

VISIT - TECHNISCHE DOKUMENTATION

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

1. 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	11
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	16
3.7	Erzeugung des Layouts	16
3.8	Das Karten-Plug-In	17
3.9	Erstellung eines Templates	18
4	Karten-Applikation	19
4.1	Einpflegen der Daten in die Karten-Applikation	19
4.2	Erstellung der Startseite für die Karten-Applikation	19
4.3	Neues Kartenelement hinzufügen	21
4.4	Bearbeitung und Löschen von angelegten Kartenelementen	23
5	Glossar-Applikation	24
5.1	Einpflegen der Daten in die Glossar-Applikation	24
5.2	Erstellung der Startseite für die Glossar-Applikation	24
5.3	Hinzufügen einer neuen Zelle	25
5.4	Hinzufügen eines Events	25
5.5	Neuen Insassen anlegen	26
5.6	Bearbeitung und Löschung von Insassen	27
6	Galerie-Applikation	27
6.1	Einpflegen der Daten in die Glossar-Applikation	27
6.2	Erstellung der Startseite für die Glossar-Applikation	27
6.3	Neuen Insassen anlegen	27
7	Dateiverwaltung	27
7.1	Zugangsdaten zum Dateimanagement	27
7.2	ViSIT-Partner-Liste	29
7.3	Upload von 3D-Objekten und Bildern, Videos und anderen Dateien	30
7.4	Hochladen von Dateien	32
7.5	Veröffentlichung einer Datei im ViSIT-Netzwerk	34
8	Tobi - App Framework	35
9	Kompression	36
9.1	Motivation	36
9.2	Grundlagen	36
9.3	Installation (?)	40
9.4	Konfiguration	40
9.5	Zugriff über die Web-Oberfläche	45
9.6	Zugriff über die API	46
10	ViSIT Metadaten und die Semantische Datenbank	47
10.1	Theoretische Grundlagen für die Semantische Datenbank	47
10.2	Technische Details zur Semantischen Datenbank	52
10.3	WissKI - Wissenschaftliche KommunikationsInfrastruktur	55
10.4	Technischer Zugang zu den Metadaten - die ViSIT REST API	61

10.5 Wichtige Technische Charakteristika der Entwicklung und den Betrieb der Semantischen Datenbank	63
10.6 Zusatzfeatures - Erweiterung der Bedienbarkeit der Semantischen Datenbank	66
10.7 Semantische Datenbank - FAQ und häufig auftretende Probleme	66
11 Schluss	68
12 Appendix	69

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	12
Abbildung 4	Installation der Extensions	13
Abbildung 5	Änderung der Sprache	14
Abbildung 6	Konfiguration der Benutzereinstellungen	14
Abbildung 7	Änderung der Benutzereinstellungen und der Sprache	15
Abbildung 8	Änderung des Passworts	15
Abbildung 9	Hinzufügen einer neuen Seite	16
Abbildung 10	Erzeugung des Layouts der neu erstellten Seite	17
Abbildung 11	Auswahl der Plug-Ins	17
Abbildung 12	Einbindung eines Plug-Ins	18
Abbildung 13	Leere Kartenübersicht	19
Abbildung 14	Erstellung der Startseite für die Karten-Applikation .	19
Abbildung 15	Text für die Startseite der Applikationen, zu finden auf https://github.com/ViSIT-Dev/appbundle	20
Abbildung 16	Erstellung der Startseite für die Karten-Applikation .	21
Abbildung 17	Ein neues Kartenelement hinzufügen	21
Abbildung 18	Listenansicht über alle angelegten Kartenelemente .	22
Abbildung 19	Startseite der Karten-Applikation	22
Abbildung 20	Ansicht der Karte im Browser	23
Abbildung 21	Kartenelement - Point of Interest - mit Detailinformation	23
Abbildung 22	Ansicht der Glossar-Applikation auf einem Tablet . .	24
Abbildung 23	Übersicht über alle angelegten Insassen	25
Abbildung 24	Konfiguration der Glossar-Applikation	25
Abbildung 25	Befüllung der Startseite der Glossar-Applikation . .	26
Abbildung 26	Startseite der Glossar-Applikation	26
Abbildung 27	Hinzufügen einer neuen Zelle	27
Abbildung 28	Erstellen einer neuen Zelle	27
Abbildung 29	Angelegte Zellen in der Listenübersicht	28
Abbildung 30	Hinzufügen eines neuen Events	28
Abbildung 31	Angelegte Events in der Listenansicht	29
Abbildung 32	Anlegen eines neuen Insassens	29
Abbildung 33	Anlegen eines Insassens	30
Abbildung 34	Angelegte Insassen in der Listenansicht	30
Abbildung 35	Ansicht der Insassen im Browser	31
Abbildung 36	Einstellung der ViSIT App Extension	31
Abbildung 37	ViSIT App Extensions	31
Abbildung 38	Ansicht der ViSIT-Partner mit Zugang zur Medien-datenbank im Peer-to-Peer-Netzwerk	32

Abbildung 39	Ansicht der bereits verfügbaren Dateien in der Dateiliste	32
Abbildung 40	Ansicht des ausgefüllten Formulars für den Dateiupload	33
Abbildung 41	Bestätigungs Nachrichten nach einem erfolgreichen Upload	33
Abbildung 42	Hochgeladene Datei in der Listenübersicht	34
Abbildung 43	Grundlegende Elemente der Geometrie eines 3D-Modells	37
Abbildung 44	Informationen aus obigen Aussagen, kombiniert als Graph.	48
Abbildung 45	Grundlegende eigene Wissensbasis (oben), erweitert um zwei externe Wissensbasen (unten).	50
Abbildung 46	Arbeitsprozess hinter der Entwicklung des ViSIT Modells.	52
Abbildung 47	Technische Infrastruktur der Semantischen Datenbank des ViSIT Projekts.	54
Abbildung 48	Zwei Beispiel-Pfade aus der WissKI Konfiguration des ViSIT Projekts.	58
Abbildung 49	Erste Maske zum Editieren eines WissKI Pfads.	59
Abbildung 50	Zweite Maske zum Editieren eines WissKI Pfads.	60
Abbildung 51	Beispiel-Pfad aus der WissKI Konfiguration des ViSIT Projekts für eine Relation zwischen zwei Entitäten der Datenbank.	60
Abbildung 52	Modell der digitalen Repräsentationen.	62
Abbildung 53	WissKI Pathbuilder Ansicht, die den Download der Pfad-Datei anbietet.	64
Abbildung 54	pom.xml Konfiguration zum produktiven Deployment der REST API.	65
Abbildung 55	Einstellungen zum Verbinden des Triplestores. Hier gezeigt: lokale Konfiguration, während die Konfiguration für Produktiv- und Testsystem oben vorhanden aber auskommentiert ist.	66
Abbildung 56	Übersicht der Konfiguration eines WissKI Salz Adapters, Teil 1.	81
Abbildung 57	Übersicht der Konfiguration eines WissKI Salz Adapters, Teil 2.	82
Abbildung 58	Detailansicht einer definierten Ontology im WissKI System am Beispiel VisMo für das ViSIT Projekt.	83
Abbildung 59	Übersicht der ersten Pfade des für das ViSIT Projekt definierten Pathbuilders.	84
Abbildung 60	Ausgangs-Interface zum Erzeugen einer Hauptentität im WissKI System.	85
Abbildung 61	Eingabe-Interface für ein Ausstellungsobjekt im ViSIT WissKI System.	86
Abbildung 62	Beispiel Ausgabe-Interface einer Partisane des ViSIT WissKI Systems.	87

TABELLENVERZEICHNIS

Tabelle 1	Überblick über alle Konfigurationsmöglichkeiten der Kompressionskomponente	40
-----------	--	----

Tabelle 2	Beispiele für die sich bei unterschiedlichen Vertexanzahlen ergebenden Texturauflösungen bei der Standardkonfiguration.	44
Tabelle 3	Pro Kompressionsstufe notwendige Parameter bei der Bildkompression	44
Tabelle 4	API Funktionen für die Digital Representations.	62
Tabelle 5	API Funktionen zum Auslesen von Objekten.	63

¹ TODO, Universität Passau

² Lehrstuhl für verteilte Informationssysteme und Data Science, Universität Passau

