



# Anforderungen für ViSIT-Applikationen und technische Dokumentation

(Stand 04.04.2019)



# Inhaltsverzeichnis

## [Inhaltsverzeichnis](#)

### [Infrastruktur](#)

#### [Netzwerk](#)

[Tabelle 1: Verbindungen](#)

[Tabelle 2: Anwendungen / Verbindungen](#)

#### [Hardwareanforderungen](#)

[Lokaler App Server \(LAS\)](#)

[Hardwareanforderungen für die Kompressions - Komponente](#)

[Client-Anwendungen](#)

#### [Softwareanforderungen](#)

[LAS](#)

[Client-Anwendungen](#)

[Lizenzen](#)

### [Tablet-Station](#)

[Hardwarebeschreibung:](#)

[Softwarebeschreibung](#)

[Mehrsprachigkeit](#)

[Modul Karte](#)

[Modul Glossar](#)

[Modul Galerie](#)

### [Fernrohr \(Stand 24.10.2018\)](#)

[AR-Applikation mit POIs](#)

[Hardwarebeschreibung:](#)

[Softwarebeschreibung:](#)

[Anforderungen:](#)

[CMS:](#)

[Beispiele:](#)

[Möglichkeit des Offline-Betriebs](#)

### [Dateimanagement](#)

[Dateiformate](#)

[Dateiupload](#)

[Dateibrowser](#)

[P2P Partner Verwaltung](#)

### [Metadatenbank, Depot](#)

[Softwarebeschreibung](#)

[Hardwarebeschreibung](#)

[WissKI \(Warenkorb\)](#)

[Inter-Iframe - Kommunikation](#)

[REST API / Technischer Output der Metadatenbank](#)

[API für allgemeine Objekte](#)

[API für die technischen Metadaten](#)

[Json Formate](#)

[Json "MediaTripleData"](#)

[Json "AlleMediaTripleData"](#)

[Was die Metadatenbank/WissKI nicht kann](#)

[Kompressions-Komponente](#)

[Hardwarevoraussetzungen](#)

[Softwarevoraussetzungen](#)

[3D-Kompressions-Algorithmen](#)

[Administration & Konfiguration](#)

[Abgrenzungskriterien](#)

# Infrastruktur

## Netzwerk

Jegliche ViSIT-Applikationen werden als Server-Client-Applikation betrieben, oder besitzen zumindest grundlegend diese Architektur. Jeder ViSIT-Partner, der eine Medieninstallation mit ViSIT-Applikationen errichten will, benötigt ein hausinternes Netzwerk (Intranet) und einen damit verbundenen Server (lokaler App-Server. Kurz **LAS**). Die ViSIT-Applikationen (Clients) sind über dieses Netzwerk mit dem LAS verbunden. Die einzelnen Clients benötigen nicht zwingend Internetzugang. Nur der LAS, auf dem das ViSIT-System installiert ist, benötigt Zugang zum globalen ViSIT-Netzwerk via Internet (siehe Tabelle 1). Dieses ViSIT-System ist eine Ansammlung von mehreren kleineren Anwendungen (Apps), die parallel auf diesen LAS laufen (Tabelle 2). Jede dieser kleineren Anwendungen ist als Serversoftware für eine spezifische Clientsoftware zuständig. Da im Zuge des ViSIT-Projektes verschiedene mediale Anwendungen entstehen, wurde die Serversoftware modular konzipiert.

Tabelle 1: Verbindungen

Verbindung:	zum LAS	zum Client	zum Kuratoren-PC	zum Internet
<b>vom LAS</b>	localhost	je nach Anwendung, siehe Tabelle 2	keine Verb. notwendig	Notwendig, Siehe Tabelle 2
<b>vom Client</b>	je nach Anwendung, siehe Tabelle 2	localhost	keine Verb. notwendig	keine Verb. notwendig
<b>vom Kuratoren-PC</b>	je nach Anwendung, siehe Tabelle 2	keine Verb. notwendig	localhost	Notwendig (verbindung zu WissKi)

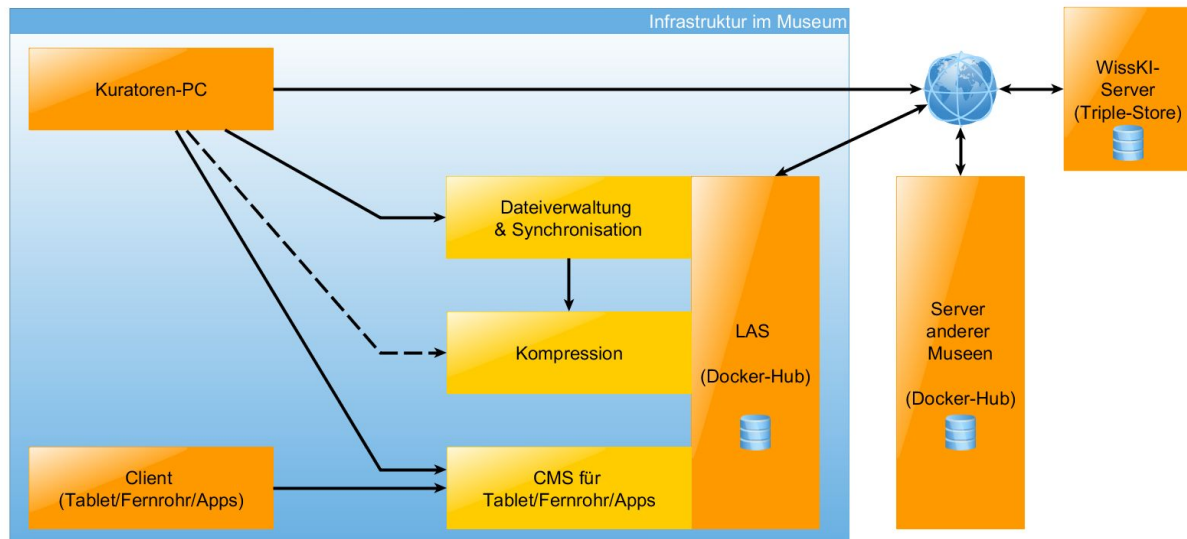
Tabelle 2: Anwendungen / Verbindungen

Hier werden die Verbindungen beschrieben, die zum und vom LAS aufgebaut werden müssen. Auf dem LAS werden mehrere Anwendungen wie beispielsweise die Dateiverwaltung oder die Tablet-Station installiert, die verschiedene Verbindungen benötigen:

ViSIT-App	vom LAS zum...		
	Client	Kuratoren PC	Internet
ViSIT-Dateiverwaltung	keine	Webanwendung Port 80, 443 TCP via LAN	<a href="#">ViSIT P2P Netzwerk 22000/TCP 21027/UDP</a> , Optimal wäre ein Zugang via SSH 22 (für docker management)
ViSIT-Tablet-Station	Webanwendung Port 80, 443 TCP via WLAN	keine	keine
ViSIT-Fernrohr	Webanwendung Port 80, 443 TCP via LAN / WLAN	keine	keine
Weitere Apps folgen			

Um den LAS einrichten zu können wird ein Zugang dazu benötigt. Dieser kann physisch sein oder via SSH (vom Intranet oder Internet aus erreichbar) bereitgestellt werden. Weiter sollte die Internetanbindung zum LAS so gewählt sein, dass die Synchronisation in einer vertretbaren Zeit stattfinden kann. Eine Anbindung über DSL sollte hier ausreichend sein.

