjectProperty"/>
181          <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Reference"/>
182          <rdfs:range rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
183          <rdfs:comment>Refers to the vismo:Resource entities that reference this vismo:Reference and
184          therefore this entity contains further and descriptive information about the resource entities.</
185          rdfs:comment>
186          <rdfs:label>referenced by</rdfs:label>
187      </owl:ObjectProperty>

188
189      <!-- http://visit.de/ontologies/vismo/referencedByEntry -->
190
191      <owl:ObjectProperty rdf:about="http://visit.de/ontologies/vismo/referencedByEntry">
192          <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topObjectProperty"/>
193          <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
194          <rdfs:range rdf:resource="http://visit.de/ontologies/vismo/ReferenceEntry"/>
195          <rdfs:comment>Reference from a vismo:Resource to a given vismo:ReferenceEntry, indicating that
196          the associated vismo:Reference contains information about the former.</rdfs:comment>
197          <rdfs:label>referenced by entry</rdfs:label>
198      </owl:ObjectProperty>

199
200
201      <!-- http://visit.de/ontologies/vismo/representsDigitally -->
202
203      <owl:ObjectProperty rdf:about="http://visit.de/ontologies/vismo/representsDigitally">
204          <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topObjectProperty"/>
205          <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/DigitalRepresentation"/>
206          <rdfs:range rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
207          <rdfs:comment>Refers from a given digital representation (a picture, 3D model, etc.) back to
208          the vismo:Resource that it originally represents.</rdfs:comment>
209          <rdfs:label>represents digitally</rdfs:label>
210      </owl:ObjectProperty>

211
212
213      <!-- http://visit.de/ontologies/vismo/routeLocation -->
214
215      <owl:ObjectProperty rdf:about="http://visit.de/ontologies/vismo/routeLocation">
216          <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topObjectProperty"/>
217          <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Traderoute"/>
218          <rdfs:range rdf:resource="http://visit.de/ontologies/vismo/Place"/>
219          <rdfs:comment>Refers (with different sub-properties) from a vismo:TradeRoute to a vismo:City
220          that is located on said route.</rdfs:comment>
221          <rdfs:label>route location</rdfs:label>
222      </owl:ObjectProperty>

223
224
225      <!-- http://visit.de/ontologies/vismo/startLocation -->
226
227      <owl:ObjectProperty rdf:about="http://visit.de/ontologies/vismo/startLocation">
228          <rdfs:subPropertyOf rdf:resource="http://visit.de/ontologies/vismo/routeLocation"/>
229          <rdfs:comment>Refers from a traderoute to the vismo:City that represents the starting point for
230          the route.</rdfs:comment>
231          <rdfs:label>start location</rdfs:label>
232      </owl:ObjectProperty>
```

```

232
233
234
235 <!--
236 ///////////////////////////////////////////////////
237 //
238 // Data properties
239 //
240 ///////////////////////////////////////////////////
241 -->
242
243
244
245
246 <!-- http://visit.de/ontologies/vismo/buildingHistory -->
247
248 <owl:DatatypeProperty rdf:about="http://visit.de/ontologies/vismo/buildingHistory">
249   <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topDataProperty"/>
250   <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Architecture"/>
251   <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
252   <rdfs:comment>Property used to (freely) describe the building history of a vismo:Architecture
253   entity.</rdfs:comment>
254   <rdfs:label>building history</rdfs:label>
255 </owl:DatatypeProperty>
256
257
258 <!-- http://visit.de/ontologies/vismo/comment -->
259
260 <owl:DatatypeProperty rdf:about="http://visit.de/ontologies/vismo/comment">
261   <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topDataProperty"/>
262   <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
263   <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
264   <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">This is a comment about a
265   given vismo entity.</rdfs:comment>
266   <rdfs:isDefinedBy rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://visit.de/
267   ontologies/vismo</rdfs:isDefinedBy>
268   <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">comment</rdfs:label>
269 </owl:DatatypeProperty>
270
271 <!-- http://visit.de/ontologies/vismo/description -->
272
273 <owl:DatatypeProperty rdf:about="http://visit.de/ontologies/vismo/description">
274   <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topDataProperty"/>
275   <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
276   <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
277   <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">This property defines a (historical)
278   description for a vismo entity.</rdfs:comment>
279   <rdfs:isDefinedBy rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://visit.de/
280   ontologies/vismo</rdfs:isDefinedBy>
281   <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">description</rdfs:label>
282 </owl:DatatypeProperty>
283
284 <!-- http://visit.de/ontologies/vismo/entryPages -->
285
286 <owl:DatatypeProperty rdf:about="http://visit.de/ontologies/vismo/entryPages">
287   <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topDataProperty"/>
288   <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/ReferenceEntry"/>
289   <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
290   <rdfs:comment>The range of pages that a given vismo:ReferenceEntry references of a
291   vismo:Reference entity.</rdfs:comment>
292   <rdfs:label>entry pages</rdfs:label>
293 </owl:DatatypeProperty>
294
295
296 <!-- http://visit.de/ontologies/vismo/helpfulLinks -->
297
298 <owl:DatatypeProperty rdf:about="http://visit.de/ontologies/vismo/helpfulLinks">
299   <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topDataProperty"/>
300   <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
301   <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
302   <rdfs:comment>This property is used to conveniently collect links to online resources that
303   contain further information of the associated vismo:Resource.</rdfs:comment>
304   <rdfs:label>helpful links</rdfs:label>
305 </owl:DatatypeProperty>
306
307
308 <!-- http://visit.de/ontologies/vismo/iconography -->

```

```

309 <owl:DatatypeProperty rdf:about="http://visit.de/ontologies/vismo/iconography">
310   <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topDataProperty"/>
311   <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
312   <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
313   <rdfs:comment>A property to associate iconography to a given vismo:Resource entity.</
314   rdfs:comment>
315   <rdfs:label>iconography</rdfs:label>
316 </owl:DatatypeProperty>
317
318
319
320 <!-- http://visit.de/ontologies/vismo/innerDescription -->
321
322 <owl:DatatypeProperty rdf:about="http://visit.de/ontologies/vismo/innerDescription">
323   <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topDataProperty"/>
324   <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Architecture"/>
325   <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
326   <rdfs:comment>Property used to describe the interior of a vismo:Architecture entity.</
327   rdfs:comment>
328   <rdfs:label>inner description</rdfs:label>
329 </owl:DatatypeProperty>
330
331
332 <!-- http://visit.de/ontologies/vismo/keyword -->
333
334 <owl:DatatypeProperty rdf:about="http://visit.de/ontologies/vismo/keyword">
335   <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topDataProperty"/>
336   <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
337   <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
338   <rdfs:comment>This property is used to address keywords for a vismo:Resource entity. These
339   refer to more general topics that can be addressed to anything out of the VisMo domain, for
340   example &quot;Trade&quot;, &quot;War&quot;/&quot;Peace&quot;, or overall temporal associations.</
341   rdfs:comment>
342   <rdfs:label>keyword</rdfs:label>
343 </owl:DatatypeProperty>
344
345
346 <!-- http://visit.de/ontologies/vismo/literature -->
347
348 <owl:DatatypeProperty rdf:about="http://visit.de/ontologies/vismo/literature">
349   <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topDataProperty"/>
350   <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
351   <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
352   <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">This property defines a
353   literature entry that contains further information about the given vismo entity.</rdfs:comment>
354   <rdfs:isDefinedBy rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://visit.de/
355   ontologies/vismo/</rdfs:isDefinedBy>
356   <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">literature</rdfs:label>
357 </owl:DatatypeProperty>
358
359
360 <!-- http://visit.de/ontologies/vismo/outerDescription -->
361
362 <owl:DatatypeProperty rdf:about="http://visit.de/ontologies/vismo/outerDescription">
363   <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topDataProperty"/>
364   <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Architecture"/>
365   <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
366   <rdfs:comment>Property used to describe the exterior of a vismo:Architecture entity.</
367   rdfs:comment>
368   <rdfs:label>outer description</rdfs:label>
369 </owl:DatatypeProperty>
370
371
372 <!-- http://visit.de/ontologies/vismo/pages -->
373
374 <owl:DatatypeProperty rdf:about="http://visit.de/ontologies/vismo/pages">
375   <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topDataProperty"/>
376   <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Reference"/>
377   <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
378   <rdfs:comment>Number of pages of a given vismo:Reference entity.</rdfs:comment>
379   <rdfs:label>pages</rdfs:label>
380 </owl:DatatypeProperty>
381
382
383 <!-- http://visit.de/ontologies/vismo/publisher -->
384
385 <owl:DatatypeProperty rdf:about="http://visit.de/ontologies/vismo/publisher">
386   <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topDataProperty"/>

```