³ TODO, FH Kufstein

⁴ TODO, Universität Passau

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

```
1 | 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 werden. 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 (Daemonmode). 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 erstmalige einloggen in das Backend (TYPO3) erfolgt mit dem **Benutzername: admin** und **Passwort: YoGrZOy1og**.

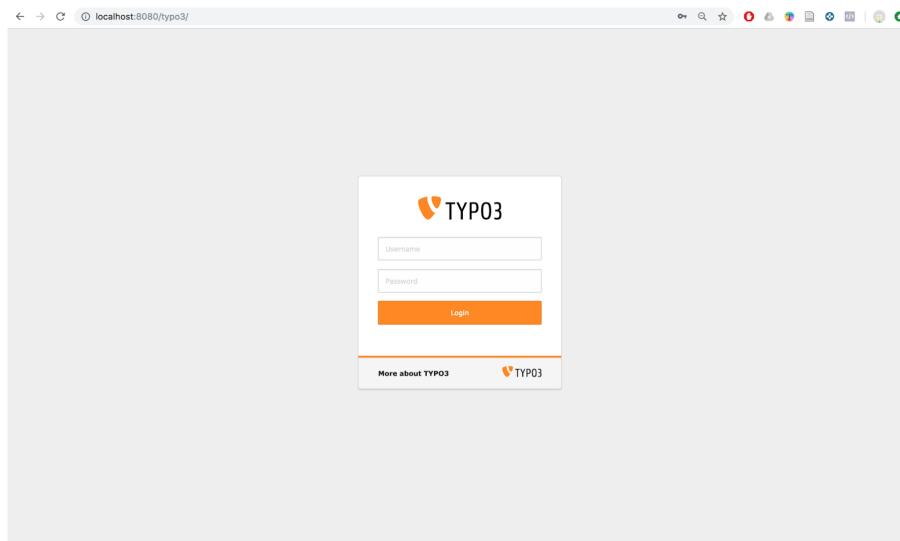


Abbildung 1: Das Login-Fenster für TYPO3 im Browser

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.

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-Benutzerstellenungen konfigurieren. Im Hauptbereich werden Webdokumente bearbeitet. Das TYPO3-Backend wird von links nach rechts abgearbeitet.

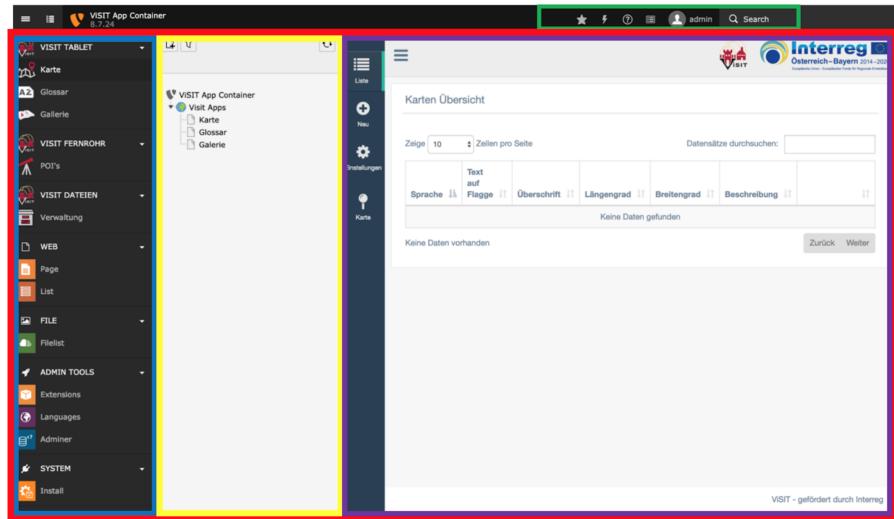


Abbildung 3: Aufbau des TYPO3-Backends

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 eingerückt 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

Upd.	A/D	Extension	Key	Version	State	Type	Actions
		Help>About	about	8.7.24	stable	System	
		TYPO3 Backend	backend	8.7.24	stable	System	
		Tools>Log	belog	8.7.24	stable	System	
		Backend User Administration	beuser	8.7.24	stable	System	
		Context Sensitive Help	context_help	8.7.24	stable	System	
		TYPO3 Core	core	8.7.24	stable	System	
		Help>TYPO3 Manual	cshmanual	8.7.24	stable	System	
		CSS styled content	css_styled_content	8.7.24	deprecated	System	
		Documentation	documentation	8.7.24	stable	System	
		Extbase Framework for Extensions	extbase	8.7.24	stable	System	
		Extension Manager	extensionmanager	8.7.24	stable	System	
		Frontend Editing	feedit	8.7.24	stable	System	
		Frontend Login for Website Users	felogin	8.7.24	stable	System	
		File>List	filelist	8.7.24	stable	System	
		Advanced file metadata	filenmetadata	8.7.24	stable	System	

Abbildung 4: Installation der Extensions

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 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.

A/D	Language	Locale	Last update	Actions
○	Afrikaans	af		
○	Albanian	sq		
○	Arabic	ar		
○	Bahasa Malaysia	ms		
○	Basque	eu		
○	Bosnian	bs		
○	Brazilian Portuguese	pt_BR		
○	Bulgarian	bg		
○	Catalan	ca		
○	Chinese (Simple)	ch		
○	Chinese (Trad)	zh		
○	Croatian	hr		
○	Czech	cs		
○	Danish	da		
...

Abbildung 5: Änderung der Sprache

Bei der Erstinstallation ist es der "admin", dabei wird ein Menü aufgeklappt, aus welchem die "User settings" ausgewählt werden (siehe Abbildung 6).

Abbildung 6: Konfiguration der Benutzereinstellungen

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 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.

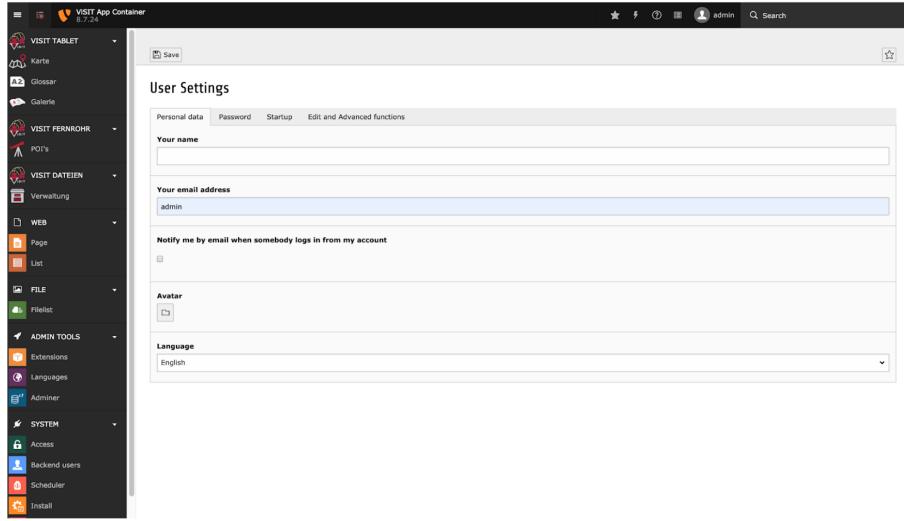


Abbildung 7: Änderung der Benutzereinstellungen und der Sprache

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. Gleicher 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.

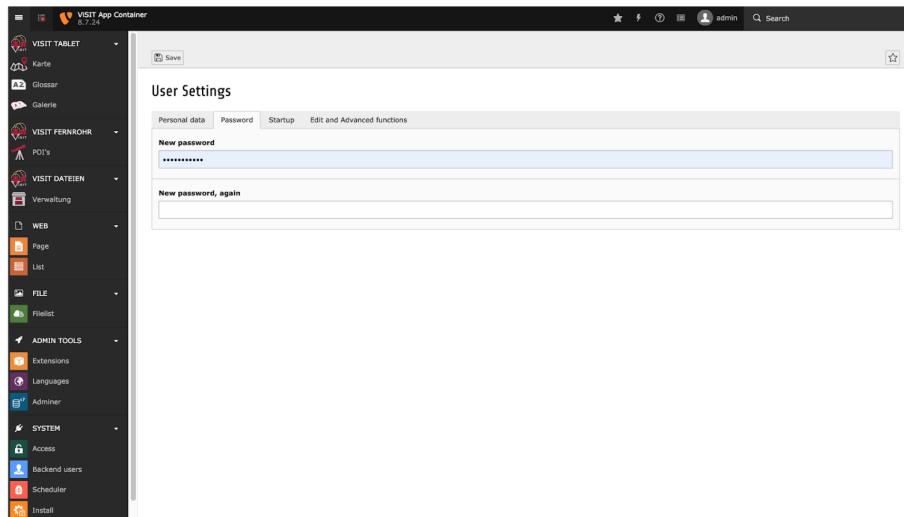


Abbildung 8: Änderung des Passworts

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

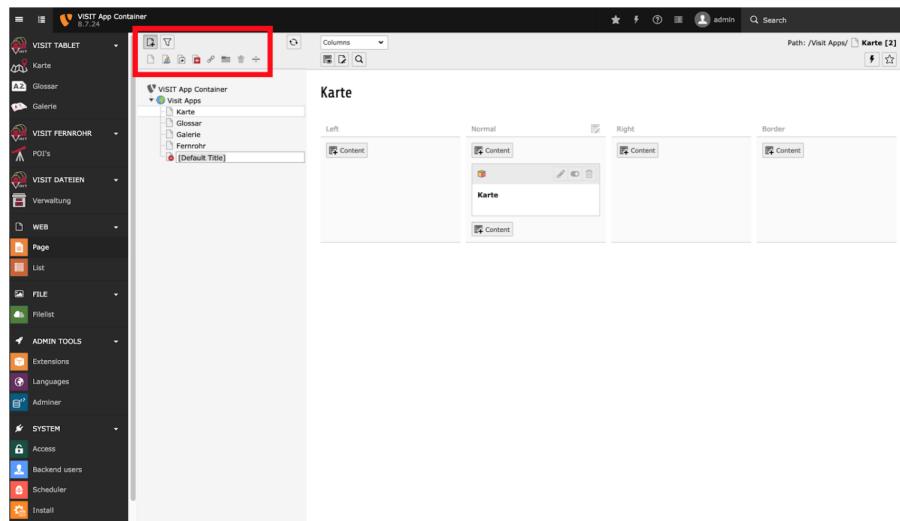


Abbildung 9: Hinzufügen einer neuen Seite

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).

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).

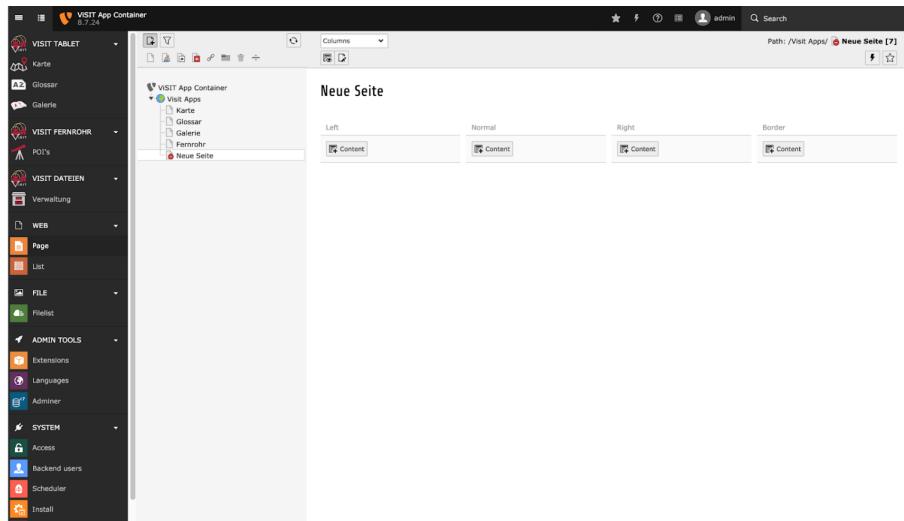


Abbildung 10: Erzeugung des Layouts der neu erstellten Seite

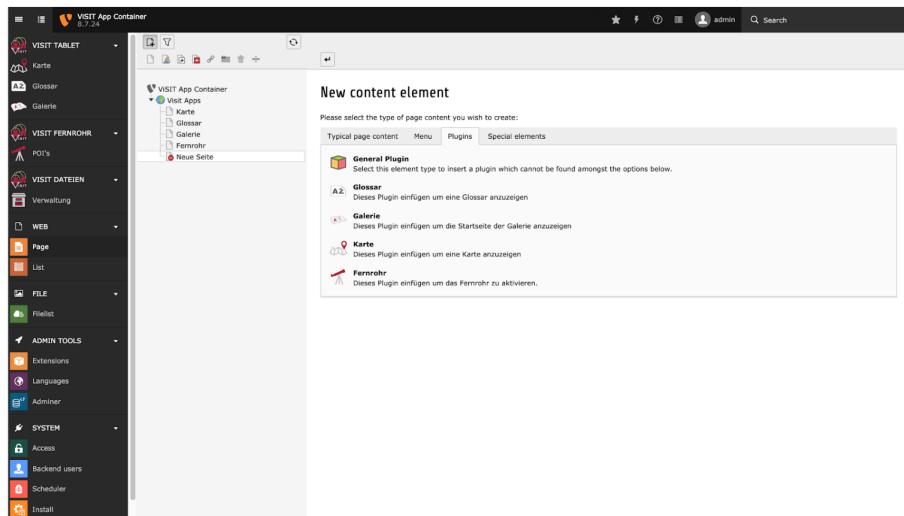


Abbildung 11: Auswahl der Plug-Ins

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.

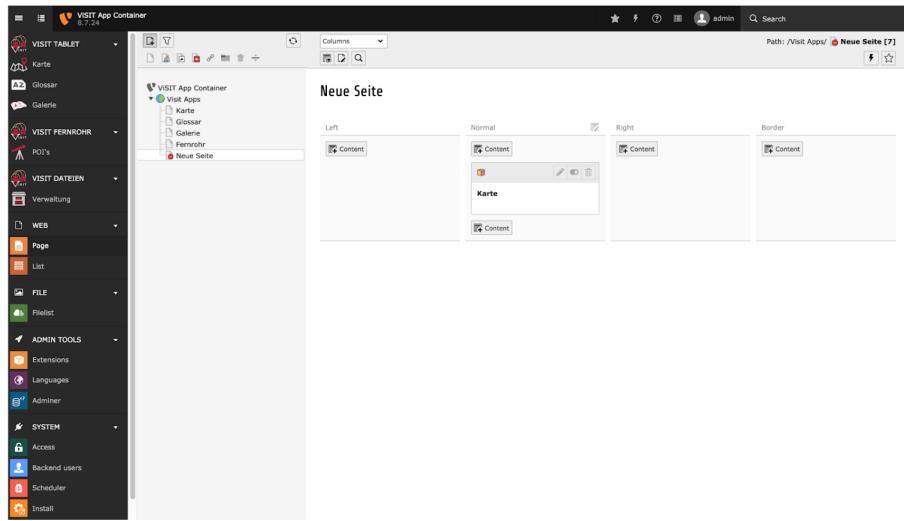


Abbildung 12: Einbindung eines Plug-Ins

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 Werkzeugsleiste 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 Einpflegen 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).

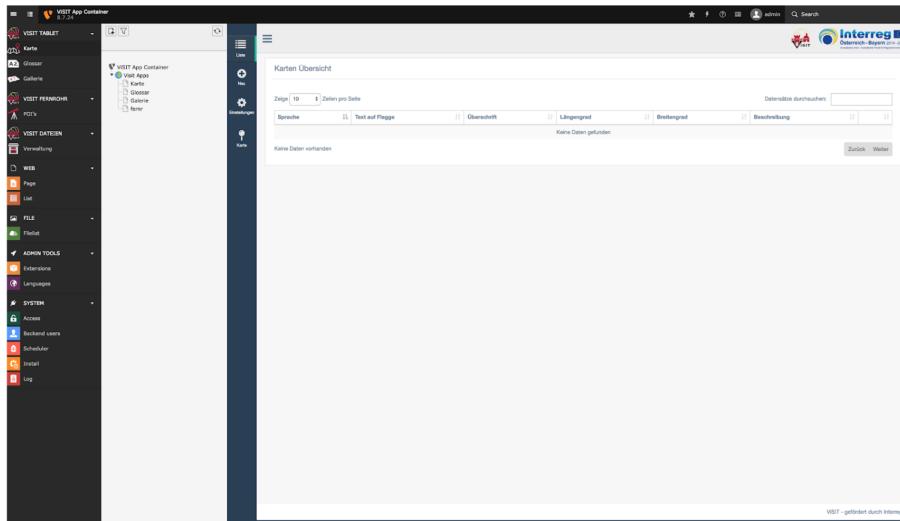


Abbildung 13: Leere Kartenübersicht

4.2 Erstellung der Startseite für die Karten-Applikation

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

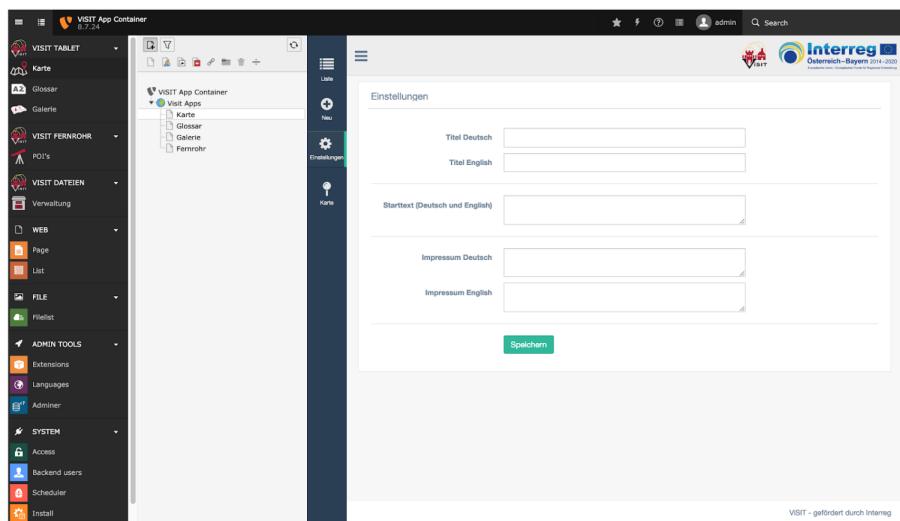


Abbildung 14: Erstellung der Startseite für die Karten-Applikation

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).

```


## Visit Apps



Type3 Extension that contains all applications



### Beispiel Startsseite 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>

 <p>
 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris eget elit a lacus sollicitudin

 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" onclick="initMap('de')"></button>
 <button type="button" class="btn btn-primary lang-btn btn-lg" data-dismiss="modal" onclick="initMap('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
 </p>

 <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"></button>
 <button type="button" class="btn btn-primary" data-dismiss="modal">Schließen</button>
 </div>
</div>

```


```

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

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 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"`.

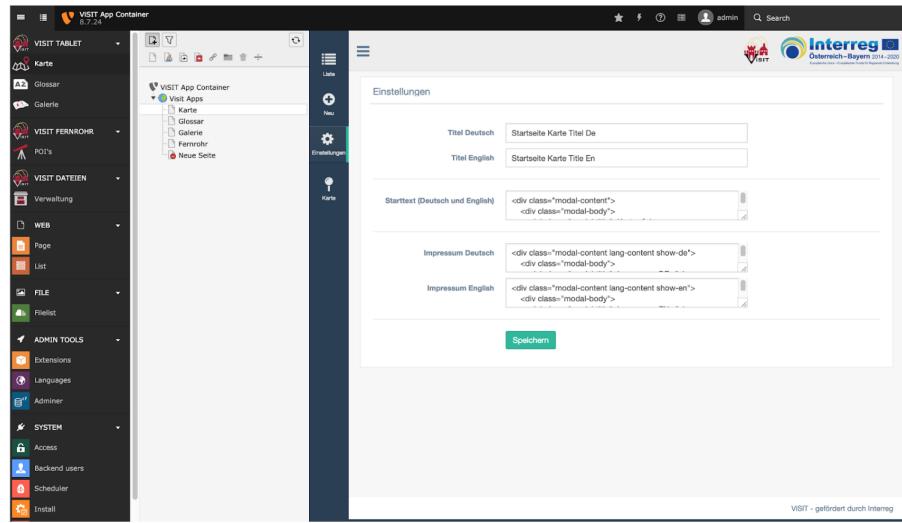


Abbildung 16: Erstellung der Startseite für die Karten-Applikation

4.3 Neues Kartenelement hinzufügen

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

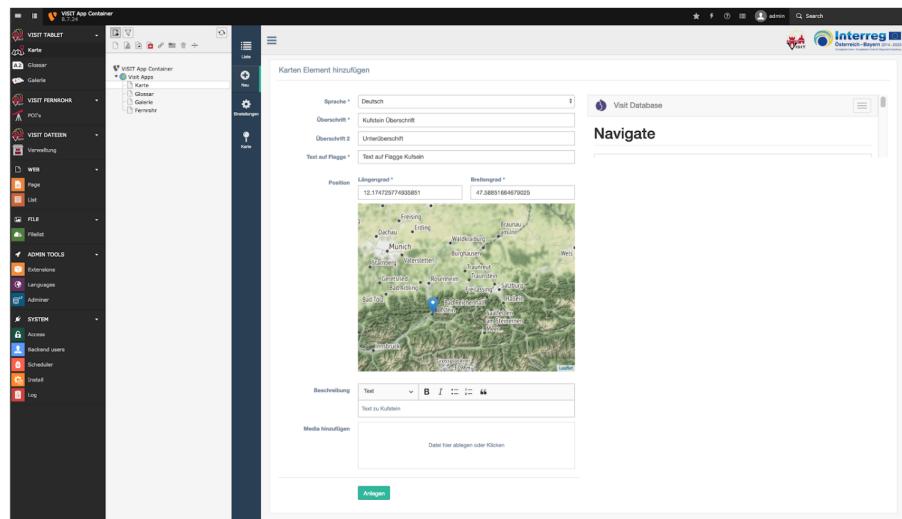


Abbildung 17: Ein neues Kartenelement hinzufügen

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

Sprache	Text auf Flagge	Überschrift	Längengrad	Breitengrad	Beschreibung
Deutsch	Text auf Flagge Kufstein	Kufstein Überschrift	12.174725774936	47.58851664679	Text zu Kufstein
Deutsch	Text auf Flagge Rosenheim	Rosenheim	12.119800071427	47.861958171191	Text zu Rosenheim
English	Text auf Flagge Saalfelden	Saalfelden EN	12.844873537072	47.41409818047	Text zu Saalfelden
English	Text auf Flagge Kufstein	Kufstein EN	12.169236878178	47.588516642268	Text zu Kufstein
English	Text auf Flagge Rosenheim	Rosenheim	12.119800071427	47.858273217776	Text zu Rosenheim

Abbildung 18: Listenansicht über alle angelegten Kartenelemente

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

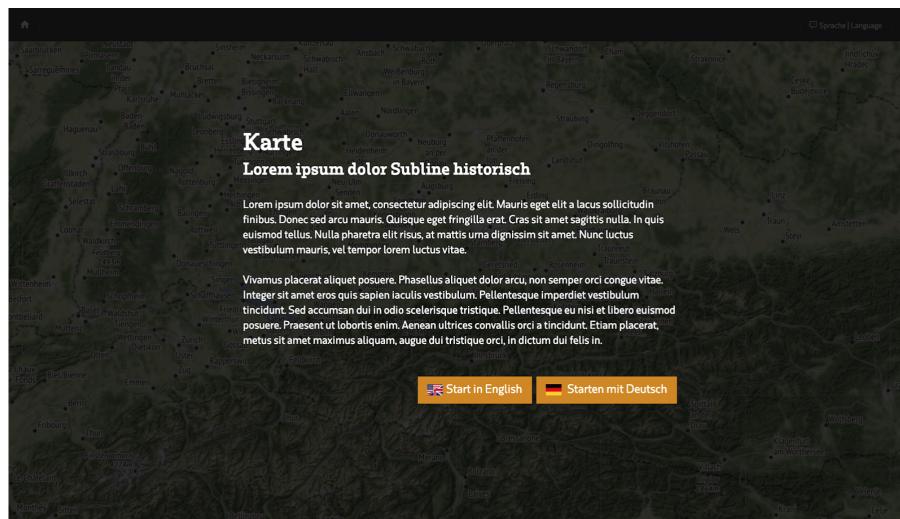


Abbildung 19: Startseite der Karten-Applikation

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).