Schaubild für alle Komponenten und deren Zugriffe untereinander:



## Hardwareanforderungen

### Lokaler App Server (LAS)

Der LAS für ViSIT hat je nach installierten Applikationen unterschiedliche Hardwareanforderungen. Die Mindestanforderungen für die Verwendung von ViSIT-Applikationen ist die Installation der Dateiverwaltung. Sie bildet die Grundlage für jegliche weiteren Installationen. Somit leiten sich die Mindestanforderungen an die Hardware direkt von den Anforderungen dieser Applikation ab. In dieser Anwendung können Daten unter den ViSIT-Partnern synchronisiert werden. Das Problem hierbei ist, dass enorme Datenmengen anfallen können und somit die Festplattenkapazität sehr groß ausgelegt werden muss. Da der Benutzer selbst entscheiden kann, welche Daten synchronisiert werden, wurde hier eine Festplattengröße von 250GB gewählt. Möchte ein ViSIT-Partner viele Dateien verwalten / synchronisieren, so muss dieser die Festplattengröße eigenständig dementsprechend dimensionieren. Ausfallsicherer Speicher mit RAID 1 oder 5 wäre zu empfehlen. Da bis dato (Stand März 2018) jegliche ViSIT-Apps als Webapps implementiert werden, reicht hierfür ein kleiner Webserver mit 8GB RAM und einem 2-Kern-Prozessor vollkommen aus.

Eine weitere ViSIT-Applikation dient der Kompression von 3D-Modellen. Diese Kompression wird im Hintergrund ausgeführt und kann sehr rechenintensiv sein. Sollen große Modelle komprimiert werden, sollten Arbeitsspeicher und Prozessor deutlich leistungsfähiger ausgelegt werden. Eine Quantifizierung der Anforderungen ist hier schwer, da der Rechenaufwand direkt von der Größe der Modelle abhängig ist. Damit die Modelle in die Kompressions-Komponente geladen werden können, muss der Arbeitsspeicher ausreichend dimensioniert sein. 32GB RAM sollten für übliche Modellgrößen (bis ca. 8 Mio. Dreiecke) ausreichend sein, sehr große Modelle können jedoch einen noch größeren Arbeitsspeicher notwendig machen. Die Rechenleistung der CPU entscheidet primär über die Dauer der

ausgeführten Kompressionsprozesse und muss dementsprechend ausgelegt werden. Empfohlen wird ein leistungsfähiger aktueller Prozessor mit mindestens vier Kernen.

Zusammenfassung Mindestanforderungen Hardware:

- Prozessor: 2 Kerne
- RAM: 8GB
- Festplatten: 250GB

Empfohlene Hardwareanforderungen:

- Prozessor: 4 Kerne
- RAM: 16 GB
- Festplatten: >500 GB

## Hardwareanforderungen für die Kompressions - Komponente

Minimal verfügbarer Arbeitsspeicher	Maximale Verarbeitungsgröße des 3D-Modelles
8GB	150MB
16GB	300MB
32GB +	700MB

## Client-Anwendungen

Anwendung	Hardware
Tablet Station	iPad pro 12.9
Fernrohr	?

## Softwareanforderungen

### LAS

Wie oben erläutert laufen unterschiedliche Anwendungen parallel auf dem LAS. Als zugrundeliegende Plattform wurde hier Docker gewählt. Jede ViSIT-App besteht somit aus einer Client-Software, die z. B. auf einem Tablet installiert ist und einer Server-App, die in

einem Docker-Container ausgeführt wird. Diese Container sind in sich geschlossene Einheiten und benötigen keine weitere zusätzliche Software. Somit ist Docker die einzige Software, die auf dem LAS benötigt wird. Docker verwendet einen virtuellen Switch, der den Containern erlaubt, mit einem Netzwerk zu kommunizieren. Wie in Tabelle 1 beschrieben muss die Konnektivität der Docker-Container mit den spezifizierten Endpunkten gewährleistet sein. Für die Entwicklung wurde Docker Community Edition in der Version 18.03 verwendet. Zu beachten ist, dass ein Laufwerk für Docker freigegeben werden muss, damit die Daten darauf gespeichert werden können. Notiz: Falls Windows verwendet wird, muss Windows Pro oder Windows Enterprise verwendet werden, da Docker auf Windows Hyper-V benötigt.

## Client-Anwendungen

Anwendung	Software
Tablet Station	Kiosk-App "SureFox"
Fernrohr	Browser mit Kiosk funktion, oder Browser im Vollbild

## Lizenzen

Applikation	Komponente	Lizenz
Linux / Ubuntu	OS	GNU GPL v2 - Open Source
Docker	Middleware	Diverse - Open Source <a href="https://www.docker.com/components-licenses">https://www.docker.com/components-licenses</a>
Docker AppContainer		Apache 2.0
Apache2	Webserver	Apache 2.0 - Open Source
PHP	HTML-Preprozessor	PHP License v3.01 - Open Source
MariaDB	Datenbankserver	GNU GPL v2 - Open Source
Typo3 CMS	Applikation	GNU GPL v2 - Open Source
Typo3 Fluid	Applikation	GNU LGPL v3 - Open Source
ThreeJS	Applikation	MIT License
JQuery	Applikation	MIT License / GNU GPL v2



JQuery TouchSwipe	Applikation	MIT License / GNU GPL v2
Bootstrap	Applikation	MIT License
Isotope JS	Applikation	GPLv3
FontAwesome	Applikation	Icons — CC BY 4.0 License Fonts — SIL OFL 1.1 License
Popper JS	Applikation	MIT License
Tether JS	Applikation	MIT License

## Tablet-Station

### Hardwarebeschreibung:

Als Endpunkt für diese Anwendung werden Tablets verwendet. Prinzipiell kann jeder Client mit einem Webbrowser verwendet werden. Jedoch wurde die Web-App explizit für das iPad Pro (12.9 Zoll, iOS version 11.4, Jahr 2017) angepasst.

### Softwarebeschreibung

Als Architektur wird eine Client/Server-Konstellation verwendet. Die serverseitige Software wird auf dem LAS (siehe Tabelle 1) betrieben. Die Kommunikation zu den Tablets erfolgt via TCP/IP (über Ethernet). Die Applikation wird in einem Docker-Container geliefert (<https://github.com/ViSIT-Dev/maincontainer>). In diesem Container ist ein Webserver, eine Datenbank und eine Installation von Typo3 enthalten. Damit der Webserver von den Tablets erreicht werden kann, muss der interne Port vom Container weitergeleitet werden. Das Typo3 muss nach der Installation noch konfiguriert (Sprache, optionales Domain Routing oder Seitennamen) werden. Im Typo3 ist die Extension "visit\_tablets" installiert. Diese Extension enthält mehrere Module, die jeweils für eine Sub-Applikation zuständig sind. Ein Wechsel zwischen diesen Modulen ist nicht vorgesehen. Auf dem Tablet wird in der SureFox Kiosk-App nur die Seite zugelassen, welche auf der gewünschten Sub-Applikation installiert ist.

### Mehrsprachigkeit

Die drei Sub-Applikationen werden so gestaltet, dass neben der Originalsprache (Deutsch) auch das Einpflegen und Ausgeben von Inhalten in der englischen Sprache ermöglicht wird. Es soll möglich sein, dass der Benutzer zwischen diesen Sprachen wählen kann.