```

465 //////////////////////////////////////////////////////////////////
466 -->
467
468
469
470
471 <!-- http://visit.de/ontologies/vismo/Activity -->
472
473 <owl:Class rdf:about="http://visit.de/ontologies/vismo/Activity">
474   <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E7_Activity"/>
475   <rdfs:subClassOf rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
476   <rdfs:comment>Activities in the ViSIT context are any type of timely historical event that can
477   contribute a timely frame for associated ViSIT concepts. For example "World War II", &
478   <rdfs:label>Activity</rdfs:label>
479 </owl:Class>
480
481
482 <!-- http://visit.de/ontologies/vismo/Architecture -->
483
484 <owl:Class rdf:about="http://visit.de/ontologies/vismo/Architecture">
485   <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E53_Place"/>
486   <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E84_Information_Carrier"/>
487   <rdfs:subClassOf rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
488   <rdfs:comment>Architecture in the ViSIT context describes every building or architectural
489   production that has been erected by mankind in some way.</rdfs:comment>
490   <rdfs:label>Architecture</rdfs:label>
491 </owl:Class>
492
493
494 <!-- http://visit.de/ontologies/vismo/BishopricAffiliation -->
495
496 <owl:Class rdf:about="http://visit.de/ontologies/vismo/BishopricAffiliation">
497   <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E55_Type"/>
498   <rdfs:comment>This class comprises headwords for bishopric affiliations for vismo:Architecture
499   resources.</rdfs:comment>
500   <rdfs:label>Bishopric Affiliation</rdfs:label>
501 </owl:Class>
502
503
504 <!-- http://visit.de/ontologies/vismo/Country -->
505
506 <owl:Class rdf:about="http://visit.de/ontologies/vismo/Country">
507   <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E53_Place"/>
508   <rdfs:subClassOf rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
509   <rdfs:label>Country</rdfs:label>
510 </owl:Class>
511
512
513
514 <!-- http://visit.de/ontologies/vismo/Dating -->
515
516 <owl:Class rdf:about="http://visit.de/ontologies/vismo/Dating">
517   <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E52_Time-Span"/>
518   <rdfs:comment>More specific class of the E52_TimeSpan and used in the ViSIT context to give
519   temporal associations with various entities.</rdfs:comment>
520   <rdfs:label>Dating</rdfs:label>
521 </owl:Class>
522
523
524
525 <!-- http://visit.de/ontologies/vismo/Description -->
526
527 <owl:Class rdf:about="http://visit.de/ontologies/vismo/Description">
528   <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E55_Type"/>
529   <rdfs:comment>Descriptions comprise characteristic types for objects in the domain of museums.
530   Therefore these are for example "painting", "oil painting", "chest",
531   etc.</rdfs:comment>
532   <rdfs:label>Description</rdfs:label>
533 </owl:Class>
534
535
536 <!-- http://visit.de/ontologies/vismo/DigitalRepresentation -->
537
538 <owl:Class rdf:about="http://visit.de/ontologies/vismo/DigitalRepresentation">
539   <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
      <rdfs:comment>A digital representation symbolises a multimedia representation of a
      vismo:Resource that can be illustrated in some way. These incorporate pictures, videos, audio
      files, and 3D models in particular.</rdfs:comment>

```

```

540     <rdfs:label>Digital Representation</rdfs:label>
541 </owl:Class>
542
543
544
545 <!-- http://visit.de/ontologies/vismo/Function -->
546
547 <owl:Class rdf:about="http://visit.de/ontologies/vismo/Function">
548     <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E55_Type"/>
549     <rdfs:comment>Functions relate to semantical and functional properties that are inherited by
vismo:Object as well as vismo:Architecture and their vismo:Structural Evolution resources. Both
an object as well as an architecture could exert "military" functions.</rdfs:comment>
550     <rdfs:label>Function</rdfs:label>
551 </owl:Class>
552
553
554
555 <!-- http://visit.de/ontologies/vismo/GeographicalAffiliation -->
556
557 <owl:Class rdf:about="http://visit.de/ontologies/vismo/GeographicalAffiliation">
558     <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E55_Type"/>
559     <rdfs:comment>This class comprises headwords for geographical affiliations for
vismo:Architecture resources.</rdfs:comment>
560     <rdfs:label>Geographical Affiliation</rdfs:label>
561 </owl:Class>
562
563
564 <!-- http://visit.de/ontologies/vismo/Group -->
565
566 <owl:Class rdf:about="http://visit.de/ontologies/vismo/Group">
567     <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E74_Group"/>
568     <rdfs:subClassOf rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
569     <rdfs:comment>This more general class will comprise different groups of people that are
associated in the VisMo context. In the first instance these are WorkingGroups (Werkst\at\ten) and joint
practices (Soziet\at\ten).</rdfs:comment>
570     <rdfs:label>Group</rdfs:label>
571 </owl:Class>
572
573
574
575 <!-- http://visit.de/ontologies/vismo/GroupDescription -->
576
577 <owl:Class rdf:about="http://visit.de/ontologies/vismo/GroupDescription">
578     <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E55_Type"/>
579     <rdfs:comment>This Class comprises descriptive types for all kinds of groups that are
associated in the VisMo context, such as Werkstatt and Soziet\at\ten.</rdfs:comment>
580     <rdfs:label>Group Description</rdfs:label>
581 </owl:Class>
582
583
584
585 <!-- http://visit.de/ontologies/vismo/HistoricalChange -->
586
587 <owl:Class rdf:about="http://visit.de/ontologies/vismo/HistoricalChange">
588     <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E9_Move"/>
589     <rdfs:subClassOf rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
590     <rdfs:label>HistoricalChange</rdfs:label>
591 </owl:Class>
592
593
594
595 <!-- http://visit.de/ontologies/vismo/InscriptionType -->
596
597 <owl:Class rdf:about="http://visit.de/ontologies/vismo/InscriptionType">
598     <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E55_Type"/>
599     <rdfs:comment>This E55_Type describes the type of an Inscription, done on various vismo:Object
entities.</rdfs:comment>
600     <rdfs:label>Inscription Type</rdfs:label>
601 </owl:Class>
602
603
604
605 <!-- http://visit.de/ontologies/vismo/Institution -->
606
607 <owl:Class rdf:about="http://visit.de/ontologies/vismo/Institution">
608     <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E53_Place"/>
609     <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E74_Group"/>
610     <rdfs:subClassOf rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
611     <rdfs:comment>An institution in the ViSIT context is primarily used for museums, which inherit
both the properties of a E53_Place as well as a E74_Group. This is necessary to make instances of
this class be able to represent a spatial entity as well as an entity that can for example hold
vismo:Objects.</rdfs:comment>
612     <rdfs:label>Institution</rdfs:label>
613

```