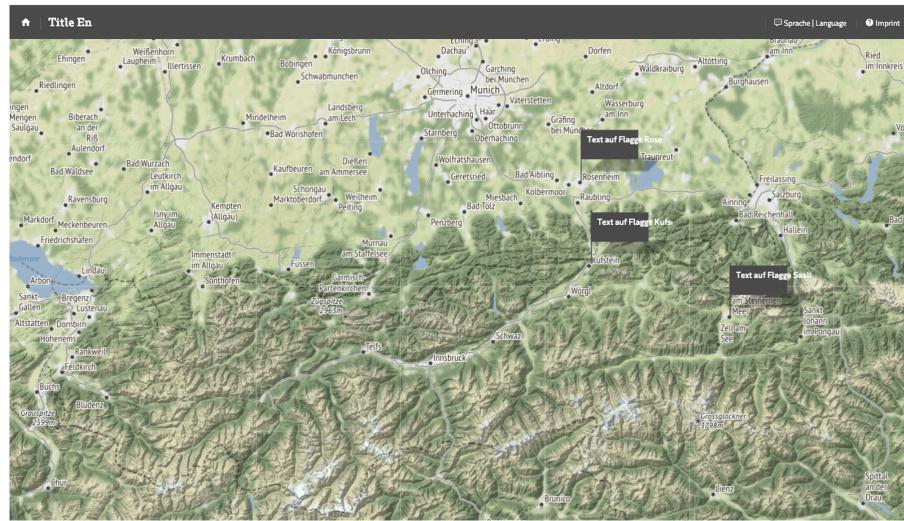


Abbildung 20: Ansicht der Karte im Browser

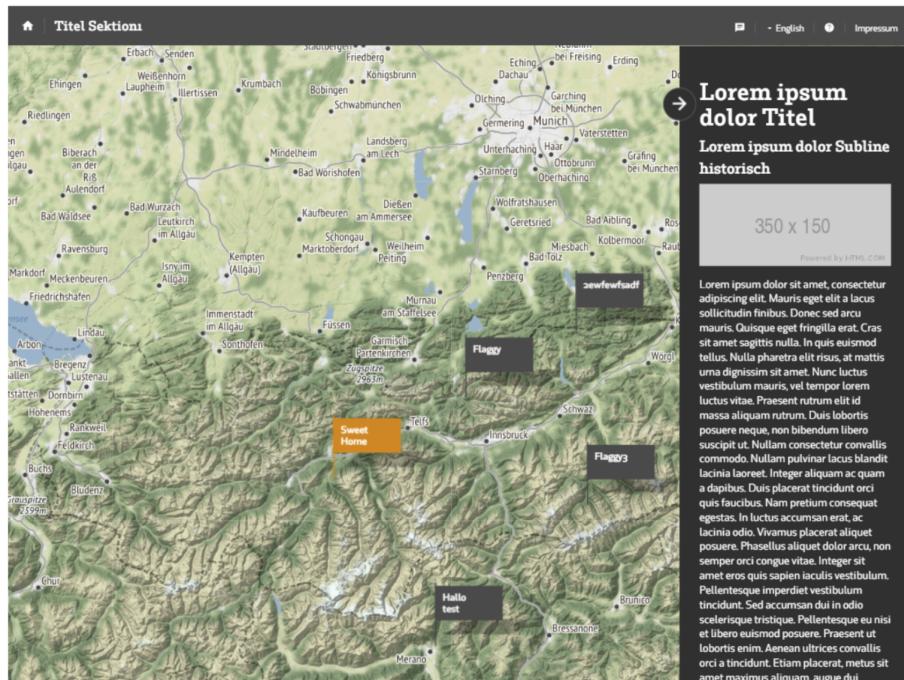


Abbildung 21: Kartenelement - Point of Interest - mit Detailinformation

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.

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.

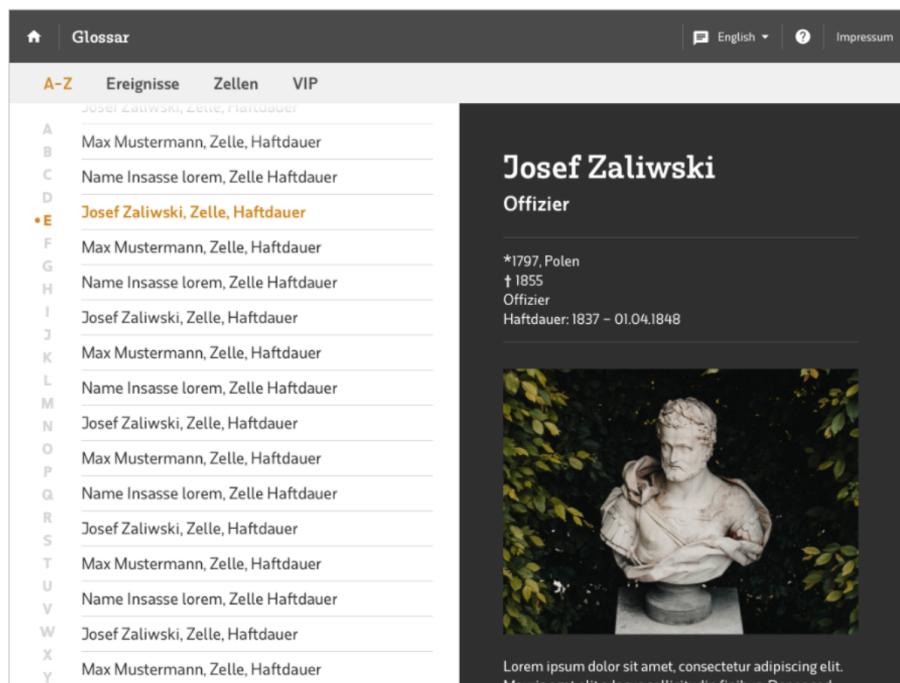


Abbildung 22: Ansicht der Glossar-Applikation auf einem Tablet

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.

5.2 Erstellung der Startseite für die Glossar-Applikation

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

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.

Sprache	Name	Geburtsdatum	Todesdatum	Inhaftiert am	Freigelassen am	Event	Zelle
Deutsch	Aa Insasse 1	01.01.1780	01.01.1810	01.01.1905	01.01.1907	Event 1 DE	Zelle 1
Deutsch	Bb Insasse 2	02.02.1600	02.02.1628	02.02.1602	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	01.01.1905	01.01.1907	Event 1 DE	Zelle 1
English	Bb Prisoner 2	02.02.1600	02.02.1628	02.02.1602	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

Abbildung 24: Konfiguration 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 angesehen 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.

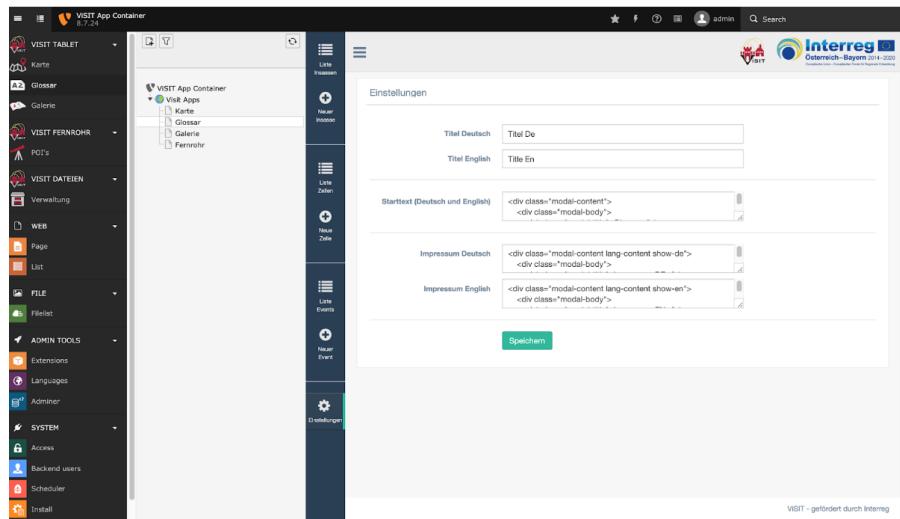


Abbildung 25: Befüllung der Startseite der Glossar-Applikation

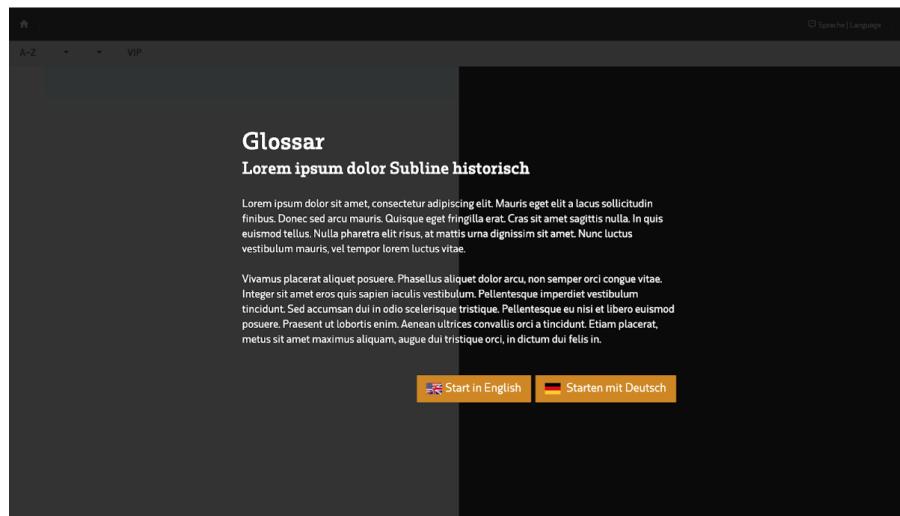


Abbildung 26: Startseite der Glossar-Applikation

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 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.

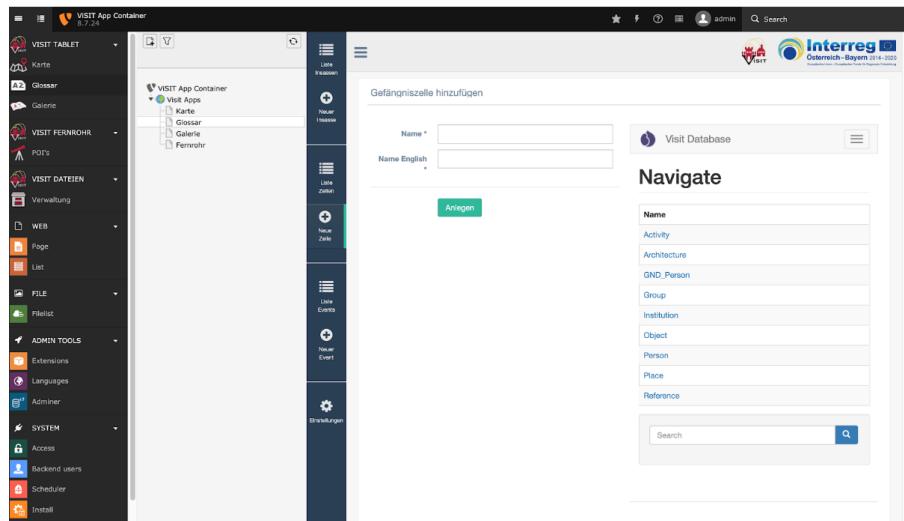


Abbildung 27: Hinzufügen einer neuen Zelle

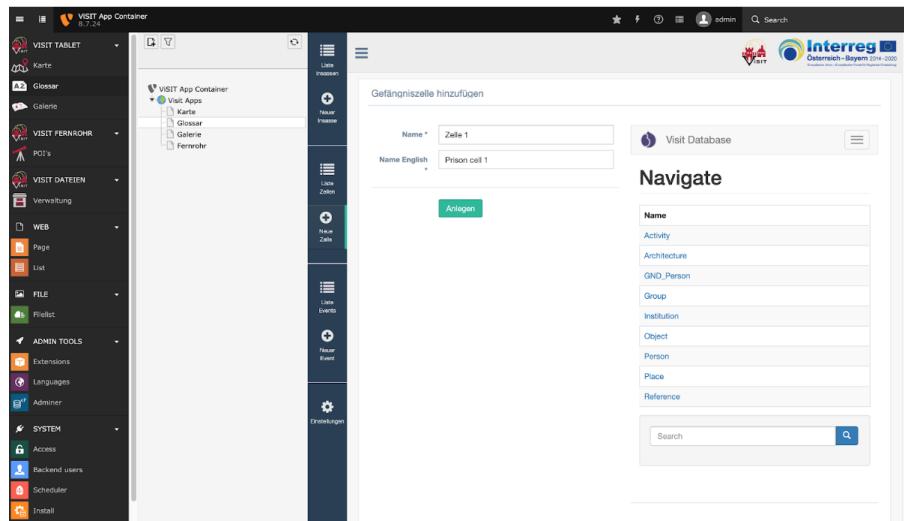


Abbildung 28: Erstellen einer neuen Zelle

5.6 Bearbeitung und Löschung von Insassen

6 GALERIE-APPLIKATION

- 6.1 Einfügen 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 Vi-SIT Medien-Datenbank verwaltet werden können. Die Verwaltung befindet

The screenshot shows the 'Gefängnis Zellen - Übersicht' (Prison Cells - Overview) page. The table lists five entries:

Name	Name En
Zelle 1	Prison cell 1
Zelle 2	Prison cell 2
Zelle 3	Prison cell 3
Zelle 4	Prison cell 4
Zelle 5	Prison cell 5

Abbildung 29: Angelegte Zellen in der Listenübersicht

The screenshot shows the 'Event hinzufügen' (Add Event) form. The 'Name' field is filled with 'Event 1 DE' and 'Name English' with 'Event 1 EN'. The right side features a 'Navigate' sidebar with a search bar and a list of categories.

Abbildung 30: Hinzufügen eines neuen Events

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-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](#).