## Modul Karte

In diesem Modul können POIs mit einer Eingabemaske, erstellt und editiert werden. Das Frontend wird gemäß Vorgabe:

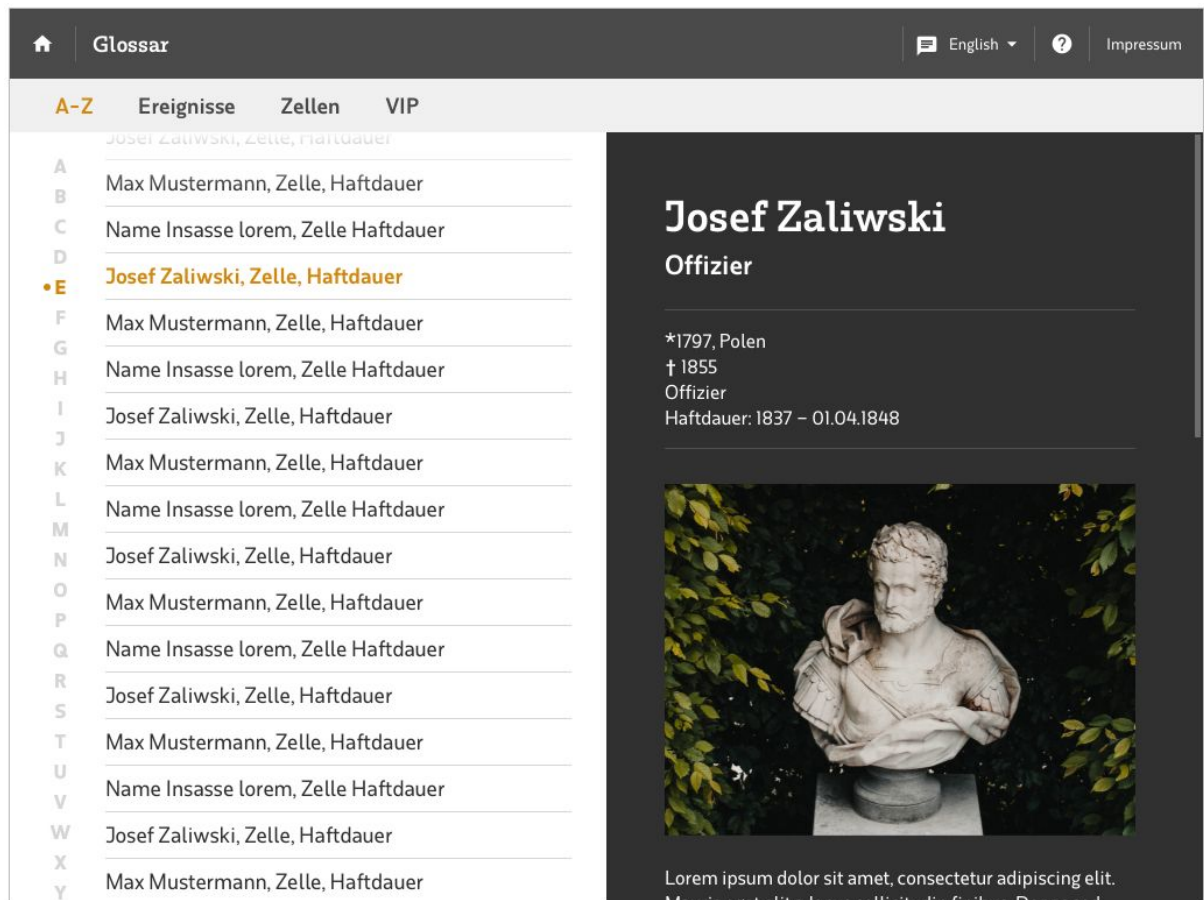


<https://app.zeplin.io/project/5b1ab36985de6d5f5f975396/screen/5b1ab3c016e7134441612020> (Stand 1.10.2018) umgesetzt

Auf dieser Karte werden die erstellten POIs angezeigt. Via klick öffnet sich der seitliche Info Bereich. Die "boundaries" sind fix definiert und können nicht beliebig erweitert werden, da das Kartenmaterial mit der Typo3 Extension ausgeliefert wird (offline Verfügbarkeit). Der Kontenbereich besteht aus einer Überschrift, Unterüberschrift, Medienobjekt (Bild, Video, 3D Objekt)

## Modul Glossar

Das Glossar beinhaltet eine Liste von Häftlingen. Die Applikation wird wie im Bild umgesetzt



Die Webapplikation ist in zwei Bereiche aufgeteilt. Rechts werden die Insassen aufgelistet (alphabetische Sortierung nach Namen bzw wenn vorhanden Nachnamen). Klickt der Benutzer auf einen der Einträge, so wird dieser auf der linken Seite angezeigt. Die Details setzen sich aus folgenden Feldern zusammen:

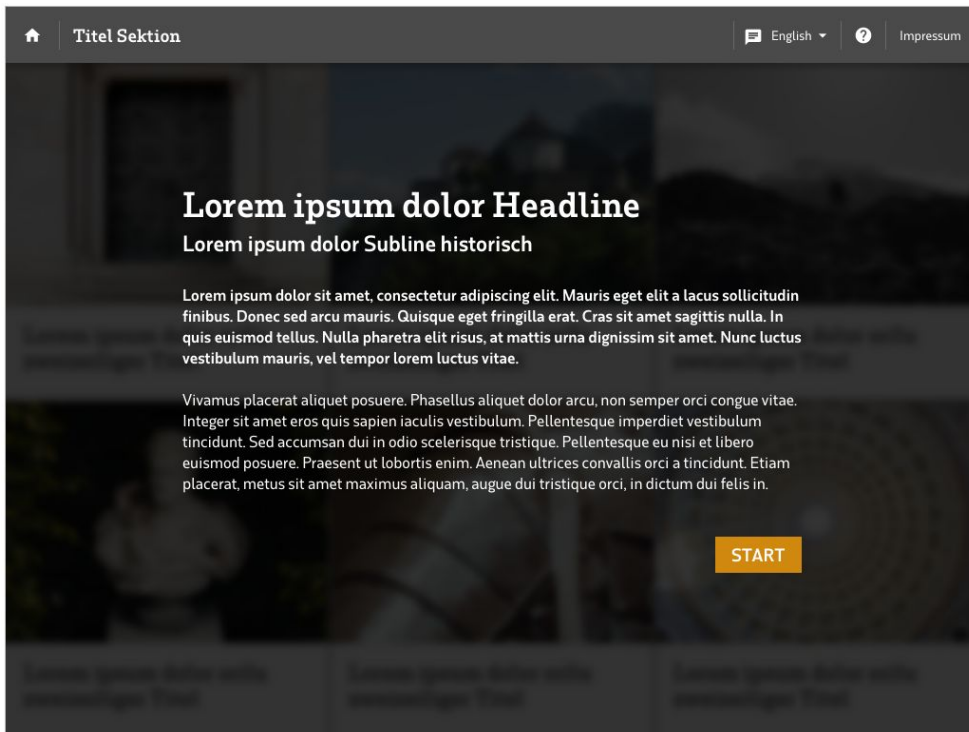
- Vorname
- Nachname
- Geburtsdatum
- Geburtsort
- Nationalität
- Sterbedatum
- Beruf
- Haftbeginn
- Haftende
- Zelle
- Untertitel
- Teasertext
- Fließtext
- Titelbild
- VIP
- Ereignis ID

Die Enumeration auf der linken Seite kann nach Name, Ereignisse, Zellen oder VIP sortiert werden.

## Modul Galerie

Die Galerie ist eine Verkettung von mehreren Unterseiten. Diese Unterseiten können folgende Layouts Besitzen:





Startbildschirm:










Nach einer bestimmten Zeit, kehrt die Applikation automatisch zum Menü zurück und zeigt den Startbildschirm an.