```

614 </owl:Class>
615
616
617
618 <!-- http://visit.de/ontologies/vismo/Marriage -->
619
620 <owl:Class rdf:about="http://visit.de/ontologies/vismo/Marriage">
621   <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E74_Group"/>
622   <rdfs:subClassOf rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
623   <rdfs:comment>A Subclass of the E74_Group in order to differentiate the participation of a
vismo:Person in a Marriage rather than any other vismo:Group.</rdfs:comment>
624   <rdfs:label>Marriage</rdfs:label>
625 </owl:Class>
626
627
628
629 <!-- http://visit.de/ontologies/vismo/MiscellaneousInteraction -->
630
631 <owl:Class rdf:about="http://visit.de/ontologies/vismo/MiscellaneousInteraction">
632   <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E7_Activity"/>
633   <rdfs:subClassOf rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
634   <rdfs:comment>This E7_Activity subclass is used for various dynamic interactions between
vismo:Resource objects, whose interaction is not yet known or more specifically not defined by
the ontology model.</rdfs:comment>
635   <rdfs:label>MiscellaneousInteraction</rdfs:label>
636 </owl:Class>
637
638
639
640 <!-- http://visit.de/ontologies/vismo/Mounting -->
641
642 <owl:Class rdf:about="http://visit.de/ontologies/vismo/Mounting">
643   <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E55_Type"/>
644   <rdfs:comment>This class comprises the various possibilities of fix/mount/place an Inscription
onto a vismo:Object.</rdfs:comment>
645   <rdfs:label>Mounting</rdfs:label>
646 </owl:Class>
647
648
649
650 <!-- http://visit.de/ontologies/vismo/Object -->
651
652 <owl:Class rdf:about="http://visit.de/ontologies/vismo/Object">
653   <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E84_Information_Carrier"/>
654   <rdfs:subClassOf rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
655   <rdfs:comment>Objects in the VisIT context subsume all sorts of items that are displayed in a
museum.</rdfs:comment>
656   <rdfs:label>Object</rdfs:label>
657 </owl:Class>
658
659
660
661 <!-- http://visit.de/ontologies/vismo/OrderAffiliation -->
662
663 <owl:Class rdf:about="http://visit.de/ontologies/vismo/OrderAffiliation">
664   <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E55_Type"/>
665   <rdfs:comment>This class comprises headwords for order affiliations for vismo:Architecture
resources.</rdfs:comment>
666   <rdfs:label>Order Affiliation</rdfs:label>
667 </owl:Class>
668
669
670
671 <!-- http://visit.de/ontologies/vismo/Person -->
672
673 <owl:Class rdf:about="http://visit.de/ontologies/vismo/Person">
674   <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E21_Person"/>
675   <rdfs:subClassOf rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
676   <rdfs:subClassOf rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
677   <rdfs:label>Person</rdfs:label>
678 </owl:Class>
679
680
681
682 <!-- http://visit.de/ontologies/vismo/Place -->
683
684 <owl:Class rdf:about="http://visit.de/ontologies/vismo/Place">
685   <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E53_Place"/>
686   <rdfs:subClassOf rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
687   <rdfs:comment>All cities, towns, settlements etc. of some sort are subsumed under this class.</
rdfs:comment>
688   <rdfs:label>Place</rdfs:label>
689 </owl:Class>
690

```

```

691
692
693      <!-- http://visit.de/ontologies/vismo/Profession -->
694
695      <owl:Class rdf:about="http://visit.de/ontologies/vismo/Profession">
696          <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E55_Type"/>
697          <rdfs:comment>Professions subsume all roles, employments, titles, authorities, etc. for persons
698          that are inherent in the cultural heritage domain.</rdfs:comment>
699          <rdfs:label>Profession</rdfs:label>
700      </owl:Class>
701
702
703      <!-- http://visit.de/ontologies/vismo/Reference -->
704
705      <owl:Class rdf:about="http://visit.de/ontologies/vismo/Reference">
706          <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E84_Information_Carrier"/>
707          <rdfs:comment>Used in the cultural use case of Visit as a reference to various textual
708          information objects that contained further and descriptive information about a given Visit
709          resource.</rdfs:comment>
710          <rdfs:label>Reference</rdfs:label>
711      </owl:Class>
712
713
714      <!-- http://visit.de/ontologies/vismo/ReferenceEntry -->
715
716      <owl:Class rdf:about="http://visit.de/ontologies/vismo/ReferenceEntry">
717          <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E84_Information_Carrier"/>
718          <rdfs:comment>A Reference Entry contains further information about the reference of a
719          vismo:Resource in a given vismo:Reference entity, like the page numbers for example.</
720          rdfs:comment>
721          <rdfs:label>Reference Entry</rdfs:label>
722      </owl:Class>
723
724
725      <!-- http://visit.de/ontologies/vismo/ReferenceType -->
726
727      <owl:Class rdf:about="http://visit.de/ontologies/vismo/ReferenceType">
728          <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E55_Type"/>
729          <rdfs:comment>Summarises the various types that references in the cultural heritage domain can
730          have.</rdfs:comment>
731          <rdfs:label>Reference Type</rdfs:label>
732      </owl:Class>
733
734
735      <!-- http://visit.de/ontologies/vismo/Resource -->
736
737      <owl:Class rdf:about="http://visit.de/ontologies/vismo/Resource">
738          <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E1_CRM_Entity"/>
739          <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
740          <rdfs:comment>A vismo:Resource adds descriptive functionality to the resources used in the
741          ViSIT context, therefore adding the possibilities of adding comments, descriptions, as well as
742          literature information to the given resource.</rdfs:comment>
743          <rdfs:label>Resource</rdfs:label>
744      </owl:Class>
745
746
747      <!-- http://visit.de/ontologies/vismo/Room -->
748
749      <owl:Class rdf:about="http://visit.de/ontologies/vismo/Room">
750          <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E53_Place"/>
751          <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E84_Information_Carrier"/>
752          <rdfs:subClassOf rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
753          <rdfs:comment>A room in the classic sense. Can only be associated with its vismo:Architecture
754          entity that contains it.</rdfs:comment>
755          <rdfs:label>Room</rdfs:label>
756      </owl:Class>
757
758
759      <!-- http://visit.de/ontologies/vismo/SacralBuilding -->
760
761      <owl:Class rdf:about="http://visit.de/ontologies/vismo/SacralBuilding">
762          <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E55_Type"/>
763          <rdfs:comment>A further type characterisation for vismo:Architecture entities, classifying them
764          by a sacral type.</rdfs:comment>
          <rdfs:label>Sacral Building</rdfs:label>
      </owl:Class>

```