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

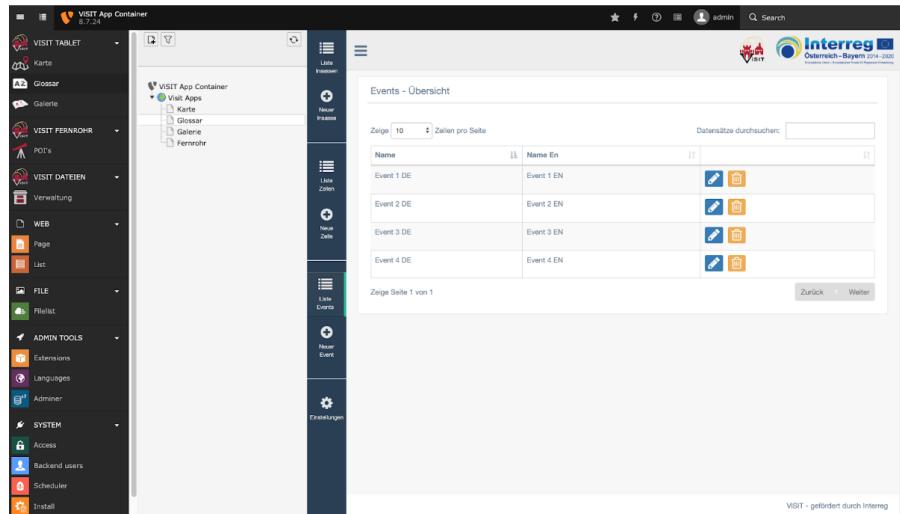


Abbildung 31: Angelegte Events in der Listenansicht

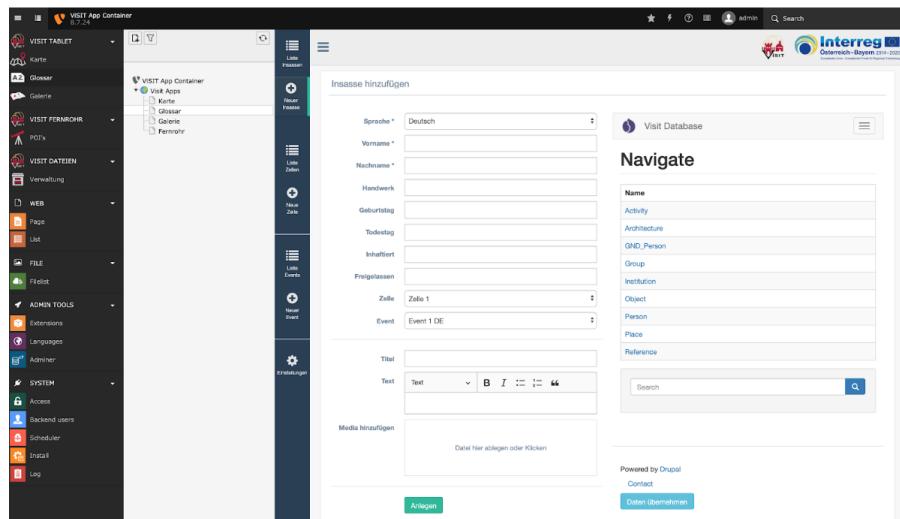


Abbildung 32: Anlegen eines neuen Insassens

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

Abbildung 33: Anlegen eines Insassens

Abbildung 34: Angelegte Insassen in der Listenansicht

Gründe 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.

Die Dateien mit dem Zugriffsmodifikator *public* und *visit* können mit einem

The screenshot shows a web-based application interface for managing inmates. On the left, there's a sidebar with navigation links like 'Karte', 'Glossar', 'Galere', 'VISIT FERNROHR', 'POD's', 'VISIT DATEIEN', 'Verwaltung', 'WEB', 'FILE', 'ADMIN TOOLS', 'SYSTEM', and 'Scheduler'. The main content area has a header 'Titel De' and sub-links 'A-Z', 'Zellen', 'Ereignis', and 'VIP'. Below this, a table lists inmates with columns 'Name' and 'Details'. The first row shows 'Aa Insasse 1' with details: 'Titel Insasse 1', 'Bb Insasse 2', and 'Cc Insasse 3'. To the right of the table is a detailed view of 'Aa Insasse 1' with fields for 'Vorname', 'Nachname', 'Geburtsdatum', 'Handwerk', and a 'Text zum Insassen 1' section.

Abbildung 35: Ansicht der Insassen im Browser

This screenshot shows the 'ViSIT App Container' interface. The left sidebar includes 'Karte', 'Glossar', 'Galere', 'VISIT FERNROHR', 'POD's', 'VISIT DATEIEN', 'Verwaltung', 'WEB', 'FILE', 'ADMIN TOOLS', 'Extensions', 'Languages', 'Admin', 'SYSTEM', 'Scheduler', and 'Install'. The main panel displays a list of extensions with columns for name, version, status, type, and actions. One extension, 'visit_tablet', is highlighted. The URL in the browser bar is `localhost:8080/typo3/index.php/M+tools_ExtensionmanagerExtensionmanagerModule%7eview-c983cb5a5d0-d8bfef27fc2bbcb3cb0d8c1c1d446b_extensionsmanager%7bextension%45..`

Abbildung 36: Einstellung der ViSIT App Extension

This screenshot shows the configuration page for the 'visit_tablet' extension. The left sidebar is identical to Abbildung 36. The main area is titled 'Configure Extension visit_tablet' and contains several configuration fields:

- Main** tab: 'Teilnehmer Name' (set to 'visit.partnername (string)'), 'API User' (set to 'visit.apiluser (string)'), 'Password for API User' (set to 'visit.apiluserPassword (string)'), 'Syncing Master ID' (set to 'visit.syncingMasterId (string)'), 'Medienobjekte komprimieren' (checkbox checked), and 'Port des Komprimierungscontainers' (set to 'visit.compressionApiPort (integer)' with value '8899').
- Debug** tab: Not visible in the screenshot.

Abbildung 37: ViSIT App Extensions

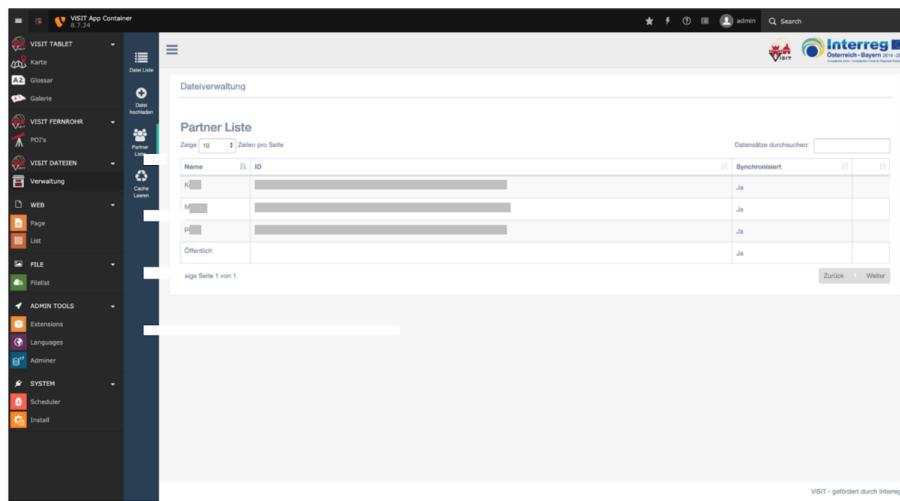


Abbildung 38: Ansicht der ViSIT-Partner mit Zugang zur Mediendatenbank im Peer-to-Peer-Netzwerk

Klick auf das blaue Downloadsymbol in der entsprechenden Zeile heruntergeladen werden.

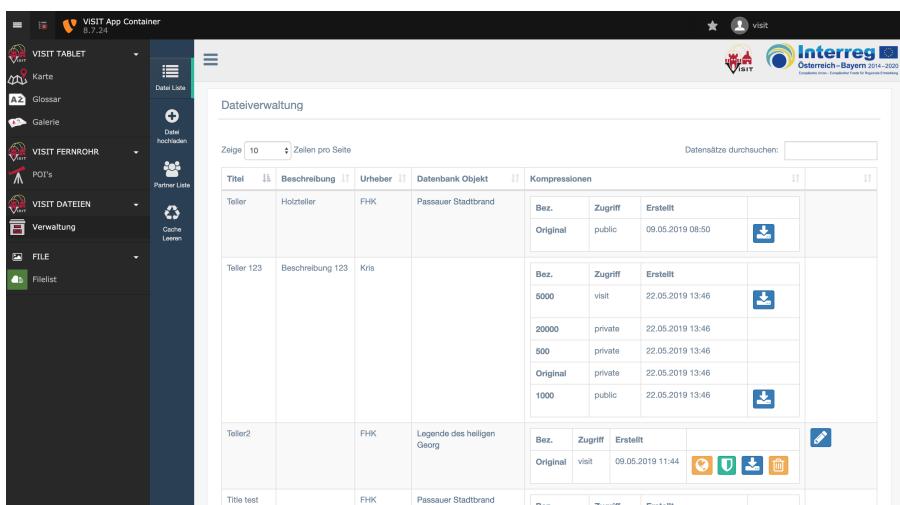


Abbildung 39: Ansicht der bereits verfügbaren Dateien in der Dateiliste

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 die-

sem 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.

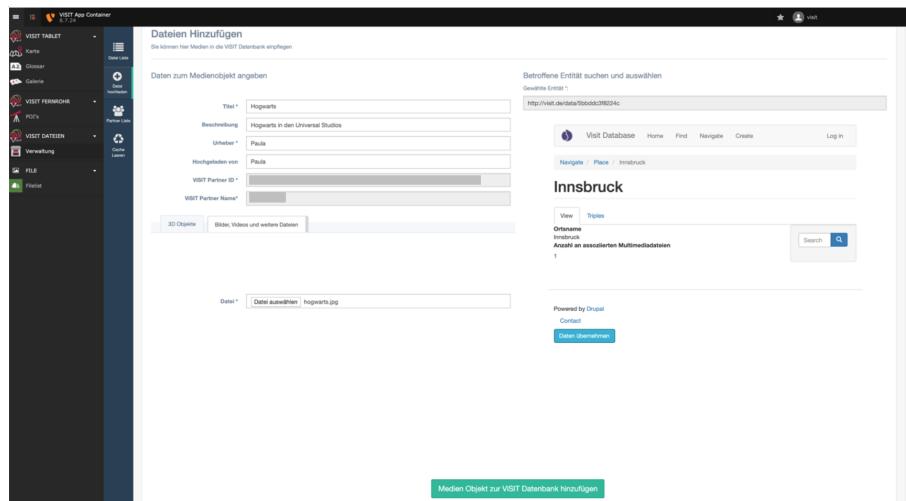


Abbildung 40: Ansicht des ausgefüllten Formulars für den Dateiupload

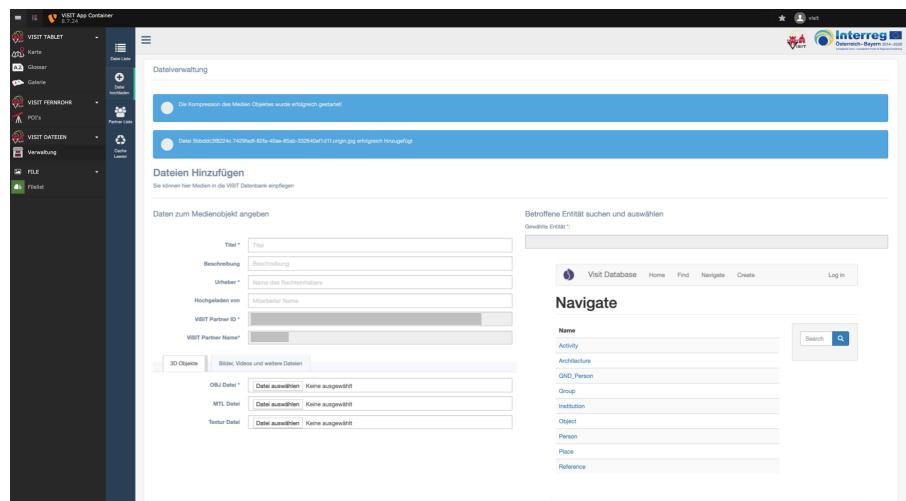


Abbildung 41: Bestätigungs Nachrichten nach einem erfolgreichen Upload

Nachdem die Datei in der ViSIT Datenbank gespeichert wurde, kann sie in der Dateiliste angesehen werden 42. Der Zugriff auf die hochgeladene

Titel	Beschreibung	Urheber	Datenbank Objekt	Kompressionen	Bez.	Zugriff	Erstellt	
Hogwarts	Hogwarts in den Universal Studios	Paula			Original	private	04.05.2019 13:54	
Teller	Holzteller	FHK	Passauer Stadtbrand		Bez.	Zugriff	Erstellt	
					Original	public	09.05.2019 08:50	
Teller 123	Beschreibung 123	Kris			Bez.	Zugriff	Erstellt	
					5000	visit	22.05.2019 13:48	
					20000	private	22.05.2019 13:48	
					500	private	22.05.2019 13:48	
					Original	private	22.05.2019 13:48	
					1000	public	22.05.2019 13:48	
Teller2		FHK	Legende des heiligen Georg		Bez.	Zugriff	Erstellt	
Title test		FHK	Passauer Stadtbrand		Original	visit	09.05.2019 11:44	

Abbildung 42: Hochgeladene Datei in der Listenübersicht

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

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 Bedienung, die über eine Web-Oberfläche erfolgt, eingegangen. Abschließend wird die Schnittstelle, 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

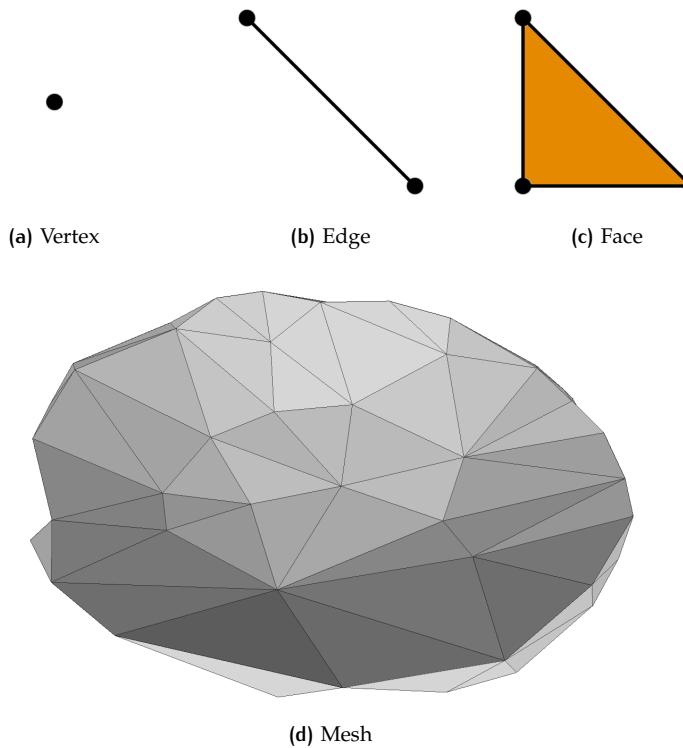


Abbildung 43: Grundlegende Elemente der Geometrie eines 3D-Modells

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.

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-Format, für jedes Face die Koordinaten eines jeden Eckpunkts separat gespeichert werden, wird die dadurch verursachte Redundanz zum Beispiel bei OBJ-Dateien 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.

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 [TODO] dargestellt ist, werden zwei durch ein Edge verbundene Vertices zu einem neuen Vertex verschmolzen. Dabei degenerieren die an dieses Edge angren-

zende 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 geringsten 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-Komprimierung [TODO Referenz]

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 null bis eins liegen, wobei sich null auf den oberen bzw. linken Rand und eins auf den unteren bzgl. rechten Rand der Texturdatei be-

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 1: Überblick über alle Konfigurationsmöglichkeiten der Kompressionskomponente

zieht. 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 Installation (?)

9.4 Konfiguration

Beim Starten des Kompressions-Systems wird automatisch eine Konfigurationsdatei (TODO wo?) mit den Standardwerten angelegt, sofern eine solche nicht bereits 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 lesen 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-Dokumentation verwiesen. Tabelle 1 gibt Aufschluss über alle Konfigurationsmöglichkeiten, welche in den folgenden Abschnitten detailliert erläutert werden.

[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))

9.4.1 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.

LÄNGE DER ARCHIV-ANZEIGE Über die Konfigurationsoption `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.

9.4.2 3D-Kompression

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

STANDARD-KOMPRESSIONSSTUFEN 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 22 aufgeführten Standardwert

```
1 [500, 1000, 5000, 20000, 50000, 200000, 500000, 2000000, 5000000, 20000000, 50000000]
```

Listing 22: 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 einem Mesh 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 Kosten 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 Schwellwert-Liste `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 2 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.