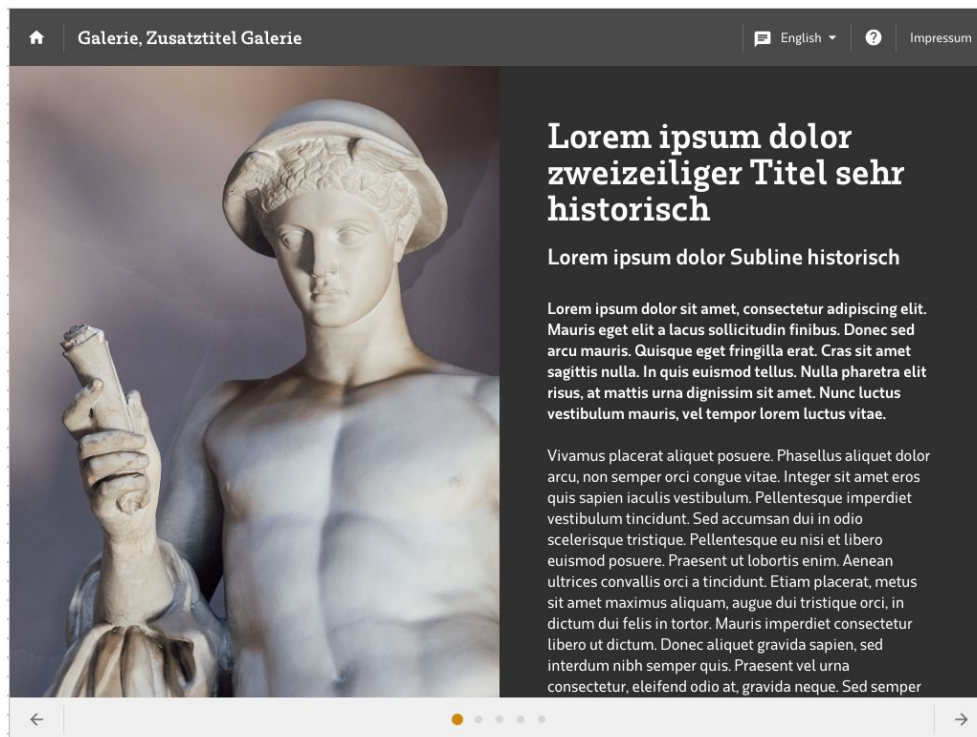
Menü 1:

<div><div> Titel Sektion</div><div><div>English</div><div>?</div><div>Impressum</div></div></div>		
		
<div> Lorem ipsum dolor orilu zweizeiliger Titel</div>	<div> Lorem ipsum dolor orilu zweizeiliger Titel</div>	<div> Lorem ipsum dolor orilu zweizeiliger Titel</div>

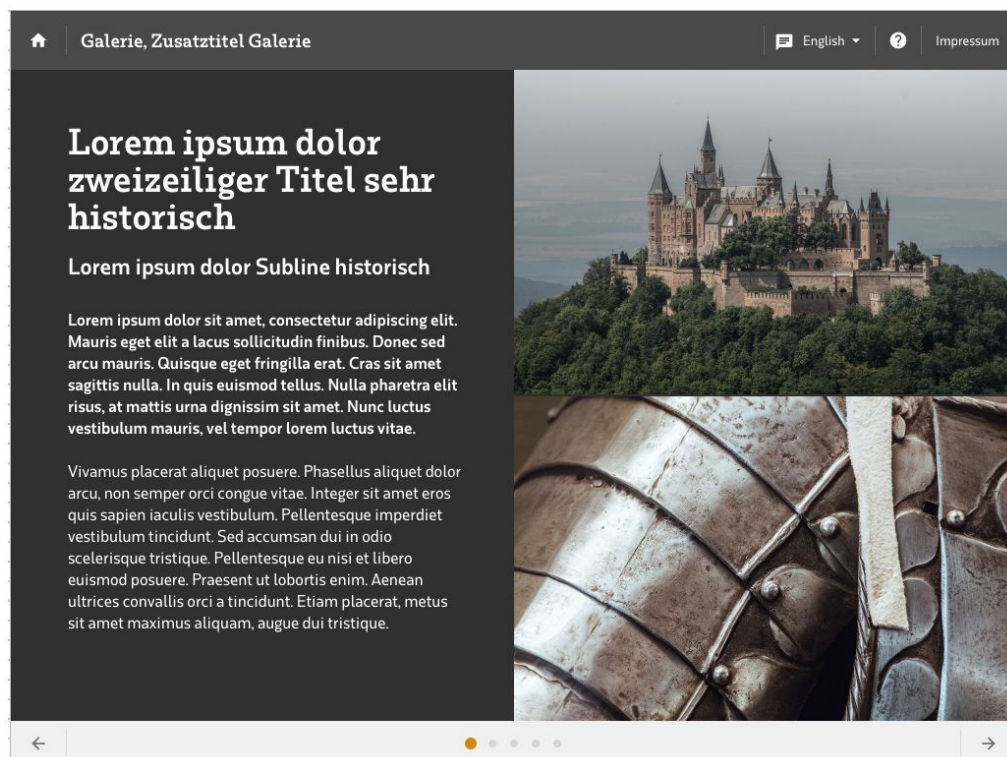
Menü 2:

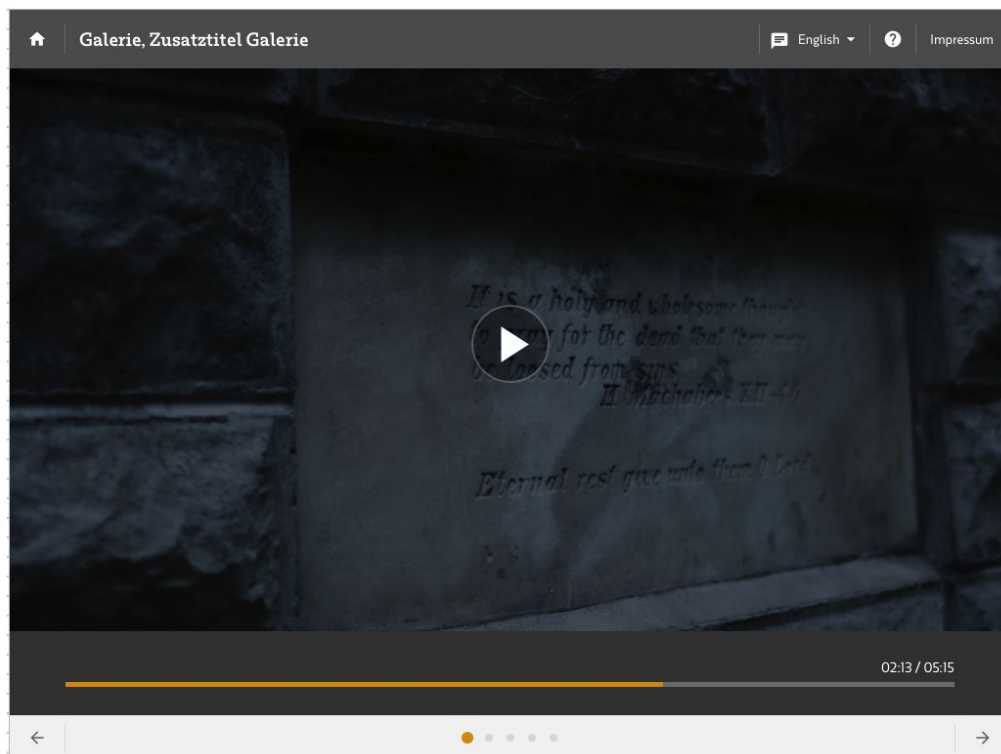
<div><div> Titel Sektion</div><div><div>English</div><div>?</div><div>Impressum</div></div></div>		
		
<div> Lorem ipsum dolor orilu zweizeiliger Titel</div>	<div> Lorem ipsum dolor orilu zweizeiliger Titel</div>	<div> Lorem ipsum dolor orilu zweizeiliger Titel</div>
		
<div> Lorem ipsum dolor orilu zweizeiliger Titel</div>	<div> Lorem ipsum dolor orilu zweizeiliger Titel</div>	<div> Lorem ipsum dolor orilu zweizeiliger Titel</div>

## Content Seite 1



## Content Seite 2





# Fernrohr (Stand 24.10.2018)

## AR-Applikation mit POIs

### Hardwarebeschreibung:

Die Hardware des Fernrohres besteht aus einem in zwei Achsen schwenkbaren Gehäuse welches einen Intel Nuc, Bildschirm und eine digitale Kamera enthält. Die Ausrichtung des Gehäuses (ab jetzt Fernrohr genannt) wird über zwei digitale Encoder und vier Entstopps (zwei pro Achse) festgestellt. An der Vorderseite befindet sich die digitale Kamera und an der Hinterseite der Bildschirm, welcher die Fernrohr-App dargestellt. Die Verarbeitung der Encoder-Daten und des Kamerasignales übernimmt der Intel Nuc PC.