```

765 <!-- http://visit.de/ontologies/vismo/SecularBuilding -->
766
767 <owl:Class rdf:about="http://visit.de/ontologies/vismo/SecularBuilding">
768   <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E55_Type"/>
769   <rdfs:comment>A further type characterisation for vismo:Architecture entities, classifying them
770     by a secular type.</rdfs:comment>
771   <rdfs:label>Secular Building</rdfs:label>
772 </owl:Class>
773
774
775 <!-- http://visit.de/ontologies/vismo/StructuralEvolution -->
776
777 <owl:Class rdf:about="http://visit.de/ontologies/vismo/StructuralEvolution">
778   <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E11_Modification"/>
779   <rdfs:subClassOf rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
780   <rdfs:comment>A vismo:StructuralEvolution changes a vismo:Architecture entity in some way. A
781     change in its basic vismo:Function can thereby be established. For example a castle that changes
782     from its military function to a museum.</rdfs:comment>
783   <rdfs:label>StructuralEvolution</rdfs:label>
784 </owl:Class>
785
786
787 <!-- http://visit.de/ontologies/vismo/Technique -->
788
789 <owl:Class rdf:about="http://visit.de/ontologies/vismo/Technique">
790   <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E55_Type"/>
791   <rdfs:comment>Techniques subsume the naming of production processes, which have the result of
792     producing an vismo:Object that are associated with the cultural heritage domain.</rdfs:comment>
793   <rdfs:label>Technique</rdfs:label>
794 </owl:Class>
795
796
797 <!-- http://visit.de/ontologies/vismo>Title -->
798
799 <owl:Class rdf:about="http://visit.de/ontologies/vismo>Title">
800   <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E35_Title"/>
801   <rdfs:comment>A Class to comprise a title in combination with a superordinate title of a
802     reference collection that contains this vismo:Reference entity.</rdfs:comment>
803   <rdfs:label>Title</rdfs:label>
804 </owl:Class>
805
806
807 <!-- http://visit.de/ontologies/vismo/Trade -->
808
809 <owl:Class rdf:about="http://visit.de/ontologies/vismo/Trade">
810   <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E9_Move"/>
811   <rdfs:subClassOf rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
812   <rdfs:comment>This class subsumes a trade of some or more vismo:TradeGood entities. A trade
813     should always be associated with a vismo:TradeRoute.</rdfs:comment>
814   <rdfs:label>Trade</rdfs:label>
815 </owl:Class>
816
817
818 <!-- http://visit.de/ontologies/vismo/TradeGood -->
819
820 <owl:Class rdf:about="http://visit.de/ontologies/vismo/TradeGood">
821   <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E55_Type"/>
822   <rdfs:comment>Tradegoods subsume titled names for the goods that are transported and sold on
823     vismo:TradeRoute objects.</rdfs:comment>
824   <rdfs:label>Tradegood</rdfs:label>
825 </owl:Class>
826
827
828 <!-- http://visit.de/ontologies/vismo/Traderoute -->
829
830 <owl:Class rdf:about="http://visit.de/ontologies/vismo/Traderoute">
831   <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
832   <rdfs:comment>A vismo:TradeRoute encompasses several vismo:City entities that are associated
833     with the vismo:Trade that is associated with the given trade route. These cities can thereby be
834     starting or end location, as well as an intermediate station.</rdfs:comment>
835   <rdfs:label>Traderoute</rdfs:label>
836 </owl:Class>
837
838
839 <!-- http://visit.de/ontologies/vismo/WorkingGroup -->

```

```

840   <owl:Class rdf:about="http://visit.de/ontologies/vismo/WorkingGroup">
841     <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E74_Group"/>
842     <rdfs:subClassOf rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
843     <rdfs:comment>Frauke :</rdfs:comment>
844     <rdfs:label>WorkingGroup</rdfs:label>
845   </owl:Class>
846 </rdf:RDF>

```

Listing 49: VisMo Ontologie in der letzten (englischen) Version.