9.4.3 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 Konfigu-

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

Tabelle 2: Beispiele für die sich bei unterschiedlichen Vertexanzahlen ergebenden Texturauflösungen bei der Standardkonfiguration.

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

Tabelle 3: Pro Kompressionsstufe notwendige Parameter bei der Bildkompression

rationsoption `imageCompressionLevels` festgelegt. Aufgrund der Komplexität dieses Parameters muss dieser im JSON-Format angegeben werden. Der Wert der Konfigurationsoption muss einem Array aus Objekten entsprechen, wobei jedes Objekt die in Tabelle 3 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 23 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 23: Standardwert für die Konfigurationsoption `imageCompressionLevels`

9.4.4 Schnittstelle Meta-Datenbank

Um die während des Kompressionsvorgangs erstellen Modelle im ViSIT-System zu registrieren, müssen die zu der Mediendatei gehörigen technischen Metadaten [TODO Wo werden diese definiert?] 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*-Verfahren. 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 24 dargestellt.

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

Listing 24: Standardwert für die Konfigurationsoption `metadbApiAuthString`

ENDPUNKT ZUM ABRUFEN DER METADATEN In der Konfigurationsoption `metadbEndpointFetchUrl` 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 In der Konfigurationsoption `metadbEndpointSendUrl` kann der Endpunkt der API zur Meta-Datenbank spezifiziert werden, über den technische Metadaten gespeichert werden können. Dieser Wert ist standardmäßig festgelegt auf <https://DOMAIN/metadb-rest-api/digrep/media> und stimmt in der Regel mit dem Wert für `metadbEndpointFetchUrl` überein.

PRÄFIX DER MEDIEN-UIDS Jede in der Meta-Datenbank registrierte mediale Repräsentation wird durch eine eindeutige 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 `http://:DOMAIN/metadb/` muss vor dem ersten Start der Kompressionskomponente geeignet angepasst werden.

9.5 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 25 dargestellten Zeichenfolge die Startseite der Kompressions-Komponente aufgerufen werden.

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

Listing 25: 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 Einträge in der linken Seitenleiste bzw. über das Aufklapp-Menü auf der linken Seite (bei kleinen Bildschirmen) navigiert werden.

9.5.1 *Startseite*

9.5.2 *Neuer Auftrag*

9.5.3 *Archiv*

9.5.4 *Einstellungen*

9.6 Zugriff über die API

[TODO Infos / Einschränkungen aus Anforderungs-Dokument] [TODO Generelle Funktionsweise, insbesondere Technische Metadaten]

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** [MM04], 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 Anwendungsfällen, 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 44](#) zu sehen ist.

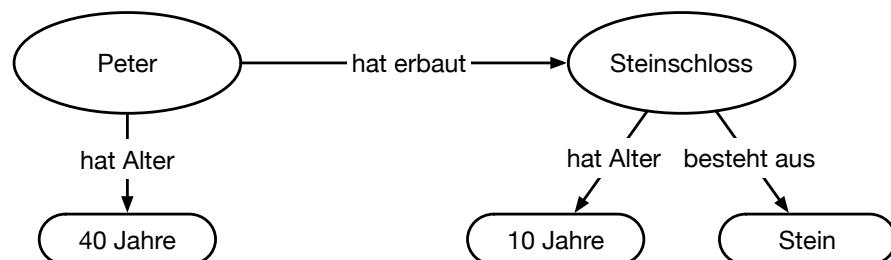


Abbildung 44: Informationen aus obigen Aussagen, kombiniert als Graph.

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 Zusammenhänge der persistierten Objekte. Dazu folgendes (frei erfundenes) Beispiel: Ausgehend von der eigenen Wissensbasis, die die Daten aus [Abbildung 44](#) 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 45](#) visualisiert.

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 Steinschlusses 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 "Steinschlusses" 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

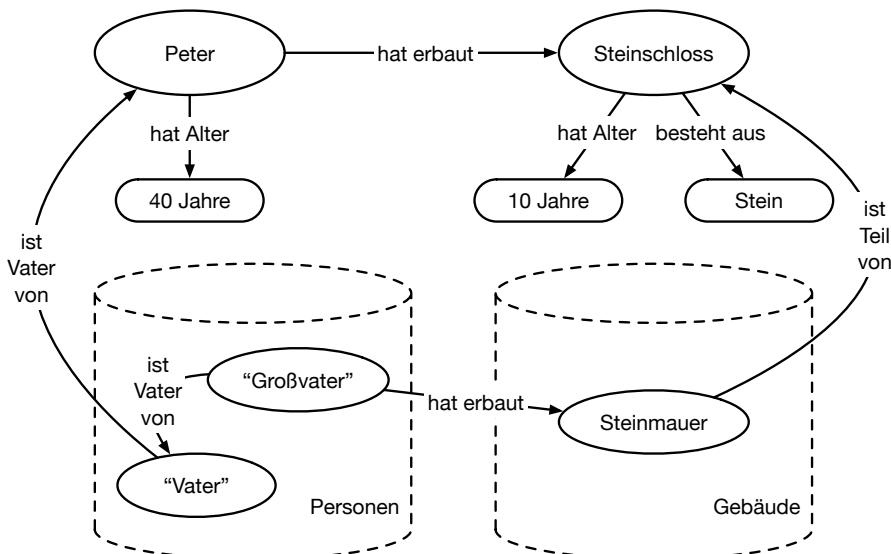


Abbildung 45: Grundlegende eigene Wissensbasis (oben), erweitert um zwei externe Wissensbasen (unten).

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** [Doe03] 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 Group und der CIDOC CRM SIG entwickelt, welche beide Arbeitsgruppen von CIDOC sind. Das CIDOC CRM ist 2000 als "Working Draft" bei der ISO/TC46/SC4 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 6.2, 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 6.2.3.

DAS VISIT MODEL – VISM0 Aufbauend auf dem CIDOC CRM wurde eine Ontologie entwickelt, die den kompletten Anwendungsfall des ViSIT Pro-

<http://network.icom.museum/cidoc/working-groups/overview/>
<http://network.icom.museum/cidoc/working-groups/crm-special-interest-group/>
<http://network.icom.museum/cidoc/>
<https://www.iso.org/committee/48798.html>
<http://www.cidoc-crm.org/Version/version-6.2>
<http://www.cidoc-crm.org/Version/version-6.2.3>

jekt 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 physi- schen Systemen, analog zum "E5_Event" des CIDOC CRM.

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

GRUPPE (GROUP): Diese Entität bezeichnet mehrere Personen, die sich zu einer Gruppierung zusammengeschlossen haben und durch eine gleiche oder ähnliche Tätigkeit miteinander verbunden sind (vgl. "E74_Group" 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 Ausstellungsob- jekten aus den Sammlungen von musealen oder museumsähnlichen Institutionen, Archiven etc., die wie Bauwerke als Informationsträ- ger betrachtet werden (vgl. "E84_Information_Carrier" des CIDOC CRM).

PERSON (PERSON): Diese Entität bezeichnet analog zur Definition im CI- DOC CRM der "E21_Person" alle natürlichen Personen, die leben oder bereits verstorben sind sowie Personen, von denen angenommen wird, dass sie lebten. Dazu zählen historische Persönlichkeiten als auch Per- sonen 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 niederge- schriebenen 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 Klas- sen 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 46](#) visua- lisiert den Entwicklungsprozess hinter VisMo.

<http://www.cidoc-crm.org/Entity/E5-Event/Version-6.2>
<http://www.cidoc-crm.org/Entity/E84-Information-Carrier/Version-6.2>
<http://www.cidoc-crm.org/Entity/E74-Group/Version-6.2>
<http://www.cidoc-crm.org/Entity/E84-Information-Carrier/Version-6.2>
<http://www.cidoc-crm.org/Entity/E21-Person/Version-6.2>

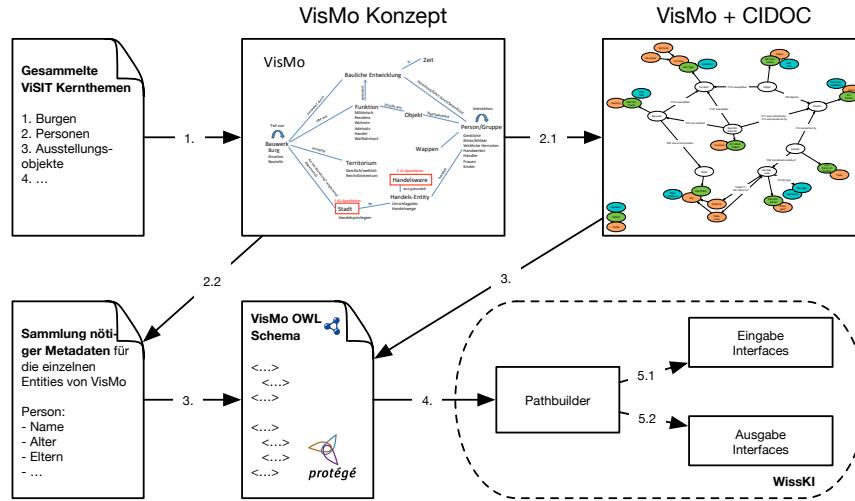


Abbildung 46: Arbeitsprozess hinter der Entwicklung des ViSIT Modells.

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 **protégé** zusammengeführt werden, um eine RDF/OWL Ontologie zu erstellen. Diese ist in ihrer letzten offiziellen Version in [Listing 29](#) im Appendix zu sehen.

Ebenfalls ist in [Abbildung 46](#) 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 Ausgabemaschen 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.

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**

ty 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 47](#) zu sehen.

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

HAPROXY Dem virtuellen Server für die ViSIT Infrastruktur ist ein **haproxy** 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 hintergründig 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** 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 diese 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

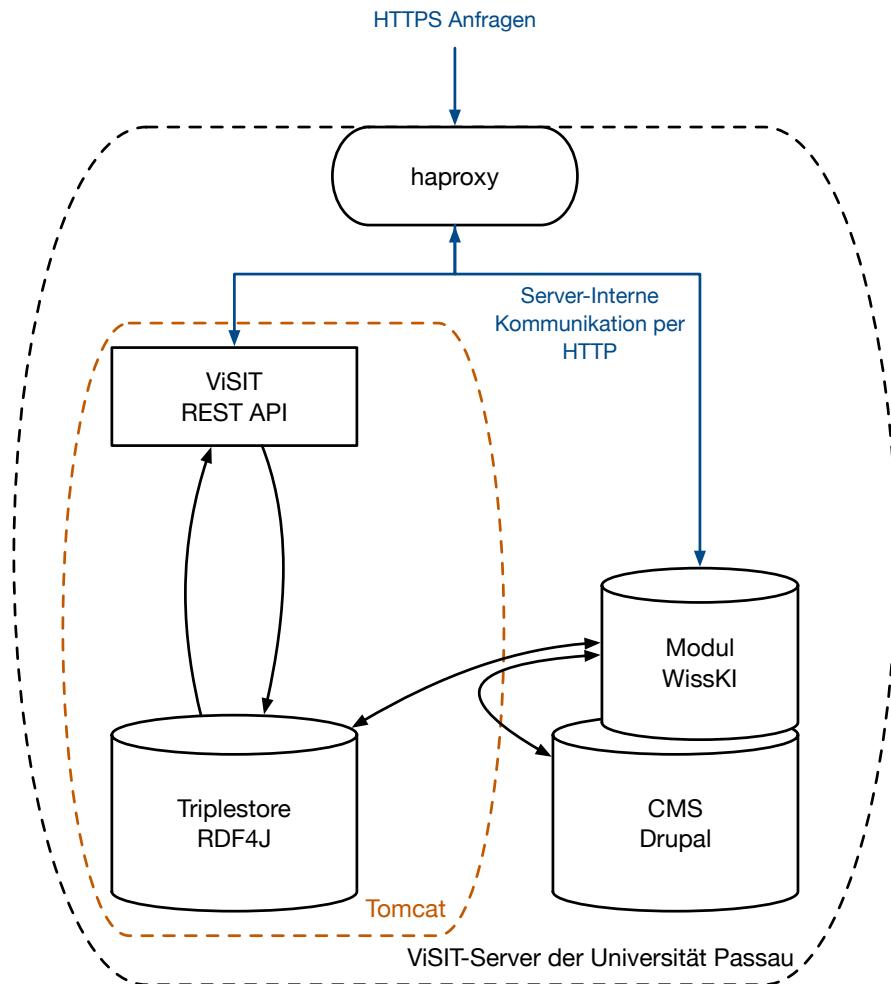


Abbildung 47: Technische Infrastruktur der Semantischen Datenbank des ViSIT Projekts.

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 Update 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 **RDF4J** 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 bewerk-

stelligt. 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 **Drupal** und dessen Modul **WissKI - Wissenschaftliche KommunikationsInfrastruktur**. 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.

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 56](#) und [Abbildung 57](#) im Appendix zeigen die Konfigurationsmöglichkeiten eines WissKI Salz Adapters, bzw. die Einstellungen die für ViSIT getätigter 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.

WRITABLE 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 Triplestore, 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 unterliegenden 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 58](#) 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 59](#) 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 wiederspiegeln 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 48](#) 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]	Unlimited	Edit
Object_identifiedBy_Title	">http://visit.de/ontologies/vismo/Object -> ecrm:P1_is_identified_by -> ecrm:E35_Title	Text (plain)	Unlimited

Abbildung 48: Zwei Beispiel-Pfade aus der WissKI Konfiguration des ViSIT Projekts.

Der zweite Pfad in [Abbildung 48](#) 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 48](#) 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 49](#) 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”.

The screenshot shows the configuration interface for a path. At the top, there's a 'Name' field with 'Object_identifiedBy_Title' and a note 'Machine name: object_identifiedby_title'. Below it, 'Path Type' is set to 'Path'. A question 'Is this Path a group?' has a dropdown with 'please select'. A status bar says '► REASONER HAS RUN. CACHE IS PREPARED'. The main area is divided into 'STEP' and 'EDIT' columns. The 'STEP' column lists the path components: 'Object' (dropdown), 'http://erlangen-crm.org/170309/P1_is_identified_by' (dropdown), 'http://erlangen-crm.org/170309/E35_Title' (dropdown), and 'please select' (dropdown). The 'EDIT' column contains dropdowns for each step. At the bottom, there's a 'Datatype Property' dropdown with 'http://erlangen-crm.org/170309/P3_has_note' selected, a 'Disambiguation Point' dropdown with 'Concept 2: http://erlangen-crm.org/170309/E35_Title' selected, and a 'Save' button.

Abbildung 49: Erste Maske zum Editieren eines WissKI Pfads.

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 50](#) 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.

[Abbildung 51](#) 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 59](#) 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 somit 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

Pathbuilder *

Name of the pathbuilder.

Path *

Name of the path.

CHOOSE FIELD

Select an existing field from bundle *Object (b15d6693d3ba884f733c2cce4cd6b015)*

ID of the mapped Field.

Type of the field that should be generated. *

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

Type of form display for field

Widget for the Field – If there is any.

Type of formatter for field

Formatter for the field – If there is any.

Cardinality

Entity reference

Save **Delete**

Abbildung 50: Zweite Maske zum Editieren eines WissKI Pfads.



Abbildung 51: Beispiel-Pfad aus der WissKI Konfiguration des ViSIT Projekts für eine Relation zwischen zwei Entitäten der Datenbank.