### Softwarebeschreibung:

Auch hier wird ein Server-Client Modell verwendet. Der Server dient hier nur für die Auslieferung der Inhalte (POIs, Points of Interests). Clientseitig werden diese Inhalte, welche an eine bestimmte Ausrichtung des Fernrohres geknüpft sind, angezeigt. Weiteres ruft der Client periodisch die aktuelle Ausrichtung des Fernrohres ab, um die Inhalte dementsprechend anzuzeigen. Um kurze Netzerkennungsausfälle zu kompensieren, ist auch ein Caching Mechanismus im Client implementiert. Die Ausrichtung des Fernrohres kann durch den Benutzer geändert werden. Richtet der Benutzer den Sucher des Fernrohr auf ein POI (Point of Interest) aus, so erscheint ein Pop-Up, welches tiefergehende Informationen für diesen POI beinhaltet.

### Anforderungen:

- Die Applikation wird an die vorgegebene Hardware angepasst (9 Zoll Bildschirm)
- Der Hintergrund der Applikation ist der Live-Video Feed der Webcam.
- Im Vordergrund werden die einzelnen POIs in Form von POI-Bereichen (möglicherweise Kreisform mit Icon und opaken Hintergrund) als Overlay angezeigt. Siehe Beispielbild unten.
- Durch Anvisieren eines POI-Bereichs mit Hilfe des Fernrohres wird der POI aktiviert. Dadurch werden die Inhalte in den Vordergrund geladen und präsentiert. Wird der Fokus im Hintergrund von dem POI-Bereich genommen, schließt die Präsentationsebene des Inhaltes.
- Die Applikation unterstützt als Inhalt folgende Medien:
  - Text
  - Bild (jpg)
  - Audio (mp3)
  - Video (mp4) - darstellung von 3D Modellen



## CMS:

Als Backend wird TYPO3 mit einer proprietären Erweiterung verwendet. Es können POIs hinzugefügt werden. Ein POI besteht aus einer Mediendatei (Bild, Audio, Video, 3D), einer Überschrift, einer Unterüberschrift, einem Icon, einem Koordinatenpaar und dem Faktor des Radius.

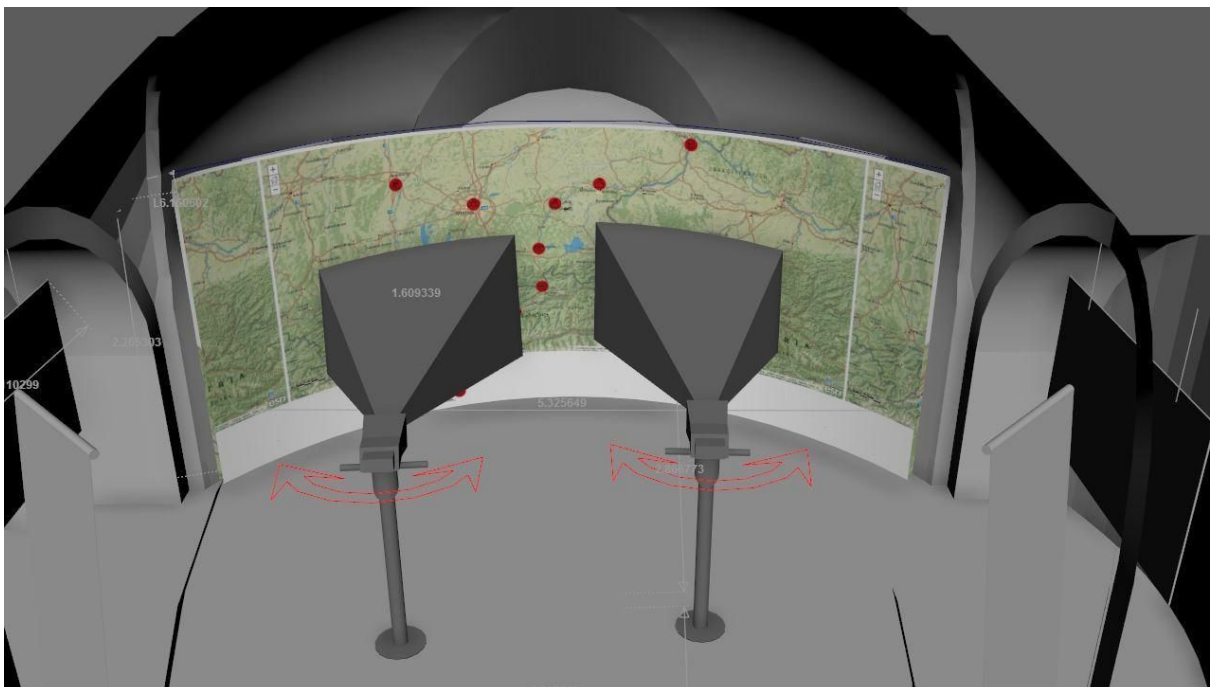
Bei multiplen Installationen des Fernrohres, kann die im CMS verwendete Seite kopiert und die POIs angepasst werden.

## Beispiele:

### 1. Beispiel POI-Bereich:



### 2. Beispiel Aufbau



## Möglichkeit des Offline-Betriebs

Da das Fernrohr mit einem Intel Nuc betrieben wird, steht genug Leistung zur Verfügung, so, dass nicht nur die Client-Software (Webbrowser mit App) sondern auch die Server-Software auf derselben physikalischen Maschine laufen kann. Das bedeutet, dass auf dem Nuc nicht nur Docker installiert und betrieben werden kann, sondern zudem auch der Docker-Container der den serverseitigen Teil Applikation enthält. Das Einpflegen der Daten ist dann durch einen Transfer eines Images möglich. Sinnvoll ist hier eine zweite Instanz dieses App-Containers für die Erstellung der Inhalte zu verwenden. Die Laufzeit dieses Containers kann als Image gespeichert werden und auf das Fernrohr (via USB-Speichermedium) übertragen und eingespielt werden.

# Dateimanagement

Das Dateimanagement ist eine Applikation, mit der die Daten in der ViSIT Medien-Datenbank verwaltet werden können. Diese Software wird ebenfalls mit dem Typo3-Container ausgeliefert, um eine einheitliche Oberfläche und Integration zu gewährleisten. Am LAS (Siehe Tabelle 1) wird dieser Container installiert und ausgeführt. Zugriff erfolgt via Browser (Adresse und Port müssen im Docker konfiguriert werden, siehe Tabelle 1)

Das Dateimanagement beinhaltet 2 grundlegende Bereiche.

1. Den Dateiupload und weitere Verarbeitung der Uploads
2. Den Dateibrowser
3. Die P2P Partner Verwaltung

Wird dieses Modul installiert, so werden einmalig eine syncthing ID und eine Uploader ID erstellt (könnte identisch sein). Diese Uploader ID wird in jedes MediaTripleData Json geschrieben um den uploader zu identifizieren (nur dieser kann das Access Level ändern). Zudem muss der Name der Institution angegeben werden. Dieser Name und die eigene syncthing-id wird der Datei hinzugefügt, die die Liste der ViSIT-Partner enthält. Dies wird benötigt um später Partner hinzuzufügen oder zu löschen.