```

1   {
2     "Object": {
3       "type": "http://visit.de/ontologies/vismo/Object",
4       "object_identifiedby_title": "string",
5       "object_has_description": "string",
6       "object_exemplifies_function": "string",
7       "object_inventory_number": "string",
8       "object_description": "string_long",
9       "object_comment": "string_long",
10      "object_keyword": "string",
11      "object_iconography": "string",
12      "object_literature": "string_long",
13      "object_current_owner": "entity_reference (http://visit.de/ontologies/vismo/Institution)",
14      "object_current_location": "entity_reference (http://visit.de/ontologies/vismo/Institution)",
15      "object_currentlocation_arch": "entity_reference (http://visit.de/ontologies/vismo/Architecture)",
16      "object_composedof_object": "entity_reference (http://visit.de/ontologies/vismo/Object)",
17      "object_partof_object": "entity_reference (http://visit.de/ontologies/vismo/Object)",
18      "object_tookpartin_activity": "entity_reference (http://visit.de/ontologies/vismo/Activity)",
19      "object_depicts_person": "entity_reference (http://visit.de/ontologies/vismo/Person)",
20      "object_depicts_architecture": "entity_reference (http://visit.de/ontologies/vismo/Architecture)",
21      "object_depicts_place": "entity_reference (http://visit.de/ontologies/vismo/Place)",
22      "object_depicts_activity": "entity_reference (http://visit.de/ontologies/vismo/Activity)",
23      "object_helpfullinks": "string",
24      "object_thumbnail": "image",
25      "object_prefidentifier_inscriptio": {
26        "type": "http://erlangen-crm.org/170309/E34_Inscription",
27        "inscription_text": "string",
28        "inscription_has_type": "string",
29        "inscription_signature": "string",
30        "inscription_mounting": "string",
31        "inscription_date": "string"
32      },
33      "object_producedby_production": {
34        "type": "http://erlangen-crm.org/170309/E12_Production",
35        "object_employs_material": "string",
36        "production_used_technique": "string",
37        "production_doneby_person": "entity_reference (http://visit.de/ontologies/vismo/Person)",
38        "production_doneby_group": "entity_reference (http://visit.de/ontologies/vismo/Group)",
39        "production_tookplaceat_place": "entity_reference (http://visit.de/ontologies/vismo/Place)",
40        "production_dating": {
41          "object_prod_dating_start": "string",
42          "object_prod_dating_end": "string",
43          "production_date_sometime": "string",
44          "object_prod_dating_century": "list_string"
45        }
46      },
47      "object_has_dimension": {
48        "type": "http://erlangen-crm.org/170309/E54_Dimension",
49        "dimension_has_measurementunit": "string",
50        "dimension_hasvalue": "string"
51      },
52      "object_transferred_custody": {
53        "type": "http://erlangen-crm.org/170309/E10_Transfer_of_Custody",
54        "custody_receiving_person": "entity_reference (http://visit.de/ontologies/vismo/Person)",
55        "custody_receiving_group": "entity_reference (http://visit.de/ontologies/vismo/Group)",
56        "custody_from_person": "entity_reference (http://visit.de/ontologies/vismo/Person)",
57        "custody_from_group": "entity_reference (http://visit.de/ontologies/vismo/Group)",
58        "object_toc_dating": {
59          "object_toc_dating_exact": "datetime",
60          "object_toc_dating_start": "string",
61          "object_toc_dating_end": "string",
62          "object_toc_dating_sometime": "string",
63          "object_toc_dating_century": "list_string"
64        }
65      },
66      "object_dating": {
67        "type": "http://visit.de/ontologies/vismo/Dating",
68        "object_dating_start": "string",
69        "object_dating_end": "string",
70        "object_dating_sometime": "string",
71        "object_dating_century": "list_string"
72      },
73      "object_refentry": {

```

Adapter Name *	visittestrepo	Machine name: vismotest
The human-readable name of this adapter. This name must be unique.		
Description	visittestrepo	
<p>The text will be displayed on the <i>adapter collection</i> page.</p> <p><input checked="" type="checkbox"/> Writable Is this Adapter writable?</p> <p><input checked="" type="checkbox"/> Preferred Local Store Is this Adapter the preferred local store?</p>		
Read URL	http://localhost:8081/rdf4j-server/repositories/visittestrepo	
Write URL	http://localhost:8081/rdf4j-server/repositories/visittestrepo/statements	
<p><input type="checkbox"/> Use graph independent rewriting rewrite queries, so that remote SPARQL storages with non-standard dataset handling do always answer right</p> <p>Default Graph URI *</p> <p>http://visit.de/data/</p> <p>Graph URI that is used to store triples in by default. May also be used as a base for new entity URIs.</p>		
Ontology graphs		

Graphs that are considered to be containing ontology information. These are used to compute class and property information like hierarchies, domain/range, etc. Leave empty let system automatically detect the graphs.

Abbildung 62: Übersicht der Konfiguration eines WissKI Salz Adapters, Teil 1.

The screenshot shows the 'Ontology graphs' tab of the WissKI Salz Adapters configuration interface. The page has a sidebar on the left with sections for 'Ontology graphs', 'Same As properties', and 'Compute type and property hierarchy and domains and ranges'. The main content area contains several panels:

- "Same As" properties**: A panel with a text input field containing the URL <http://www.w3.org/2002/07/owl#sameAs>.
- The properties this store uses to mark two URIs as meaning the same (Drupal) entity.**: A note stating "All of them will be used at the same time when saving a matching pair. Make sure these are symmetric."
- Add standard sameAs property**: A section with a dropdown menu showing "- select -" and a link to "Inverse property selection".
- Allows selecting properties in inverse direction in pathbuilder.**: A note stating "These properties are marked with a leading '^'. E.g. for '^ex:prop1', the triple x2 ex:prop x1 must hold instead of x1 ex:prop1 x2."
- ▼ COMPUTE TYPE AND PROPERTY HIERARCHY AND DOMAINS AND RANGES**: A section with a "READ CAREFULLY" note and a list of what clicking the 'Start Reasoning' button initiates:
 - the class hierarchy
 - the property hierarchy
 - the domains of all properties
 - the ranges of all properties
 A note states: "In the specified triple store. This will take several minutes. The pathbuilders relying on this adapter will become much faster by doing this."
- Start Reasoning**: A button with a dropdown menu showing "Always do reasoning on this adapter." (checked), "Re-Compute results" (unchecked), and "You already have reasoning results in your cache".
- CHECK REASONING RESULTS**: A button.
- Update adapter**: A blue button.
- Delete**: A red button.

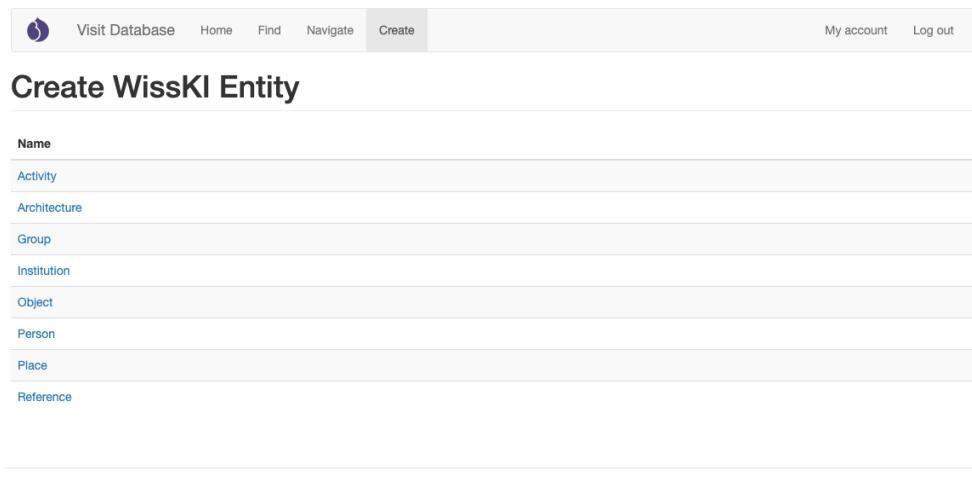
Abbildung 63: Übersicht der Konfiguration eines WissKI Salz Adapters, Teil 2.

Currently loaded Ontology:		
Name	IRI	Version
http://visit.de/ontologies/vismo/	none	http://visit.de/ontologies/vismo/0.4.5/
http://erlangen-crm.org/170309/	none	none
http://xmlns.com/foaf/0.1/	none	none
Delete Ontology		
Short Name	URI	
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#	
skos	http://www.w3.org/2004/02/skos/core#	
protege	http://protege.stanford.edu/plugins/owl/protege#	
xsp	http://www.owl-ontologies.com/2005/08/07/xsp.owl#	
owl	http://www.w3.org/2002/07/owl#	
xsd	http://www.w3.org/2001/XMLSchema#	
swrl	http://www.w3.org/2003/11/swrl#	
ecrm	http://erlangen-crm.org/170309/	
swrlb	http://www.w3.org/2003/11/swrlb#	
rdfs	http://www.w3.org/2000/01/rdf-schema#	
crm	http://www.cidoc-crm.org/cidoc-crm/	
xml	http://www.w3.org/XML/1998/namespace	
schema	http://schema.org/	
terms	http://purl.org/dc/terms/	
ns	http://www.w3.org/2003/06/sw-vocab-status/ns#	
wot	http://xmlns.com/wot/0.1/	
foaf	http://xmlns.com/foaf/0.1/	

Abbildung 64: Detailansicht einer definierten Ontology im WissKI System am Beispiel VisMo für das ViST Projekt.

Edit Pathbuilder: visittestrepo_pathes						
Home > Administration > Configuration > WissKI > Pathbuilders		Show row weights				
+ Add Path		+ Add Existing Path				
TITLE	PATH	ENABLED	FIELD TYPE	CARDINALITY	WEIGHT	OPERATIONS
↳ Object	Group [http://visit.de/ontologies/vismo/Object]	✓	Text (plain)	Unlimited	Unlimited	Edit ↗
↳ Object_identifiedBy_Title	http://visit.de/ontologies/vismo/Object -> ecrm:P1_is_identified_by -> ecrm:E35_Title	✓	Text (plain)	Unlimited	Unlimited	Edit ↗
↳ Object_has_Description	http://visit.de/ontologies/vismo/Object -> ecrm:P2_has_type -> http://visit.de/ontologies/vismo/Description	✓	Text (plain)	Unlimited	Unlimited	Edit ↗
↳ Object_exemplifies_Function	http://visit.de/ontologies/vismo/Object -> ecrm:P32_exemplifies -> http://visit.de/ontologies/vismo/Function	✓	Text (plain)	Unlimited	Unlimited	Edit ↗
↳ Object_Dating	Group [http://visit.de/ontologies/vismo/Object -> ecrm:p160_has_temporal_projection -> http://visit.de/ontologies/vismo/Dating]	✓	Text (plain)	Unlimited	Unlimited	Edit ↗
↳ Object_Dating_start	http://visit.de/ontologies/vismo/Object -> ecrm:p160_has_temporal_projection -> http://visit.de/ontologies/vismo/Dating	✓	Text (plain)	Unlimited	Unlimited	Edit ↗
↳ Object_Dating_end	http://visit.de/ontologies/vismo/Object -> ecrm:p160_has_temporal_projection -> http://visit.de/ontologies/vismo/Dating	✓	Text (plain)	Unlimited	Unlimited	Edit ↗
↳ Object_Dating_sometime	http://visit.de/ontologies/vismo/Object -> ecrm:p160_has_temporal_projection -> http://visit.de/ontologies/vismo/Dating	✓	Text (plain)	Unlimited	Unlimited	Edit ↗
↳ Object_Dating_century	http://visit.de/ontologies/vismo/Object -> ecrm:p160_has_temporal_projection -> http://visit.de/ontologies/vismo/Dating	✓	List (text)	Unlimited	Unlimited	Edit ↗
↳ Object_composedOf_Object	http://visit.de/ontologies/vismo/Object -> ecrm:p46_is_composed_of -> http://visit.de/ontologies/vismo/Object	✓	Entity reference	Unlimited	Unlimited	Edit ↗
↳ Object_partOf_Object	http://visit.de/ontologies/vismo/Object -> ecrm:p46_forms_part_of -> http://visit.de/ontologies/vismo/Object	✓	Entity reference	Unlimited	Unlimited	Edit ↗
↳ Object_description	http://visit.de/ontologies/vismo/Object	✓	Text (plain, long)	Unlimited	Unlimited	Edit ↗
↳ Object_producedBy_Production	Group [http://visit.de/ontologies/vismo/Object -> ecrm:P108_was_produced_by -> ecrm:E12_Production]	✓	Unlimited	Unlimited	Unlimited	Edit ↗
↳ Object_employs_Material	http://visit.de/ontologies/vismo/Object -> ecrm:E12_was_produced_by -> ecrm:p126_Employed -> ecrm:E57_Material	✓	Text (plain)	Unlimited	Unlimited	Edit ↗
↳ Production_used_Technique	http://visit.de/ontologies/vismo/Object -> ecrm:p108_was_produced_by -> ecrm:p32_used_general_technique -> http://visit.de/ontologies/vismo/Technique	✓	Text (plain)	Unlimited	Unlimited	Edit ↗

Abbildung 65: Übersicht der ersten Pfade des für das ViSiT Projekt definierten Pathbuilders.



Powered by [Drupal](#)

[Contact](#)

Abbildung 66: Ausgangs-Interface zum Erzeugen einer Hauptentität im WissKI System.