“eine Ebene nach rechts” geschoben wird, wie dies im Beispiel für die ersten beiden Pfade zu sehen ist (ebenfalls in Abbildung 48 zu sehen). Eine Untergruppe zum Objekt bildet zum Beispiel der Pfad mit der ID “0bject_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 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 49](#) 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 60](#) im Appendix) ist unter anderem das Eingabe-Interface für die Ausstellungsobjekte (Object) zu erreichen, dieses ist in [Abbildung 61](#) 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 Partiane aus dem ViSIT Projekt ist in [Abbildung 62](#) 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 Framework 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 52](#), zugrunde:

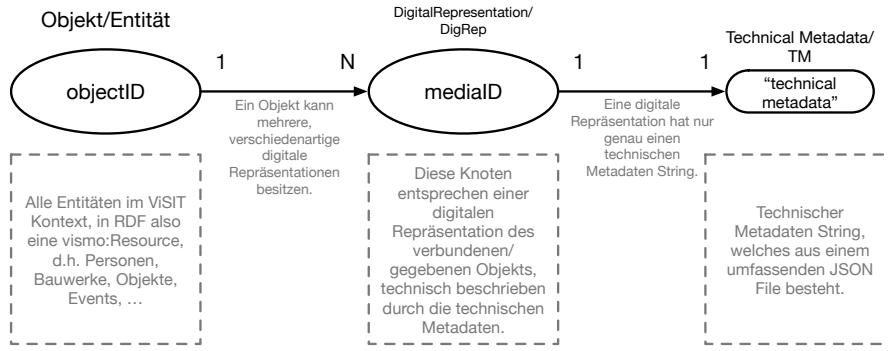


Abbildung 52: Modell der digitalen Repräsentationen.

Die möglichen API Funktionen sind, in [Tabelle 4](#) die folgenden (die API Pfade sind relativ zur jeweiligen REST API angegeben):

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 4: API Funktionen für die Digital Representations.

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 30](#) 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 5](#):

Wichtig zu wissen beim Auslesen der Objekte ist, dass die angesprochene ID der API die RDF ID des Knotens der entsprechenden En-

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

Tabelle 5: API Funktionen zum Auslesen von Objekten.

tität ist (z.B. "http://visit.de/data/5c4ae02a2d569"), 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" 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.

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.

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

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 downloadet werden. [Abbildung 53](#) zeigt diese Funktionalität mit dem “Create Exportfile” Button.

The screenshot shows the WissKI Pathbuilder interface with the following sections:

- Pathbuilder Definition Import:** A section for importing pathbuilder definitions. It includes a text input field for the path to a definition file and a dropdown menu for setting default mode to "Keep settings from import file".
- Set default mode to:** A dropdown menu set to "Keep settings from import file". Below it is a question: "What should the fields and groups mode be set to?" followed by an "Import" button.
- EXPORT TEMPLATES:** A list of generated export files, including:
 - rdf4jpathbuildertest_20180315T204613
 - visitrepo_pathes_20180808T170845
 - visitrepo_pathes_20180905T182510
 - visitrepo_pathes_20181128T104132
 - visitrepo_pathes_20190107T110029
 - visitrepo_pathes_20190109T143959
 - visitrepo_pathes_20190123T151843
 - visitrepo_pathes_20190131T143855
- Create Exportfile:** A button at the bottom of the export list, highlighted with a red border.
- Save and generate bundles and fields:** A blue button at the bottom left.
- Delete:** A link at the bottom right.

Abbildung 53: WissKI Pathbuilder Ansicht, die den Download der Pfad-Datei anbietet.

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 26: 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 Repository 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:

- 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 54](#) zu sehen. Wichtig ist, dass Zeile 4 **nicht** kommentiert ist.

```
<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 54: pom.xml Konfiguration zum produktiven Deployment der REST API.

- 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 55](#) 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.

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

```

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

10.6 Zusatzfeatures - Erweiterung der Bedienbarkeit der Semantischen Datenbank

copy and paste feature
excel importer

10.7 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:

```
1 | drush cr
```

Listing 27: Befehl für einen Cache Flush eines Drupal Systems.

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:

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

Listing 28: Befehl zum Updaten 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 Vi-SIT 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.

11 SCHLUSS

blub

12 APPENDIX

```

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

```

```

74      <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/ReferenceEntry"/>
75      <rdfs:range rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
76      <rdfs:comment>Reference from a vismo:ReferenceEntry to a vismo:Resource, associating
77      the describing nature of the associated vismo:Reference.</rdfs:comment>
78      <rdfs:label>entry is about</rdfs:label>
79      </owl:ObjectProperty>
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
89          illustrates the 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
102         in the ViSIT context) to a digital representation of it, e.g. a picture that shows the
103         respective 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
125          <rdfs:label>interaction target</rdfs:label>
126      </owl:ObjectProperty>
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
134          interstation for the 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
146      vismo:Resource.</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,
159          indicating the given city is part of a trade route and therefore its associated trade.</
160          rdfs:comment>
161          <rdfs:label>part of trade route</rdfs:label>
162      </owl:ObjectProperty>
163
164
165      <!-- http://visit.de/ontologies/vismo/reference -->
166
167      <owl:ObjectProperty rdf:about="http://visit.de/ontologies/vismo/reference">
168          <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topObjectProperty"/>
169          <owl:inverseOf rdf:resource="http://visit.de/ontologies/vismo/referencedBy"/>
170          <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
171          <rdfs:range rdf:resource="http://visit.de/ontologies/vismo/Reference"/>
172          <rdfs:comment>Issues that the referenced vismo:Reference contains further and
173          descriptive information about the given vismo:Resource entity.</rdfs:comment>
174          <rdfs:label>reference</rdfs:label>
175      </owl:ObjectProperty>
176
177
178      <!-- http://visit.de/ontologies/vismo/referencedBy -->
179
180      <owl:ObjectProperty rdf:about="http://visit.de/ontologies/vismo/referencedBy">
181          <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topObjectProperty"/>
182          <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Reference"/>
183          <rdfs:range rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
184          <rdfs:comment>Refers to the vismo:Resource entities that reference this
185          vismo:Reference and therefore this entity contains further and descriptive information
186          about the resource entities.</rdfs:comment>
187          <rdfs:label>referenced by</rdfs:label>
188      </owl:ObjectProperty>
189
190
191      <!-- http://visit.de/ontologies/vismo/referencedByEntry -->
192
193      <owl:ObjectProperty rdf:about="http://visit.de/ontologies/vismo/referencedByEntry">
194          <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topObjectProperty"/>
195          <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
196          <rdfs:range rdf:resource="http://visit.de/ontologies/vismo/ReferenceEntry"/>
197          <rdfs:comment>Reference from a vismo:Resource to a given vismo:ReferenceEntry,
198          indicating that the associated vismo:Reference contains information about the former.</
199          rdfs:comment>
200          <rdfs:label>referenced by entry</rdfs:label>
201      </owl:ObjectProperty>
202
203
204      <!-- http://visit.de/ontologies/vismo/representsDigitally -->
205
206      <owl:ObjectProperty rdf:about="http://visit.de/ontologies/vismo/representsDigitally">
207          <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topObjectProperty"/>
208          <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/DigitalRepresentation"/>
209          <rdfs:range rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
210          <rdfs:comment>Refers from a given digital representation (a picture, 3D model, etc.)
211          back to the vismo:Resource that it originally represents.</rdfs:comment>
212          <rdfs:label>represents digitally</rdfs:label>
213      </owl:ObjectProperty>
214
215      <!-- http://visit.de/ontologies/vismo/routeLocation -->
216
217      <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/Traderroute"/>
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
220   vismo:City 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 traderroute to the vismo:City that represents the starting
230   point for the route.</rdfs:comment>
231   <rdfs:label>start location</rdfs:label>
232 </owl:ObjectProperty>

233
234
235 <!--
236 ///////////////////////////////////////////////////
237 //
238 // Data properties
239 //
240 ///////////////////////////////////////////////////
241 -->

242
243
244
245 <!-- http://visit.de/ontologies/vismo/buildingHistory -->
246
247 <owl:DatatypeProperty rdf:about="http://visit.de/ontologies/vismo/buildingHistory">
248   <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topDataProperty"/>
249   <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Architecture"/>
250   <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
251   <rdfs:comment>Property used to (freely) describe the building history of a
252   vismo:Architecture entity.</rdfs:comment>
253   <rdfs:label>building history</rdfs:label>
254 </owl:DatatypeProperty>

255
256
257 <!-- http://visit.de/ontologies/vismo/comment -->
258
259 <owl:DatatypeProperty rdf:about="http://visit.de/ontologies/vismo/comment">
260   <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topDataProperty"/>
261   <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
262   <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
263   <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">This is a
264   comment about a given vismo entity.</rdfs:comment>
265   <rdfs:isDefinedBy rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://visit.
266   de/ontologies/vismo</rdfs:isDefinedBy>
267   <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">comment</
268   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
278   defines a (historical) description for a vismo entity.</rdfs:comment>
279   <rdfs:isDefinedBy rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://visit.
280   de/ontologies/vismo/</rdfs:isDefinedBy>
281   <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">description</
282   rdfs:label>
283 </owl:DatatypeProperty>

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
303      that 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
310    <owl:DatatypeProperty rdf:about="http://visit.de/ontologies/vismo/iconography">
311      <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topDataProperty"/>
312      <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
313      <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
314      <rdfs:comment>A property to associate iconography to a given vismo:Resource entity.</
315      rdfs:comment>
316      <rdfs:label>iconography</rdfs:label>
317    </owl:DatatypeProperty>
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 refer to more general topics that can be addressed to anything out of the VisMo
domain, for example "Trade", "War", "Peace", or overall
temporal associations.</rdfs:comment>
      <rdfs:label>keyword</rdfs:label>
339    </owl:DatatypeProperty>
340
341
342
343    <!-- http://visit.de/ontologies/vismo/literature -->
344
345    <owl:DatatypeProperty rdf:about="http://visit.de/ontologies/vismo/literature">
346      <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topDataProperty"/>
347      <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
348      <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
349      <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">This property
350      defines a literature entry that contains further information about the given vismo
entity.</rdfs:comment>
      <rdfs:isDefinedBy rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://visit.
de/ontologies/vismo/</rdfs:isDefinedBy>
      <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">literature</
351      rdfs:label>
352    </owl:DatatypeProperty>
353
354
355

```

```

356
357      <!-- http://visit.de/ontologies/vismo/outerDescription -->
358
359      <owl:DatatypeProperty rdf:about="http://visit.de/ontologies/vismo/outerDescription">
360          <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topDataProperty"/>
361          <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Architecture"/>
362          <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
363          <rdfs:comment>Property used to describe the exterior of a vismo:Architecture entity.</rdfs:comment>
364          <rdfs:label>outer description</rdfs:label>
365      </owl:DatatypeProperty>
366
367
368
369      <!-- http://visit.de/ontologies/vismo/pages -->
370
371      <owl:DatatypeProperty rdf:about="http://visit.de/ontologies/vismo/pages">
372          <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topDataProperty"/>
373          <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Reference"/>
374          <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
375          <rdfs:comment>Number of pages of a given vismo:Reference entity.</rdfs:comment>
376          <rdfs:label>pages</rdfs:label>
377      </owl:DatatypeProperty>
378
379
380
381      <!-- http://visit.de/ontologies/vismo/publisher -->
382
383      <owl:DatatypeProperty rdf:about="http://visit.de/ontologies/vismo/publisher">
384          <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topDataProperty"/>
385          <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Reference"/>
386          <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
387          <rdfs:comment>The publisher of the given vismo:Reference entity.</rdfs:comment>
388          <rdfs:label>publisher</rdfs:label>
389      </owl:DatatypeProperty>
390
391
392
393      <!-- http://visit.de/ontologies/vismo/series -->
394
395      <owl:DatatypeProperty rdf:about="http://visit.de/ontologies/vismo/series">
396          <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topDataProperty"/>
397          <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Reference"/>
398          <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
399          <rdfs:label>series</rdfs:label>
400      </owl:DatatypeProperty>
401
402
403
404      <!-- http://visit.de/ontologies/vismo/superordinateTitle -->
405
406      <owl:DatatypeProperty rdf:about="http://visit.de/ontologies/vismo/superordinateTitle">
407          <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topDataProperty"/>
408          <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Title"/>
409          <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
410          <rdfs:comment>Textual String to name the title of the superordinate reference collection that incorporates the associated vismo:Reference entity.</rdfs:comment>
411          <rdfs:label>superordinate title</rdfs:label>
412      </owl:DatatypeProperty>
413
414
415
416      <!-- http://visit.de/ontologies/vismo/technicalMetadata -->
417
418      <owl:DatatypeProperty rdf:about="http://visit.de/ontologies/vismo/technicalMetadata">
419          <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topDataProperty"/>
420          <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/DigitalRepresentation"/>
421          <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
422          <rdfs:comment>Refers from a vismo:DigitalRepresentation to a JSON formatted String that represents the technical metadata that is produced by the process that creates given digital representation.</rdfs:comment>
423          <rdfs:label>technical metadata</rdfs:label>
424      </owl:DatatypeProperty>
425
426
427
428      <!-- http://visit.de/ontologies/vismo/thumbnail -->
429
430      <owl:DatatypeProperty rdf:about="http://visit.de/ontologies/vismo/thumbnail">
431          <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topDataProperty"/>

```

```

432   <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
433   <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
434   <rdfs:comment>A thumbnail generated for the respective digital representation. In
435   general a base 64 encoding.</rdfs:comment>
436   <rdfs:label>thumbnail</rdfs:label>
437 </owl:DatatypeProperty>

438
439
440 <!-- http://visit.de/ontologies/vismo/volume -->
441
442 <owl:DatatypeProperty rdf:about="http://visit.de/ontologies/vismo/volume">
443   <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topDataProperty"/>
444   <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/Reference"/>
445   <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
446   <rdfs:comment>Number of volumes of a given publication series.</rdfs:comment>
447   <rdfs:label>volume</rdfs:label>
448 </owl:DatatypeProperty>

449
450
451
452 <!-- http://www.w3.org/2002/07/owl#topDataProperty -->
453
454 <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#topDataProperty">
455   <rdfs:domain rdf:resource="http://visit.de/ontologies/vismo/DigitalRepresentation"/>
456   <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
457 </rdf:Description>

458
459
460
461 <!--
462 //////////////////////////////////////////////////////////////////
463 // Classes
464 //
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
477   that can contribute a timely frame for associated ViSIT concepts. For example &quot;
478   World War II&quot;, &quot;The battle for town x&quot;, etc.</rdfs:comment>
479   <rdfs:label>Activity</rdfs:label>
480 </owl:Class>

481
482
483 <!-- http://visit.de/ontologies/vismo/Architecture -->
484
485 <owl:Class rdf:about="http://visit.de/ontologies/vismo/Architecture">
486   <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E53_Place"/>
487   <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E84_Information_Carrier
488   "/>
489   <rdfs:subClassOf rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
490   <rdfs:comment>Architecture in the ViSIT context describes every building or
491   architectural production that has been erected by mankind in some way.</rdfs:comment>
492   <rdfs:label>Architecture</rdfs:label>
493 </owl:Class>

494
495
496 <!-- http://visit.de/ontologies/vismo/BishopricAffiliation -->
497
498 <owl:Class rdf:about="http://visit.de/ontologies/vismo/BishopricAffiliation">
499   <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E55_Type"/>
500   <rdfs:comment>This class comprises headwords for bishopric affiliations for
501   vismo:Architecture resources.</rdfs:comment>
502   <rdfs:label>Bishopric Affiliation</rdfs:label>
503 </owl:Class>

504
505 <!-- http://visit.de/ontologies/vismo/Country -->