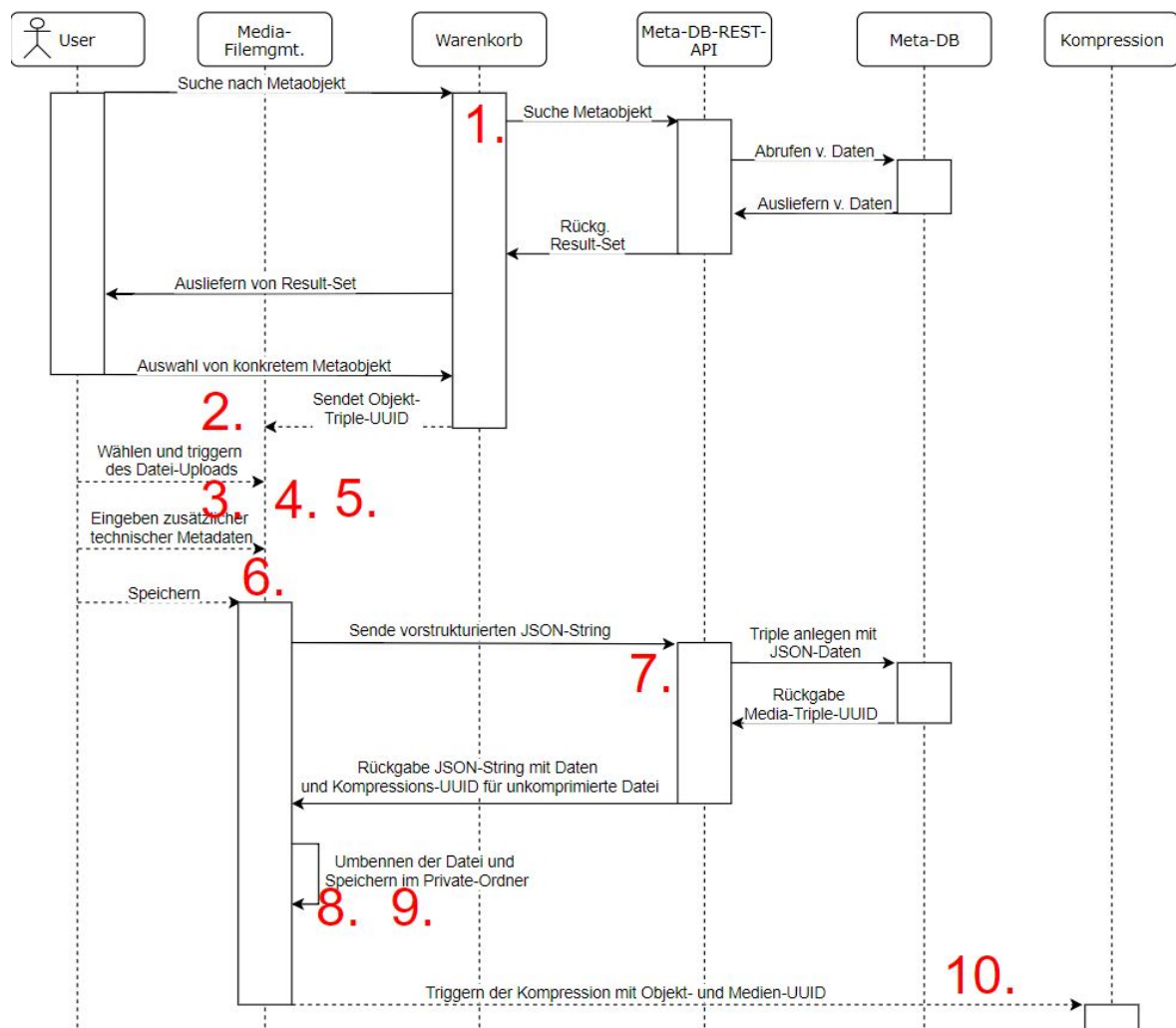
## Dateiformate

Siehe Dateiupload Punkt 3.

## Dateiupload

1. Der Benutzer sucht im Warenkorb nach einem Metadaten Objekt.
2. Wird der Benutzer fündig, kann dieser mit einem Klick auf den Button "Datei diesem Objekt hinzufügen..." (befindet sich unterhalb des Warenkorbes) das gefundene Objekt auswählen. Die im nächsten Schritt ausgewählten Dateien werden an das ausgewählte Objekt angehängt.
  - a. Existiert das Objekt nicht, muss dieses in der externen WissKI Oberfläche angelegt werden.
  - b. Der "Auswählen" Button wird nur eingeblendet, wenn ein Objekt im Warenkorb gefunden wurde. ("UUID != false" siehe [Inter-Iframe - Kommunikation](#); UUID wird verwendet um die objectTripleID im JSON, siehe unten zu setzen)
3. Der Benutzer kann nun auswählen, welchen Dateityp er hochladen möchte.  
([https://colorlib.com/polygon/gentelella/general\\_elements.html](https://colorlib.com/polygon/gentelella/general_elements.html) → Tab - Modell)
  - a. Einzeldatei Objekte (Bild, Video, Audio, etc.) → maximale Dateigröße 1GB
  - b. 3D-Modell Objekte: Komposition aus .obj, .mtl(optional), .jpg(optional) (siehe [Abgrenzungskriterien](#)) → maximale Dateigröße siehe [Hardwareanforderungen](#)

4. Der Benutzer kann nun die hochzuladenden Dateien von seinen lokalen Dateien auswählen.
5. Der Benutzer kann/muss nun weitere Datenfelder hinzufügen.  
Beschreibung: optional  
Urheber (owner): muss  
Uploader (uploader): Uploader ID (read only oder hidden)  
Digitalisator (creator) - Dienstleister/Person die das Digitalisat erstellt hat: optional  
OPT-IN - Box für Datenschutz (e.g. Mit dem Upload bestätige ich, dass ich keine Rechte dritter verletze.)
6. Der Benutzer klickt auf "Medien Dateien hochladen".
7. Die `objectTripleID` wird an die Meta API gesendet. Hier wird das MediaTriple (DigRep) erzeugt und die UUID `mediaTripleID` returned.
8. Die Dateien werden in den Privaten SyncThing Ordner kopiert. Dateiname ist `objectTripleID.mediaTripleID.origin.extension`
9. Es wird ein MediaTripleData - JSON erzeugt (siehe [Json "MediaTripleData"](#)). Alle Felder werden gesetzt: Kompressionsstufe "origin" ist die originale Datei. Weitere Stufen werden beispielsweise mit "full-hd" oder 2000 (für 2000 Vertices) bezeichnet
10. Das Json und die `mediaTripleID` wird nun an die Meta API gesendet (update MediaTriple)
  - a. Optional JSON im Cache File speichern
11. Das Json wird nun an den Kompressionscontainer API gesendet.



Zeichnung veraltet, bitte Beschreibung folgen.