```

74     "type": "http://visit.de/ontologies/vismo/ReferenceEntry",
75     "object_refentry_pages": "string",
76     "object_refentry_in_reference": "entity_reference (http://visit.de/ontologies/vismo/Reference)"
77   },
78   "object_digitalrepresentation": {
79     "type": "http://visit.de/ontologies/vismo/DigitalRepresentation",
80     "object_dr_technicalmetadata": "string_long"
81   }
82 },
83 "Person": {
84   "type": "http://visit.de/ontologies/vismo/Person",
85   "person_idby_actorappel": "string",
86   "person_hastype_profession": "string",
87   "person_firstname": "string",
88   "person_lastname": "string",
89   "person_pseudonym": "string",
90   "person_alternatename": "string",
91   "person_carries_title": "string",
92   "person_comment": "string_long",
93   "person_description": "string_long",
94   "person_keyword": "string",
95   "person_iconography": "string",
96   "person_parentof_person": "entity_reference (http://visit.de/ontologies/vismo/Person)",
97   "person_ischildof_person": "entity_reference (http://visit.de/ontologies/vismo/Person)",
98   "person_ownerof_architecture": "entity_reference (http://visit.de/ontologies/vismo/Architecture)",
99   "person_motiv_arch_production": "entity_reference (http://visit.de/ontologies/vismo/Architecture)",
100  "person_carriedout_arch_prod": "entity_reference (http://visit.de/ontologies/vismo/Architecture)",
101  "person_infld_arch_production": "entity_reference (http://visit.de/ontologies/vismo/Architecture)",
102  "person_motiv_structevol": "entity_reference (http://visit.de/ontologies/vismo/Architecture)",
103  "person_carriedout_structevol": "entity_reference (http://visit.de/ontologies/vismo/Architecture)",
104  "person_infld_structevol": "entity_reference (http://visit.de/ontologies/vismo/Architecture)",
105  "person_depictedon_object": "entity_reference (http://visit.de/ontologies/vismo/Object)",
106  "person_participatedin_activity": "entity_reference (http://visit.de/ontologies/vismo/Activity)",
107  "person_receivedcustody_of_object": "entity_reference (http://visit.de/ontologies/vismo/Object)",
108  "person_lostcustody_of_object": "entity_reference (http://visit.de/ontologies/vismo/Object)",
109  "person_helpfullinks": "string",
110  "person_thumbnail": "image",
111  "person_birth": {
112    "type": "http://erlangen-crm.org/170309/E67_Birth",
113    "person_mother": "entity_reference (http://visit.de/ontologies/vismo/Person)",
114    "person_father": "entity_reference (http://visit.de/ontologies/vismo/Person)",
115    "person_birthplace": "entity_reference (http://visit.de/ontologies/vismo/Place)",
116    "person_birth_dating": {
117      "person_birth_dating_exact": "datetime",
118      "person_birth_dating_start": "string",
119      "person_birth_dating_end": "string",
120      "person_birth_dating_sometime": "string"
121    }
122  },
123  "person_death": {
124    "type": "http://erlangen-crm.org/170309/E69_Death",
125    "person_deathplace": "entity_reference (http://visit.de/ontologies/vismo/Place)",
126    "person_death_dating": {
127      "person_death_dating_exact": "datetime",

```

Home > Create WissKI Entity	
Show row weights	
TITEL	
<input type="text"/> ⓘ Add another item	
Show row weights	
OBJEKTBEZEICHNUNG	
<input type="text"/> ⓘ <p>Erläutern Sie hier möglichst präzise und schlagwortartig ein, um was für einen Objekttyp es sich handelt, z.B. "Gemälde", "Türrahmung" oder "Henkelkrug". Achtung! Für den Titel eines Objekts (z.B. "Mona Lisa") zieht das Feld "Titel" zur Verfügung.</p> Add another item	
Show row weights	
FUNKTION	
<input type="text"/> ⓘ <p>Hier können Sie beschreiben, was das Objekt ursprünglich für eine Funktion hatte und in welchen Zusammenhangen es genutzt wurde, z.B.: Spezifikation einer Bügelflasche als "Bienenfalle" oder eines Henkelkrugs als "Essigkrug".</p> Add another item	
Show row weights	
ALLGEMEINE OBJEKTDATIERUNG	
<input type="text"/> ⓘ Add new wisski entity <p>In dieser Gruppe können Sie Angaben zur allgemeinen Objektdatierung machen.</p> Add another item	
Show row weights	
OBJEKT BESTEHT AUS: (TEILOBJEKten)	
<input type="text"/> ⓘ <p>Referenzfeld – Wenn das Objekt aus mehreren Teilen besteht, geben Sie diese hier bitte einzeln an.</p> Add another item	
Show row weights	
TEILOBJEKT VON: (OBJEKT)	
<input type="text"/> ⓘ	

Abbildung 67: Eingabe-Interface für ein Ausstellungsobjekt im ViSIT WissKI System.

The screenshot shows a web-based application interface for managing historical objects. At the top, there's a navigation bar with links for 'Visit Database', 'Home', 'Find', 'Navigate', 'Create', 'My account', and 'Log out'. Below the navigation is a breadcrumb trail: 'Navigate / Object / Partisane (587)'. The main title is 'Partisane (587)'. Underneath, there are tabs for 'View' (which is selected), 'Edit', 'Delete', 'Triples', and 'Devel'. A detailed list of object properties follows:

- Objektbezeichnung**: Partisane
- Prunkpartisane**
- Inscription**
- Text**: I.P.D.G.E.P.S.R.-I.P.E.G.D.L.
- Anbringung**: Über dem Passauer Wolf
- Datierung**: 1698
- Inventarnummer**: 00045
- Darstellung**: Passauer Wolf
- Wappen des Fürstbischofs Johann Philipp Graf von Lamberg**
- Standort (Museum)**: Oberhausmuseum
- Herstellung**
- Material**: Eisen
- Holz**
- Technik**: Geschmiedet
- Objekt besteht aus: (Teilobjekten)**: Wappen des Fürstbischofs Johann Philipp von Lamberg
- Standort (Bauwerk)**: Veste Oberhaus
- Maße**
- Abmessung**: Breite
- Wert**: 17,7 cm
- Abmessung**: 215
- Wert**: 215 cm
- Allgemeine Objektdatierung**: freie Datierung

Abbildung 68: Beispiel Ausgabe-Interface einer Partisane des ViSIT WissKI Systems.