```

```

506
507      <owl:Class rdf:about="http://visit.de/ontologies/vismo/Country">
508          <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E53_Place"/>
509          <rdfs:subClassOf rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
510          <rdfs:label>Country</rdfs:label>
511      </owl:Class>
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
519          to give temporal associations with various entities.</rdfs:comment>
520          <rdfs:label>Dating</rdfs:label>
521      </owl:Class>
522
523
524      <!-- http://visit.de/ontologies/vismo/Description -->
525
526      <owl:Class rdf:about="http://visit.de/ontologies/vismo/Description">
527          <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E55_Type"/>
528          <rdfs:comment>Descriptions comprise characteristic types for objects in the domain of
529          museums. Therefore these are for example "painting", "oil painting";
530          &quot;chest&quot;; etc.</rdfs:comment>
531          <rdfs:label>Description</rdfs:label>
532      </owl:Class>
533
534
535      <!-- http://visit.de/ontologies/vismo/DigitalRepresentation -->
536
537      <owl:Class rdf:about="http://visit.de/ontologies/vismo/DigitalRepresentation">
538          <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
539          <rdfs:comment>A digital representation symbolises a multimedia representation of a
540          vismo:Resource that can be illustrated in some way. These incorporate pictures, videos,
541          audio files, and 3D models in particular.</rdfs:comment>
542          <rdfs:label>Digital Representation</rdfs:label>
543      </owl:Class>
544
545
546      <!-- http://visit.de/ontologies/vismo/Function -->
547
548      <owl:Class rdf:about="http://visit.de/ontologies/vismo/Function">
549          <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E55_Type"/>
550          <rdfs:comment>Functions relate to semantical and functional properties that are
551          inherited by vismo:Object as well as vismo:Architecture and their vismo:Structural
552          Evolution resources. Both an object as well as an architecture could exert "
553          military" functions.</rdfs:comment>
554          <rdfs:label>Function</rdfs:label>
555      </owl:Class>
556
557
558      <!-- http://visit.de/ontologies/vismo/GeographicalAffiliation -->
559
560      <owl:Class rdf:about="http://visit.de/ontologies/vismo/GeographicalAffiliation">
561          <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E55_Type"/>
562          <rdfs:comment>This class comprises headwords for geographical affiliations for
563          vismo:Architecture resources.</rdfs:comment>
564          <rdfs:label>Geographical Affiliation</rdfs:label>
565      </owl:Class>
566
567
568      <!-- http://visit.de/ontologies/vismo/Group -->
569
570      <owl:Class rdf:about="http://visit.de/ontologies/vismo/Group">
571          <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E74_Group"/>
572          <rdfs:subClassOf rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
573          <rdfs:comment>This more general class will comprise different groups of people that
574          are associated in the VisMo context. In the first instance these are WorkingGroups (
      Werkst\"aften) and joint practices (Soziet\"aten).</rdfs:comment>
          <rdfs:label>Group</rdfs:label>
      </owl:Class>

```

```

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
580              are associated in the VisMo context, such as Werkstatt and Soziet\"a t.</rdfs:comment>
581              <rdfs:label>Group Description</rdfs:label>
582      </owl:Class>
583
584
585
586      <!-- http://visit.de/ontologies/vismo/HistoricalChange -->
587
588      <owl:Class rdf:about="http://visit.de/ontologies/vismo/HistoricalChange">
589          <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E9_Move"/>
590          <rdfs:subClassOf rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
591          <rdfs:label>HistoricalChange</rdfs:label>
592      </owl:Class>
593
594
595
596      <!-- http://visit.de/ontologies/vismo/InscriptionType -->
597
598      <owl:Class rdf:about="http://visit.de/ontologies/vismo/InscriptionType">
599          <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E55_Type"/>
600          <rdfs:comment>This E55_Type describes the type of an Inscription, done on various
601              vismo:Object entities.</rdfs:comment>
602          <rdfs:label>Inscription Type</rdfs:label>
603      </owl:Class>
604
605
606
607      <!-- http://visit.de/ontologies/vismo/Institution -->
608
609      <owl:Class rdf:about="http://visit.de/ontologies/vismo/Institution">
610          <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E53_Place"/>
611          <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E74_Group"/>
612          <rdfs:subClassOf rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
613          <rdfs:comment>An institution in the ViSIT context is primarily used for museums,
614              which inherit both the properties of a E53_Place as well as a E74_Group. This is
615              necessary to make instances of this class be able to represent a spatial entity as well
616              as an entity that can for example hold vismo:Objects.</rdfs:comment>
617          <rdfs:label>Institution</rdfs:label>
618      </owl:Class>
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645

```

```

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            "/>
655        <rdfs:subClassOf rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
656        <rdfs:comment>Objects in the ViSiT context subsume all sorts of items that are
657        displayed in a museum.</rdfs:comment>
658        <rdfs:label>Object</rdfs:label>
659    </owl:Class>
660
661
662    <!-- http://visit.de/ontologies/vismo/OrderAffiliation -->
663
664    <owl:Class rdf:about="http://visit.de/ontologies/vismo/OrderAffiliation">
665        <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E55_Type"/>
666        <rdfs:comment>This class comprises headwords for order affiliations for
667        vismo:Architecture resources.</rdfs:comment>
668        <rdfs:label>Order Affiliation</rdfs:label>
669    </owl:Class>
670
671
672    <!-- http://visit.de/ontologies/vismo/Person -->
673
674    <owl:Class rdf:about="http://visit.de/ontologies/vismo/Person">
675        <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E21_Person"/>
676        <rdfs:subClassOf rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
677        <rdfs:subClassOf rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
678        <rdfs:label>Person</rdfs:label>
679    </owl:Class>
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
688        this class.</rdfs:comment>
689        <rdfs:label>Place</rdfs:label>
690    </owl:Class>
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.
698        for persons 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            "/>
708        <rdfs:comment>Used in the cultural use case of Visit as a reference to various
709        textual information objects that contained further and descriptive information about a
710        given Visit resource.</rdfs:comment>
711        <rdfs:label>Reference</rdfs:label>
712    </owl:Class>
713
714
715    <!-- http://visit.de/ontologies/vismo/ReferenceEntry -->
716
717    <owl:Class rdf:about="http://visit.de/ontologies/vismo/ReferenceEntry">
718        <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E84_Information_Carrier
719            "/>
```

```

717     <rdfs:comment>A Reference Entry contains further information about the reference of a
718     vismo:Resource in a given vismo:Reference entity, like the page numbers for example.</
719     rdfs:comment>
720         <rdfs:label>Reference Entry</rdfs:label>
721     </owl:Class>

722
723     <!-- http://visit.de/ontologies/vismo/ReferenceType -->
724
725     <owl:Class rdf:about="http://visit.de/ontologies/vismo/ReferenceType">
726         <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E55_Type"/>
727             <rdfs:comment>Summarises the various types that references in the cultural heritage
728             domain can have.</rdfs:comment>
729             <rdfs:label>Reference Type</rdfs:label>
730     </owl:Class>

731
732
733     <!-- http://visit.de/ontologies/vismo/Resource -->
734
735     <owl:Class rdf:about="http://visit.de/ontologies/vismo/Resource">
736         <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E1_CRM_Entity"/>
737         <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
738             <rdfs:comment>A vismo:Resource adds descriptive functionality to the resources used
739             in the ViSIT context, therefore adding the possibilities of adding comments,
740             descriptions, as well as literature information to the given resource.</rdfs:comment>
741             <rdfs:label>Resource</rdfs:label>
742     </owl:Class>

743
744     <!-- http://visit.de/ontologies/vismo/Room -->
745
746     <owl:Class rdf:about="http://visit.de/ontologies/vismo/Room">
747         <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E53_Place"/>
748         <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E84_Information_Carrier
749             "/>
750             <rdfs:subClassOf rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
751                 <rdfs:comment>A room in the classic sense. Can only be associated with its
752                 vismo:Architecture entity that contains it.</rdfs:comment>
753                 <rdfs:label>Room</rdfs:label>
754     </owl:Class>

755
756     <!-- http://visit.de/ontologies/vismo/SacralBuilding -->
757
758     <owl:Class rdf:about="http://visit.de/ontologies/vismo/SacralBuilding">
759         <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E55_Type"/>
760             <rdfs:comment>A further type characterisation for vismo:Architecture entities,
761             classifying them by a sacral type.</rdfs:comment>
762             <rdfs:label>Sacral Building</rdfs:label>
763     </owl:Class>

764
765
766     <!-- http://visit.de/ontologies/vismo/SecularBuilding -->
767
768     <owl:Class rdf:about="http://visit.de/ontologies/vismo/SecularBuilding">
769         <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E55_Type"/>
770             <rdfs:comment>A further type characterisation for vismo:Architecture entities,
771             classifying them by a secular type.</rdfs:comment>
772             <rdfs:label>Secular Building</rdfs:label>
773     </owl:Class>

774
775
776     <!-- http://visit.de/ontologies/vismo/StructuralEvolution -->
777
778     <owl:Class rdf:about="http://visit.de/ontologies/vismo/StructuralEvolution">
779         <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E11_Modification"/>
780         <rdfs:subClassOf rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
781             <rdfs:comment>A vismo:StructuralEvolution changes a vismo:Architecture entity in some
782             way. A change in its basic vismo:Function can thereby be established. For example a
783             castle that changes from its military function to a museum.</rdfs:comment>
784             <rdfs:label>StructuralEvolution</rdfs:label>
785     </owl:Class>

```

```

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
792          result of producing an vismo:Object that are associated with the cultural heritage
793          domain.</rdfs:comment>
794          <rdfs:label>Technique</rdfs:label>
795      </owl:Class>

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
802          of a 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
813          trade 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
823          sold on 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
833          associated with the vismo:Trade that is associated with the given trade route. These
834          cities can thereby be starting or end location, as well as an intermediate station.</
835          rdfs:comment>
836          <rdfs:label>Traderoute</rdfs:label>
837      </owl:Class>

838
839
840      <!-- http://visit.de/ontologies/vismo/WorkingGroup -->
841
842      <owl:Class rdf:about="http://visit.de/ontologies/vismo/WorkingGroup">
843          <rdfs:subClassOf rdf:resource="http://erlangen-crm.org/170309/E74_Group"/>
844          <rdfs:subClassOf rdf:resource="http://visit.de/ontologies/vismo/Resource"/>
845          <rdfs:comment>Frauke :)</rdfs:comment>
846          <rdfs:label>WorkingGroup</rdfs:label>
847      </owl:Class>
848
849  </rdf:RDF>

```

Listing 29: 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",

```

Adapter Name *	visittestrepo	Machine name: visomotes
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</p> <p>Is this Adapter writable?</p> <p><input checked="" type="checkbox"/> Preferred Local Store</p> <p>Is this Adapter the preferred local store?</p> <p>Read URL</p> <p><code>http://localhost:8081/rdf4j-server/repositories/visittestrepo</code></p> <p>Write URL</p> <p><code>http://localhost:8081/rdf4j-server/repositories/visittestrepo/statements</code></p> <p><input type="checkbox"/> Use graph independent rewriting rewrite queries, so that remote SPARQL stores with non-standard dataset handling do always answer right</p> <p>Default Graph URL *</p> <p><code>http://visitmot.de/data/</code></p> <p>Graph URL that is used to store triples in by default. May also be used as a base for new entity URLs.</p> <p>Ontology graphs</p>		

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 if system automatically detect the graphs.

Abbildung 56: Übersicht der Konfiguration eines WissKI Salz Adapters, Teil 1.

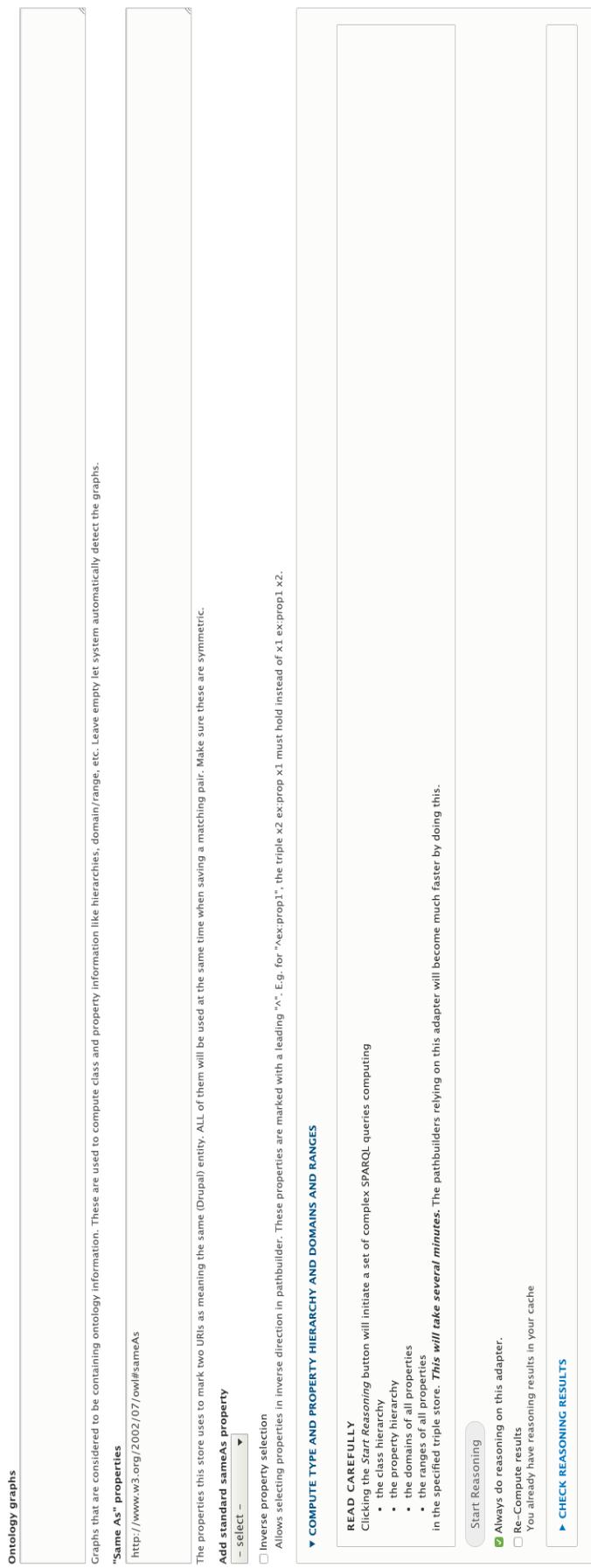


Abbildung 57: Übersicht der Konfiguration eines WissKI Salz Adapters, Teil 2.

Select the store for which you want to load an ontology.			
Currently loaded Ontology:			
Name	Iri	Version	Graph
http://visit.de/ontologies/vismo/	none	http://visit.de/ontologies/vismo/0.4.5/	https://a.uguu.se/Qb0jthNQ28qD.owl
http://erlangen-crm.org/170309/	none	none	http://erlangen-crm.org/170309/
http://xmins.com/foaf/0.1/	none	none	http://xmins.com/foaf/0.1/
Delete Ontology			
Short Name	URI		
ref	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://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://xmins.com/wot/0.1/		
frmf	https://xmins.com/frmf/n1/		

Abbildung 58: Detailansicht einer definierten Ontology im WissKI System am Beispiel VisMo für das ViSIT Projekt.

TITLE	PATH	ENABLED	FIELD TYPE	CARDINALITY	WEIGHT	OPERATIONS
+ Object	Group [http://visit.de/ontologies/vismo/Object]	✓	Text (plain)	Unlimited	Edits	▼
+ Object_identifiedBy_Title	http://visit.de/ontologies/vismo/Object -> ecrm:P1_is_idenitified_by -> ecrm:E35_Title	✓	Text (plain)	Unlimited	Edits	▼
+ Object_has_Description	http://visit.de/ontologies/vismo/Object -> ecrm:P2_has_type -> http://visit.de/ontologies/vismo/Description	✓	Text (plain)	Unlimited	Edits	▼
+ Object_exemplifies_Function	http://visit.de/ontologies/vismo/Object -> ecrm:P137_exemplifies -> ecrm:P160_has_temporal_projection -> http://visit.de/ontologies/vismo/Function	✓	Text (plain)	Unlimited	Edits	▼
+ Object_Dating	Group [http://visit.de/ontologies/vismo/Object -> ecrm:P160_has_temporal_projection -> http://visit.de/ontologies/vismo/Dating]	✓	Text (plain)	Unlimited	Edits	▼
+ Object_Dating_start	http://visit.de/ontologies/vismo/Object -> ecrm:P160_has_temporal_projection -> http://visit.de/ontologies/vismo/Dating	✓	Text (plain)	Unlimited	Edits	▼
+ Object_Dating_end	http://visit.de/ontologies/vismo/Object -> ecrm:P160_has_temporal_projection -> http://visit.de/ontologies/vismo/Dating	✓	Text (plain)	Unlimited	Edits	▼
+ Object_Dating_sometime	http://visit.de/ontologies/vismo/Object -> ecrm:P160_has_temporal_projection -> http://visit.de/ontologies/vismo/Dating	✓	Text (plain)	Unlimited	Edits	▼
+ Object_Dating_century	http://visit.de/ontologies/vismo/Object -> ecrm:P160_has_temporal_projection -> http://visit.de/ontologies/vismo/Dating	✓	List (text)	Unlimited	Edits	▼
+ Object_composedOf_Object	http://visit.de/ontologies/vismo/Object -> ecrm:P46_is_composed_of -> http://visit.de/ontologies/vismo/Object	✓	Entity reference	Unlimited	Edits	▼
+ Object_PartOf_Object	http://visit.de/ontologies/vismo/Object -> ecrm:P46i_forms_part_of -> http://visit.de/ontologies/vismo/Object	✓	Entity reference	Unlimited	Edits	▼
+ Object_description	http://visit.de/ontologies/vismo/Object	✓	Text (plain, long)	Unlimited	Edits	▼
+ Object_producedBy_Production	Group [http://visit.de/ontologies/vismo/Object -> ecrm:P108i_was_produced_by -> ecrm:E12_Production]	✓	Text (plain)	Unlimited	Edits	▼
+ Object_employs_Material	http://visit.de/ontologies/vismo/Object -> ecrm:P108i_was_produced_by -> ecrm:E12_Production -> ecrm:P126_employed -> ecrm:E57_Material	✓	Text (plain)	Unlimited	Edits	▼
+ Production_useOf_technique	http://visit.de/ontologies/vismo/Object -> ecrm:P108i_was_produced_by -> ecrm:E12_Production -> ecrm:P32_used_general_technique -> http://visit.de/ontologies/vismo/technique	✓	Text (plain)	Unlimited	Edits	▼

Abbildung 59: Übersicht der ersten Pfade des für das ViSIT Projekt definierten Pathbuilders.

 Visit Database Home Find Navigate Create My account Log out

Abbildung 60: Ausgangs-Interface zum Erzeugen einer Hauptentität im WissKI System.

Home » Create WissKI Entity

TITEL	<input type="text"/> <input type="button" value="Show row weights"/>
OBJEKTEZEICHNUNG	<input type="text"/> <input type="button" value="Show row weights"/>
FUNKTION	<input type="text"/> <input type="button" value="Show row weights"/>
OBJEKT BESTEHT AUS: (TEILOBJECTEN)	<input type="text"/> <input type="button" value="Show row weights"/>
TEILOBJECT VON: (OBJEKT)	<input type="text"/> <input type="button" value="Show row weights"/>

Geben Sie hier den Titel des Objekts ein, das Sie anzeigen wollen, z.B. "Einzug der Kriemhild".

Fügen Sie hier möglichst präzise und schlagwortartig ein, um was für einen Objekttyp es sich handelt, z.B. "Gemälde", "Turrahmung" oder "Henkelkrug". Achtung! Für den Titel eines Objekts (z.B. "Mona Lisa") steht das Feld "Titel" zur Verfügung.

Hier können Sie beschreiben, was das Objekt ursprünglich für eine Funktion hatte und in welchen Zusammenhangen es genutzt wurde, z.B.: Spezifikation einer Bügeleisflasche als "Bierflasche" oder eines Henkelkrugs als "Essigkrug".

ALLGEMEINE OBJEKTDATIERUNG

 In dieser Gruppe können Sie Angaben zur allgemeinen Objektdatierung machen.

Abbildung 61: 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, a breadcrumb trail indicates the current location: 'Navigate / Object / Partisane (587)'. The main content area is titled 'Partisane (587)' and contains a table of properties. The properties listed include:

- Objektbezeichnung**: Partisane
- Prunkpartisane**
- Inscription**
- Text**: I.P.D.G.E.P.S.R.-I.P.E.G.D.L.
- Anbringung**: Über dem Passauer Wolf
- Datering**: 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

Below the table, there are tabs for 'View', 'Edit', 'Delete', 'Triples', and 'Devel'.

Abbildung 62: Beispiel Ausgabe-Interface einer Partisane des ViSIT WissKI Systems.