## Dateibrowser

Der Dateibrowser bildet die zentrale Komponente. Hier werden alle Dateien im Visit Netzwerk aufgelistet und verwaltet. Diese Verwaltung benötigt mehrere Komponenten. Ein Objekt im RDF Store kann beliebig viele Medien besitzen, die wiederum beliebig viele

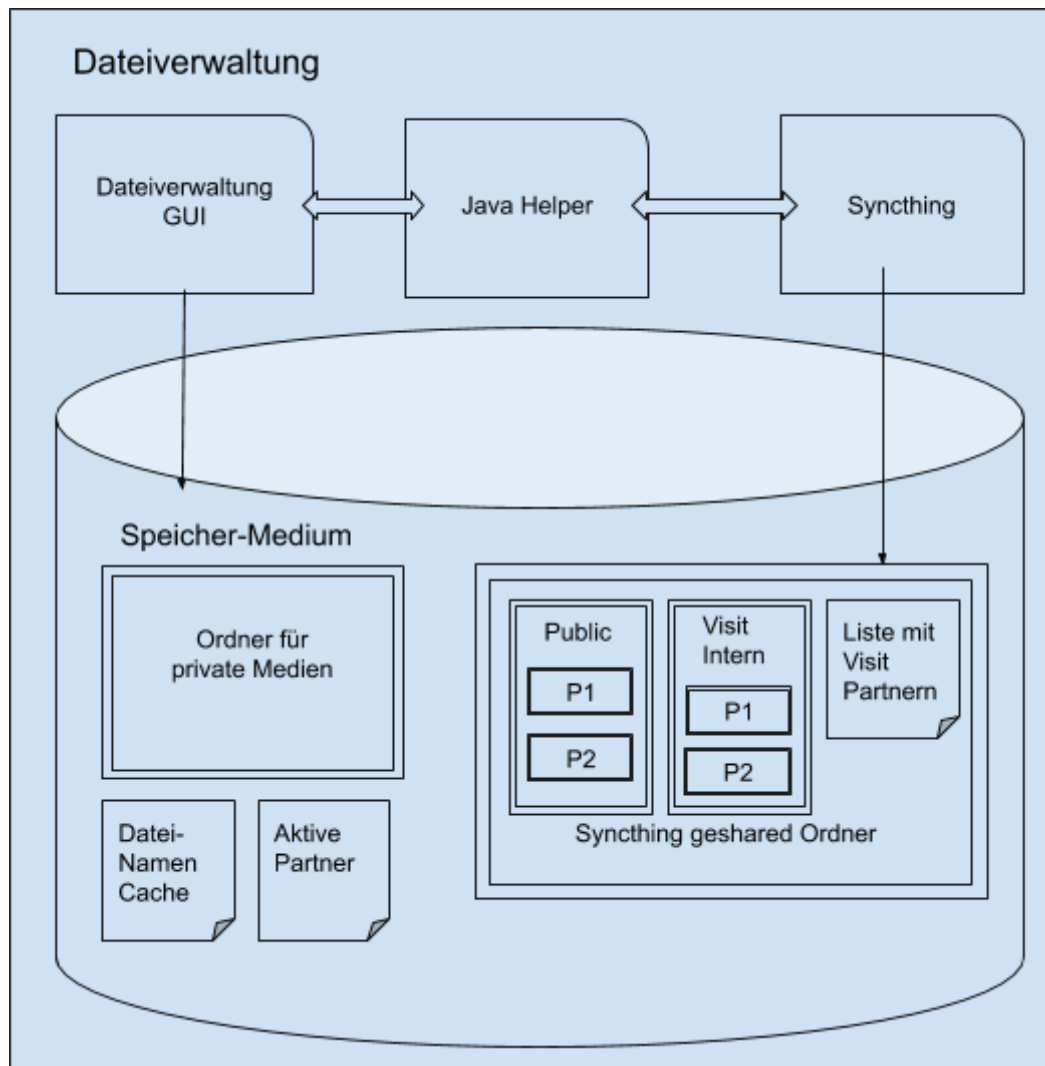
- Die Dateiverwaltung GUI erlaubt den Benutzer Dateien zu Listen
  - Da Diese Dateien als "objectTripleID.mediaTripleID.origin.extension" gespeichert werden, muss der "echte" Echte Dateiname gecached werden (in einer Datei beispielsweise)
  - Somit iteriert der File Browser über alle lokalen Dateien und lädt mit der im Dateinamen enthaltene ObjectTripleID in den Cache (Daten werden via die API GET: Gegeben *objectID* abgerufen)
  - Ein Listeneintrag wird tabellarisch angezeigt mit Medien-ID ("mediaTripleID"), Object-ID ("objectTripleID"), Beschreibung ("description"), Dateityp ("MIMEtype") und Erstellt ("createDate"). Ist die Berechtigung in den Kompressionsstufen gleich wird auch diese

angezeigt unter Berechtigung ("accessLevel") andernfalls wird hier "-" angezeigt

- Wird dieser Listeneintrag angeklickt so expandiert dieser und zeigt tabellarisch alle Kompressionsstufen an Berechtigung ("accessLevel") und Dateigröße ("fileSize")
- Die Dateiverwaltung erlaubt dem Benutzer die Berechtigungen zu ändern oder die Datei zu löschen
  - wenn er selber der besitzer dieser Dateien ist → Feld "uploader" ist im JSON MediaTripleData identisch mit der lokal Uploader ID
  - Wird die Berechtigung einer Kompressionsstufe geändert, so wird die Datei in den entsprechenden Ordner verschoben und der Eintrag in der Meta-Daten DB geändert. Syncthing rollt diese veränderung an die Anderen Clients aus.
  - Ein Eintrag kann gelöscht werden. Datei wird gelöscht und eintrag aus der Meta-DB gelöscht?

## P2P Partner Verwaltung

Da prinzipiell kein speicherlimit vorhanden ist, muss die Möglichkeit vorhanden sein Verschiedene Visit. In der Datei mit der Liste der ViSIT Partner, sind alle Syncthing IDs eingetragen. Wird ein Partner hinzugefügt, so wird dies über das ein Hilfsprogramm (<https://bitbucket.org/visit2016/syncthing-control/src/master/src/syncthing/resources/Help.txt>) abgewickelt. Der hinzugefügte Partner wird zusätzlich in der Datei vermerkt, die die aktiven Partner enthält. Syncthing sollte nun die Dateien dieses Partners lokal synchronisieren.



Somit ist ergeben sich folgende Dateipfade

/private/*	Private Dateien
/Sync/public/#partnername#/*	Öffentliche Dateien
/Sync/visit/#partnername#/*	Dateien für Visit intern
/Sync/visitMembers	Liste mit Visit Partner Format: #partnername#:syncthing id
/dataCache	Dateinamen Cache
/synchedMembers	Liste mit aktiven partner Format: #partnername#:syncthing id

# Metadatenbank, Depot

Die Metadatenbank, die im Hintergrund des Gesamt-Visit-Systems arbeitet, besteht aus zwei technischen Komponenten, die zusammen installiert werden müssen:

- Einem **RDF Triplestore**, der die Metadaten als RDF Triples speichert
- Einem **Drupal CMS**, welches **WissKI** als Modul unterstützt, und damit für die Speicherung der Daten im CMS und die Konvertierung der Daten für den Triplestore verantwortlich ist

## Softwarebeschreibung

Der aktuelle Stand<sup>1</sup> ist auf einem Server am Zentrum für Informationstechnologie und Medienmanagement (ZIM) an der Universität Passau installiert. Dieser unterstützt tägliche Backups und ist durch standardisierte Sicherheitsmaßnahmen geschützt. Die Adresse dafür lautet: <https://database.visit.uni-passau.de/drupal/>

Jegliche Triplestores sind mit dem WissKI System benutzbar, Feedback der WissKI Entwickler empfiehlt jedoch eine RDF4J Datenbank<sup>2</sup> zu benutzen. Das Drupal System<sup>3</sup> ist auf der neuesten Version 8.x.

Die meisten Triplestores, sowie auch die RDF4J, sind per Apache Tomcat<sup>4</sup> zu betreiben. Im aktuellen Stand besitzt dieser die Version 8.5.24.