```

128      "person_death_dating_start": "string",
129      "person_death_dating_end": "string",
130      "person_death_dating_sometime": "string"
131    },
132  },
133  "person_marriage": {
134    "type": "http://visit.de/ontologies/vismo/Marriage",
135    "marriage_partner_person": "entity_reference (http://visit.de/ontologies/vismo/Person)",
136    "marriage_begin_dating": {
137      "marriage_begin_dating_exact": "datetime",
138      "marriage_begin_dating_start": "string",
139      "marriage_begin_dating_end": "string",
140      "marriage_begin_dating_sometime": "string"
141    },
142    "marriage_end_dating": {
143      "marriage_end_dating_exact": "datetime",
144      "marriage_end_dating_start": "string",
145      "marriage_end_dating_end": "string",
146      "marriage_end_dating_sometime": "string"
147    }
148  },
149  "person_refentry": {
150    "type": "http://visit.de/ontologies/vismo/ReferenceEntry",
151    "person_refentry_pages": "string",
152    "person_refentry_in_reference": "entity_reference (http://visit.de/ontologies/vismo/Reference)"
153  },
154  "person_digitalrepresentation": {
155    "type": "http://visit.de/ontologies/vismo/DigitalRepresentation",
156    "person_dr_technicalmetadata": "string_long"
157  },
158  },
159  "Architecture": {
160    "type": "http://visit.de/ontologies/vismo/Architecture",
161    "architecture_idby_title": "string",
162    "arch_sacraltype": "string",
163    "arch_has_seculartype": "string",
164    "arch_bishopricaffiliation": "string",
165    "arch_geographicaffiliation": "string",
166    "arch_orderaffiliation": "string",
167    "architecture_description": "string_long",

```

```

168 "architecture_comment": "string_long",
169 "architecture_keyword": "string",
170 "architecture_iconography": "string",
171 "architecture_innerdescription": "string_long",
172 "architecture_outerdescription": "string_long",
173 "architecture_depictedby_object": "entity_reference (http://visit.de/ontologies/vismo/Object)",
174 "architecture_buildinghistory": "string_long",
175 "arch_currentlyholds_object": "entity_reference (http://visit.de/ontologies/vismo/Object)",
176 "architecture_exemplify_function": "string",
177 "architecture_location_place": "entity_reference (http://visit.de/ontologies/vismo/Place)",
178 "arch_currentowner_person": "entity_reference (http://visit.de/ontologies/vismo/Person)",
179 "arch_currentowner_group": "entity_reference (http://visit.de/ontologies/vismo/Group)",
180 "arch_currentowner_institution": "entity_reference (http://visit.de/ontologies/vismo/Institution)",
181 "architecture_contains_arch": "entity_reference (http://visit.de/ontologies/vismo/Architecture)",
182 "architecture_fallswithin_arch": "entity_reference (http://visit.de/ontologies/vismo/Architecture)"
183 ,
184 "architecture_tookpartin_activity": "entity_reference (http://visit.de/ontologies/vismo/Activity)",
185 "architecture_helpfullinks": "string",
186 "architecture_thumbnail": "image",
187 "arch_producedby_production": {
188   "type": "http://erlangen-crm.org/170309/E12\_Production",
189   "production_motivatedby_person": "entity_reference (http://visit.de/ontologies/vismo/Person)",
190   "production_carriedoutby_person": "entity_reference (http://visit.de/ontologies/vismo/Person)",
191   "production_inflby_person": "entity_reference (http://visit.de/ontologies/vismo/Person)",
192   "arch_prod_motivatedby_group": "entity_reference (http://visit.de/ontologies/vismo/Group)",
193   "arch_prod_carriedoutby_group": "entity_reference (http://visit.de/ontologies/vismo/Group)",
194   "arch_prod_inflby_group": "entity_reference (http://visit.de/ontologies/vismo/Group)",
195   "arch_production_dating": {
196     "arch_prod_dating_start": "string",
197     "arch_prod_dating_end": "string",
198     "arch_production_sometime": "string",
199     "arch_prod_dating_century": "list_string"
200   },
201 },
202 "arch_modifiedby_structevolution": {
203   "type": "http://visit.de/ontologies/vismo/StructuralEvolution",
204   "structuralevolution_idby_title": "string",
205   "structuralevolution_description": "string_long",
206   "structuralevolution_comment": "string_long",
207   "structevol_exemplifies.function": "string",
208   "structevol_motivatedby_person": "entity_reference (http://visit.de/ontologies/vismo/Person)",
209   "structevol_carriedoutby_person": "entity_reference (http://visit.de/ontologies/vismo/Person)",
210   "structevol_influencedby_person": "entity_reference (http://visit.de/ontologies/vismo/Person)",
211   "structevol_motivby_group": "entity_reference (http://visit.de/ontologies/vismo/Group)",
212   "structevol_carriedoutby_group": "entity_reference (http://visit.de/ontologies/vismo/Group)",
213   "structevol_inflby_group": "entity_reference (http://visit.de/ontologies/vismo/Group)",
214   "arch_structevol_dating": {
215     "arch_structevol_dating_start": "string",
216     "arch_structevol_dating_end": "string",
217     "arch_evol_dat_sometime": "string",
218     "arch_structevol_dating_century": "list_string"
219   },
220 },
221 "arch_refentry": {
222   "type": "http://visit.de/ontologies/vismo/ReferenceEntry",
223   "arch_refentry_pages": "string",
224   "arch_refentry_in_reference": "entity_reference (http://visit.de/ontologies/vismo/Reference)"
225 },
226 "architecture_digitalrepresentati": {
227   "type": "http://visit.de/ontologies/vismo/DigitalRepresentation",
228   "architecture_dr_techmetadata": "string_long"
229 },
230 },
231 "Place": {
232   "type": "http://visit.de/ontologies/vismo/Place",
233   "place_idby_placeappel": "string",
234   "place_description": "string_long",
235   "place_comment": "string_long",
236   "place_keyword": "string",
237   "place.iconography": "string",
238   "place_holds_architecture": "entity_reference (http://visit.de/ontologies/vismo/Architecture)",
239   "place_witnessed_activity": "entity_reference (http://visit.de/ontologies/vismo/Activity)",
240   "place_wasbirthplaceof_person": "entity_reference (http://visit.de/ontologies/vismo/Person)",
241   "place_wasdeathplaceof_person": "entity_reference (http://visit.de/ontologies/vismo/Person)",
242   "place_isdepictedby_object": "entity_reference (http://visit.de/ontologies/vismo/Object)",
243   "place_helpfullinks": "string",
244   "place_thumbnail": "image",
245   "place_refentry": {
246     "type": "http://visit.de/ontologies/vismo/ReferenceEntry",
247     "place_refentry_pages": "string",
248     "place_refentry_in_reference": "entity_reference (http://visit.de/ontologies/vismo/Reference)"
249   },
250   "place_digitalrepresentation": {
251     "type": "http://visit.de/ontologies/vismo/DigitalRepresentation",

```