```

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   "object_dating": {
66     "type": "http://visit.de/ontologies/vismo/Dating",
67     "object_dating_start": "string",
68     "object_dating_end": "string",
69     "object_dating_sometime": "string",
70     "object_dating_century": "list_string"
71   },
72   "object_refentry": {
73     "type": "http://visit.de/ontologies/vismo/ReferenceEntry",
74     "object_refentry_pages": "string",
75     "object_refentry_in_reference": "entity_reference (http://visit.de/ontologies/vismo/
76     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_infl_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_object": "entity_reference (http://visit.de/ontologies/vismo/Object)",
108  "person_lostcustodyof_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",
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_icongraphy": "string",
171     "architecture_innerdescription": "string_long",
172     "architecture_outerdescription": "string_long",
173     "architecture_depictedby_object": "entity_reference (http://visit.de/ontologies/vismo/
174       Object)",
175     "architecture_buildinghistory": "string_long",
176     "arch_currentlyholds_object": "entity_reference (http://visit.de/ontologies/vismo/Object)
177       ",
178     "architecture_exemplify_function": "string",
179     "architecture_location_place": "entity_reference (http://visit.de/ontologies/vismo/Place)
180       ",
181     "arch_currentowner_person": "entity_reference (http://visit.de/ontologies/vismo/Person)",
182     "arch_currentowner_group": "entity_reference (http://visit.de/ontologies/vismo/Group)",
183     "arch_currentowner_institution": "entity_reference (http://visit.de/ontologies/vismo/
184       Institution)",
185     "architecture_contains_arch": "entity_reference (http://visit.de/ontologies/vismo/
186       Architecture)",
187     "architecture_fallswithin_arch": "entity_reference (http://visit.de/ontologies/vismo/
188       Architecture)",
189     "architecture_tookpartin_activity": "entity_reference (http://visit.de/ontologies/vismo/
190       Activity)",
191     "architecture_helpfullinks": "string",
192     "architecture_thumbnail": "image",
193     "arch_producedby_production": {
194       "type": "http://erlangen-crm.org/170309/E12_Production",
195       "production_motivatedby_person": "entity_reference (http://visit.de/ontologies/vismo/
196         Person)",
197       "production_carriedoutby_person": "entity_reference (http://visit.de/ontologies/vismo/
198         Person)",
199       "production_inflby_person": "entity_reference (http://visit.de/ontologies/vismo/Person)
200       ",
201       "arch_prod_motivatedby_group": "entity_reference (http://visit.de/ontologies/vismo/
202         Group)",
203       "arch_prod_carriedoutby_group": "entity_reference (http://visit.de/ontologies/vismo/
204         Group)",
205       "arch_prod_inflby_group": "entity_reference (http://visit.de/ontologies/vismo/Group)",
206       "arch_production_dating": {
207         "arch_prod_dating_start": "string",
208         "arch_prod_dating_end": "string",
209         "arch_production_sometime": "string",
210         "arch_prod_dating_century": "list_string"
211       }
212     },
213     "arch_modifiedby_structevolution": {
214       "type": "http://visit.de/ontologies/vismo/StructuralEvolution",
215       "structuralevolution_idby_title": "string",
216       "structuralevolution_description": "string_long",
217       "structuralevolution_comment": "string_long",
218       "structevol_exemplifies_function": "string",
219       "structevol_motivatedby_person": "entity_reference (http://visit.de/ontologies/vismo/
220         Person)",
221       "structevol_carriedoutby_person": "entity_reference (http://visit.de/ontologies/vismo/
222         Person)",
223       "structevol_influencedby_person": "entity_reference (http://visit.de/ontologies/vismo/
224         Person)",
225       "structevol_motivby_group": "entity_reference (http://visit.de/ontologies/vismo/Group),
226
227       "structevol_carriedoutby_group": "entity_reference (http://visit.de/ontologies/vismo/
228         Group)",
229       "structevol_inflby_group": "entity_reference (http://visit.de/ontologies/vismo/Group)",
230       "arch_structevol_dating": {
231         "arch_structevol_dating_start": "string",
232         "arch_structevol_dating_end": "string",
233         "arch_evol_dat_sometime": "string",
234         "arch_structevol_dating_century": "list_string"
235       }
236     }
237   }
238 }
```

```

220     "arch_refentry": {
221         "type": "http://visit.de/ontologies/vismo/ReferenceEntry",
222         "arch_refentry_pages": "string",
223         "arch_refentry_in_reference": "entity_reference (http://visit.de/ontologies/vismo/
224             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/
239         Architecture)",
240     "place_witnessed_activity": "entity_reference (http://visit.de/ontologies/vismo/Activity)
241         ",
242     "place_wasbirthplaceof_person": "entity_reference (http://visit.de/ontologies/vismo/
243         Person)",
244     "place_wasdeathplaceof_person": "entity_reference (http://visit.de/ontologies/vismo/
245         Person)",
246     "place_isdepictedby_object": "entity_reference (http://visit.de/ontologies/vismo/Object)",

247     "place_helpfullinks": "string",
248     "place_thumbnail": "image",
249     "place_refentry": {
250         "type": "http://visit.de/ontologies/vismo/ReferenceEntry",
251         "place_refentry_pages": "string",
252         "place_refentry_in_reference": "entity_reference (http://visit.de/ontologies/vismo/
253             Reference)"
254     },
255     "place_digitalrepresentation": {
256         "type": "http://visit.de/ontologies/vismo/DigitalRepresentation",
257         "place_dr_techmetadata": "string"
258     }
259 },
260 "Institution": {
261     "type": "http://visit.de/ontologies/vismo/Institution",
262     "institution_idby_appel": "string",
263     "institution_ownerof_arch": "entity_reference (http://visit.de/ontologies/vismo/
264         Architecture)",
265     "institution_fallswithin_place": "entity_reference (http://visit.de/ontologies/vismo/
266         Place)",
267     "institution_address": "string",
268     "institution_owns_catalog": "entity_reference (http://visit.de/ontologies/vismo/Reference)
269         ",
270     "institution_loc_catalog": "entity_reference (http://visit.de/ontologies/vismo/Reference)
271         ",
272     "institution_helpfullinks": "string"
273 },
274 "Group": {
275     "type": "http://visit.de/ontologies/vismo/Group",
276     "group_idby_actorappel": "string",
277     "group_keyword": "string",
278     "group_iconography": "string",
279     "group_produced_object": "entity_reference (http://visit.de/ontologies/vismo/Object)",
280     "group_ownerof_architecture": "entity_reference (http://visit.de/ontologies/vismo/
281         Architecture)",
282     "group_motiv_arch_production": "entity_reference (http://visit.de/ontologies/vismo/
283         Architecture)",
284     "group_carriedout_arch_production": "entity_reference (http://visit.de/ontologies/vismo/
285         Architecture)",
286     "group_infl_arch_production": "entity_reference (http://visit.de/ontologies/vismo/
287         Architecture)",
288     "group_motiv_structevol": "entity_reference (http://visit.de/ontologies/vismo/
289         Architecture)",
290     "group_carriedout_structevol": "entity_reference (http://visit.de/ontologies/vismo/
291         Architecture)",
292     "group_infl_structevol": "entity_reference (http://visit.de/ontologies/vismo/
293         Architecture)",
294     "group_receivedcustodyof_object": "entity_reference (http://visit.de/ontologies/vismo/
295         Object)",
296     "group_lostcustodyof_object": "entity_reference (http://visit.de/ontologies/vismo/Object)
297         ",
298     "group_refentry": {
299

```

```

280     "type": "http://visit.de/ontologies/vismo/ReferenceEntry",
281     "group_refentry_pages": "string",
282     "group_refentry_in_reference": "entity_reference (http://visit.de/ontologies/vismo/
283     Reference)"
284   },
285   "Reference": {
286     "type": "http://visit.de/ontologies/vismo/Reference",
287     "reference_keyword": "string",
288     "reference_has_type": "string",
289     "reference_publisher": "string",
290     "reference_series": "string",
291     "reference_volume": "integer",
292     "reference_pages": "integer",
293     "reference_catalog_owner": "entity_reference (http://visit.de/ontologies/vismo/
294     Institution)",
295     "reference_catalog_location": "entity_reference (http://visit.de/ontologies/vismo/
296     Institution)",
297     "reference_title": {
298       "type": "http://visit.de/ontologies/vismo/Title",
299       "reference_title_title": "string",
300       "reference_title_superordinate": "string"
301     },
302     "reference_producedby_production": {
303       "type": "http://erlangen-crm.org/170309/E12_Production",
304       "production_authormame": "string",
305       "production_year": "integer",
306       "ref_production_placeofpub": "string"
307     },
308     "reference_entry": {
309       "type": "http://visit.de/ontologies/vismo/ReferenceEntry",
310       "reference_entry_pages": "string",
311       "reference_entry_about_activity": "entity_reference (http://visit.de/ontologies/vismo/
312       Activity)",
313       "reference_entry_about_arch": "entity_reference (http://visit.de/ontologies/vismo/
314       Architecture)",
315       "reference_entry_about_object": "entity_reference (http://visit.de/ontologies/vismo/
316       Object)",
317       "reference_entry_about_place": "entity_reference (http://visit.de/ontologies/vismo/
318       Place)",
319       "reference_entry_about_group": "entity_reference (http://visit.de/ontologies/vismo/
320       Group)",
321       "reference_entry_about_person": "entity_reference (http://visit.de/ontologies/vismo/
322       Person)"
323     },
324     "reference_catalog_dating": {
325       "type": "http://visit.de/ontologies/vismo/Dating",
326       "catalog_exhibition_start": "datetime",
327       "catalog_exhibition_end": "datetime"
328     }
329   },
330   "Activity": {
331     "type": "http://visit.de/ontologies/vismo/Activity",
332     "activity_idby_title": "string",
333     "activity_description": "string_long",
334     "activity_comment": "string_long",
335     "activity_keyword": "string",
336     "activity_iconography": "string",
337     "activity_hadparticipant_person": "entity_reference (http://visit.de/ontologies/vismo/
338     Person)",
339     "activity_used_architecture": "entity_reference (http://visit.de/ontologies/vismo/
340     Architecture)",
341     "activity_used_object": "entity_reference (http://visit.de/ontologies/vismo/Object)",
342     "activity_tookplaceat_place": "entity_reference (http://visit.de/ontologies/vismo/Place)",

343     "activity_isdepictedby_object": "entity_reference (http://visit.de/ontologies/vismo/
344     Object)",
345     "activity_helpfullinks": "string",
346     "activity_thumbnail": "image",
347     "activity_dating": {
348       "type": "http://visit.de/ontologies/vismo/Dating",
349       "activity_dating_exact": "datetime",
350       "activity_dating_end": "string",
351       "activity_dating_start": "string",
352       "activity_dating_sometime": "string"
353     },
354     "activity_refentry": {
355       "type": "http://visit.de/ontologies/vismo/ReferenceEntry",
356       "activity_refentry_pages": "string",
357     }
358   }
359 
```

```
346     "activity_refentry_in_reference": "entity_reference (http://visit.de/ontologies/vismo/  
347     Reference)"  
348 },  
349 "activity_digitalrepresentation": {  
350     "type": "http://visit.de/ontologies/vismo/DigitalRepresentation",  
351     "activity_dr_techmetadata": "string"  
352 }  
353 }
```

Listing 30: 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/>.