```

251     "place_dr_techmetadata": "string"
252   }
253 },
254 "Institution": {
255   "type": "http://visit.de/ontologies/vismo/Institution",
256   "institution_idby_appel": "string",
257   "institution_ownerof_arch": "entity_reference (http://visit.de/ontologies/vismo/Architecture)",
258   "institution_fallswithin_place": "entity_reference (http://visit.de/ontologies/vismo/Place)",
259   "institution_address": "string",
260   "institution_owns_catalog": "entity_reference (http://visit.de/ontologies/vismo/Reference)",
261   "institution_loc_catalog": "entity_reference (http://visit.de/ontologies/vismo/Reference)",
262   "institution_helpfullinks": "string"
263 },
264 "Group": {
265   "type": "http://visit.de/ontologies/vismo/Group",
266   "group_idby_actorappel": "string",
267   "group_keyword": "string",
268   "group_iconography": "string",
269   "group_produced_object": "entity_reference (http://visit.de/ontologies/vismo/Object)",
270   "group_ownerof_architecture": "entity_reference (http://visit.de/ontologies/vismo/Architecture)",
271   "group_motiv_arch_production": "entity_reference (http://visit.de/ontologies/vismo/Architecture)",
272   "group_carriedout_arch_production": "entity_reference (http://visit.de/ontologies/vismo/
273     Architecture)",
274   "group_infl_arch_production": "entity_reference (http://visit.de/ontologies/vismo/Architecture)",
275   "group_motiv_structevol": "entity_reference (http://visit.de/ontologies/vismo/Architecture)",
276   "group_carriedout_structevol": "entity_reference (http://visit.de/ontologies/vismo/Architecture)",
277   "group_infl_structevol": "entity_reference (http://visit.de/ontologies/vismo/Architecture)",
278   "group_receivedcustodyof_object": "entity_reference (http://visit.de/ontologies/vismo/Object)",
279   "group_lostcustodyof_object": "entity_reference (http://visit.de/ontologies/vismo/Object)",
280   "group_refentry": {
281     "type": "http://visit.de/ontologies/vismo/ReferenceEntry",
282     "group_refentry_pages": "string",
283     "group_refentry_in_reference": "entity_reference (http://visit.de/ontologies/vismo/Reference)"
284   },
285 },
286 "Reference": {
287   "type": "http://visit.de/ontologies/vismo/Reference",
288   "reference_keyword": "string",
289   "reference_has_type": "string",
290   "reference_publisher": "string",
291   "reference_series": "string",
292   "reference_volume": "integer",
293   "reference_pages": "integer",
294   "reference_catalog_owner": "entity_reference (http://visit.de/ontologies/vismo/Institution)",
295   "reference_catalog_location": "entity_reference (http://visit.de/ontologies/vismo/Institution)",
296   "reference_title": {
297     "type": "http://visit.de/ontologies/vismo>Title",
298     "reference_title_title": "string",
299     "reference_title_superordinate": "string"
300   },
301   "reference_producedby_production": {
302     "type": "http://erlangen-crm.org/170309/E12_Production",
303     "production_authorname": "string",
304     "production_year": "integer",
305     "ref_production_placeofpub": "string"
306   },
307   "reference_entry": {
308     "type": "http://visit.de/ontologies/vismo/ReferenceEntry",
309     "reference_entry_pages": "string",
310     "reference_entry_about_activity": "entity_reference (http://visit.de/ontologies/vismo/Activity)",
311     "reference_entry_about_arch": "entity_reference (http://visit.de/ontologies/vismo/Architecture)",
312     "reference_entry_about_object": "entity_reference (http://visit.de/ontologies/vismo/Object)",
313     "reference_entry_about_place": "entity_reference (http://visit.de/ontologies/vismo/Place)",
314     "reference_entry_about_group": "entity_reference (http://visit.de/ontologies/vismo/Group)",
315     "reference_entry_about_person": "entity_reference (http://visit.de/ontologies/vismo/Person)"
316   },
317   "reference_catalog_dating": {
318     "type": "http://visit.de/ontologies/vismo/Dating",
319     "catalog_exhibition_start": "datetime",
320     "catalog_exhibition_end": "datetime"
321   },
322   "Activity": {
323     "type": "http://visit.de/ontologies/vismo/Activity",
324     "activity_idby_title": "string",
325     "activity_description": "string_long",
326     "activity_comment": "string_long",
327     "activity_keyword": "string",
328     "activity_iconography": "string",
329     "activity_hadparticipant_person": "entity_reference (http://visit.de/ontologies/vismo/Person)",
330     "activity_used_architecture": "entity_reference (http://visit.de/ontologies/vismo/Architecture)",
331     "activity_used_object": "entity_reference (http://visit.de/ontologies/vismo/Object)",
332     "activity_tookplaceat_place": "entity_reference (http://visit.de/ontologies/vismo/Place)",
333     "activity_isdepictedby_object": "entity_reference (http://visit.de/ontologies/vismo/Object)"
```

```
334     "activity_helpfullinks": "string",
335     "activity_thumbnail": "image",
336     "activity_dating": {
337         "type": "http://visit.de/ontologies/vismo/Dating",
338         "activity_dating_exact": "datetime",
339         "activity_dating_end": "string",
340         "activity_dating_start": "string",
341         "activity_dating_sometime": "string"
342     },
343     "activity_refentry": {
344         "type": "http://visit.de/ontologies/vismo/ReferenceEntry",
345         "activity_refentry_pages": "string",
346         "activity_refentry_in_reference": "entity_reference (http://visit.de/ontologies/vismo/Reference)"
347     },
348     "activity_digitalrepresentation": {
349         "type": "http://visit.de/ontologies/vismo/DigitalRepresentation",
350         "activity_dr_techmetadata": "string"
351     }
352 }
353 }
```

Listing 50: JSON “Schema” der ViSIT Daten, die über die REST API ausgespielt werden.

REFERENCES

- [Bot+10] Mario Botsch u. a. *Polygon mesh processing*. AK Peters/CRC Press, 2010.
- [Cida] ISO 21127:2006 - *Information and Documentation – A Reference Ontology for the Interchange of Cultural Heritage Information*. Standard. International Organization for Standardization, Sep. 2006.
- [Cidb] ISO 21127:2014 - *Information and Documentation – A Reference Ontology for the Interchange of Cultural Heritage Information*. Standard. International Organization for Standardization, Sep. 2014.
- [Doe03] Martin Doerr. "The CIDOC Conceptual Reference Module: An Ontological Approach to Semantic Interoperability of Metadata". In: *AI Magazine* 24.3 (2003), S. 75.
- [GH97] Michael Garland und Paul S Heckbert. "Surface simplification using quadric error metrics". In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. 1997, S. 209–216.
- [GH98] Michael Garland und Paul S Heckbert. "Simplifying surfaces with color and texture using quadric error metrics". In: *Proceedings Visualization'98 (Cat. No. 98CB36276)*. IEEE. 1998, S. 263–269.
- [Hit+07] Pascal Hitzler u. a. *Semantic Web: Grundlagen*. Springer-Verlag, 2007.
- [MM04] Frank Manola und Eric Miller. *RDF Primer*. W3C Recommendation. W3C, Feb. 2004. URL: <http://www.w3.org/TR/2004/REC-rdf-primer-20040210/>.