Für die Installation von Drupal und WissKI sind folgende Anforderungen gegeben:

- Apache 2.x, aktuell installiert: 2.4.18
- MySQL 5.5.3 oder höher, aktuell installiert: 5.7.23
- PHP 5.5.9 oder höher, aktuell installiert: 7.0.32
- Für WissKI: Drupal 8.x.

## Hardwarebeschreibung

Die technischen Komponenten, die in diesem Überabschnitt aufgeführt sind, benötigen per se nur genügend Hardware, um die Anforderungen die im Abschnitt "Softwarebeschreibung" aufgeführt sind, zu betreiben. Diese sind zur heutigen Zeit verhältnismäßig vernachlässigbar, jedoch könnte eine bessere Hardware-Unterstützung zu einer besseren Performanz der Metadatenbank führen, gerade wenn viele Benutzer gleichzeitig darauf tätig sind.

Dies kann zum aktuellen Zeitpunkt noch nicht abgeschätzt werden, jedoch könnte die aktuell laufende Testphase Aufschluss darüber geben. Etwaige Ergebnisse werden ergänzt.

---

<sup>1</sup> 9.10.2018

<sup>2</sup> <http://rdf4j.org/>

<sup>3</sup> <https://www.drupal.org/>

<sup>4</sup> <http://tomcat.apache.org/>



## WissKI (Warenkorb)

Das WissKI Framework unterstützt die Interaktionen mit den Metadaten, darunter:

- Das **Erzeugen** von Metadaten, orientiert an den Hauptentitäten des Visit Models VisMo:
  - *Activity* (Ereignisse, Vorkommnisse, Aktivitäten)
  - *Architecture* (Architekturobjekte jeglicher Art)
  - *Group* (Gruppen, Vereine, Bruderschaften, Sozietäten)
  - *Institution* (Institutionen, die am Metadaten austausch teilnehmen, also idR. Museen)
  - *Object* (Ausstellungsobjekte)
  - *Person* (Personen)
  - *Place* (Orte)
  - *Reference* (Literatur)
- **Einfaches Browsen** anhand der Titel der erzeugten Entitäten (Navigate)
- **Detailliertes Suchen** der erzeugten Entitäten (Find)
- **Abändern** bereits erzeugter Entitäten, indem eine Entität per Browsen oder Suchen ausgewählt und dann per Eingabemaske geändert wird

Eine Änderung/Erweiterung/Anpassung des Metadatenmodells ist **nicht** ohne weiteres möglich. Dies benötigt einen Domänen-Experten mit WissKI-Kenntnissen und ist mit erheblichem Zeitaufwand verbunden.

## Inter-Iframe - Kommunikation

Bei jedem Seitenaufruf wird ein Javascript ausgeführt, welches die aktuelle URL als auch die Triple-UUID (falls vorhanden, andernfalls "false") über `window.parent.dispatchEvent` an das Parent-Frame übergibt.

(<https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage>)

## REST API / Technischer Output der Metadatenbank

Im Rahmen des Visit Projekts wird eine REST API neben der Metadatenbank entwickelt, die auf den Metadaten-Bestand des Triplestores zugreifen kann und angepasste Informationen zurückliefert.

Der REST API wird unter folgendem Standard-Pfad exposed:

<https://database.visit.uni-passau.de/metadb-rest-api/>

Die einzigen Anforderungen, die diese API umsetzen wird, sind die folgenden:

## API für allgemeine Objekte

Gegeben die ID einer Entität gibt die API eine JSON Repräsentation der Entität zurück. Dabei werden, orientiert am REST Standard, die primitiven Datentypen direkt mitgegeben.

Eine Referenz auf andere Objekte beinhaltet wiederum eine ID, die (mit der hier beschriebenen Funktionalität) aufgelöst werden kann.

Zu vermerken ist hier, dass die CIDOC CRM Ontologie (auf welcher VisMo aufbaut) intern weitere Entitäten benutzt, um gewisse Umstände zu modellieren. An diesen entsprechenden Stellen löst die API diese auf, um einfachere Informationen zu propagieren.

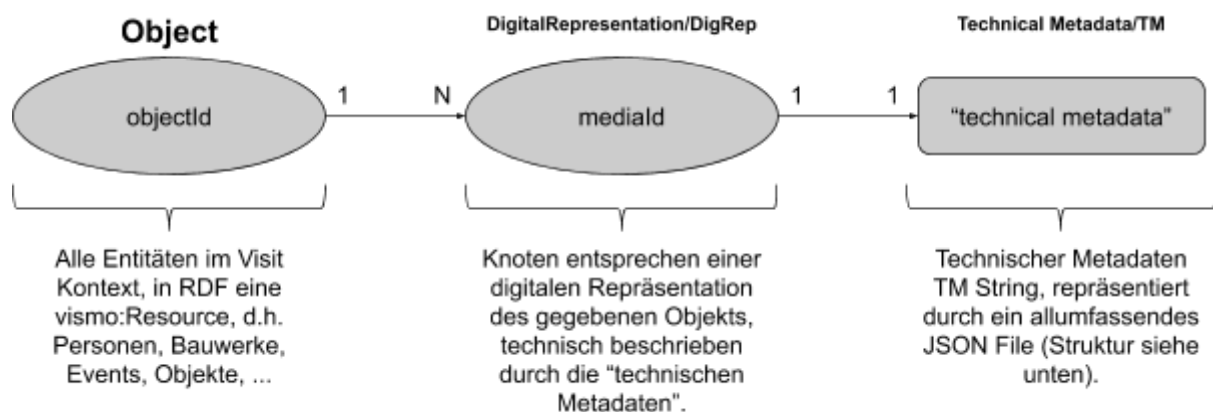
Zu obiger Funktionalität bietet die API an, die Repräsentation des Objekts ebenfalls per WissKI View Path anzufragen. Dieser wird intern automatisch aufgelöst und auf die eigentliche objectID gemapped.

Damit bietet die API folgende Methode für allgemeine Objekte:

HTTP	API Pfad (relativ zur API)	Request Params	Kurzbeschreibung
GET	/object	id	Gibt JSON Repräsentation des Objekts mit der ID objectID zurück.
GET	/wisskiobject	wisskipath	Gibt JSON Repräsentation des Objekts mit der dem WissKI View Path wisskipath zurück.

## API für die technischen Metadaten

Die technischen Metadaten basieren auf folgendem "Schema":



Die REST API stellt dafür folgende Methoden zur Verfügung:

HTTP	API Pfad (relativ zur API)	Request Params	Kurzbeschreibung
GET	/digrep/media	id	Gibt TM String des mit der ID spezifizierten DigRep Knotens zurück.
PUT	/digrep/media	id,	Updated den TM String durch "newData" des per

		newData	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	mediald, objectld	Löscht den DigRep Knoten mit der mediald, zugehörig zum Objekt mit der objectld.

Eine Swagger-Beschreibung der API wird ebenfalls unter folgendem Pfad zur Verfügung stehen:

<https://database.visit.uni-passau.de/metadb-rest-api/swagger-ui.html>

## Json Formate

### Json "MediaTripleData"

```
{
    "title"                : string*,           /* Titel über
Objekt vom Benutzer */
    "description"          : string,           /* Beschreibung
über Objekt vom Benutzer */
    "objectTripleID"       : TripleID*,       /* ID des Objektes
das Abgebildet wurde */
    "objectTripleURL"      : TripleURL*,       /* ViSIT RDF URL
des Objektes das Abgebildet wurde */
    "mediaTripleID"        : TripleID*,       /* ID des Medien
Triples welches dieses JSON Enthält */
    "mediaTripleURL"       : TripleURL*,       /* ViSIT RDF URL
des Medien Triples welches dieses JSON Enthält */
    "createDate"           : date*,           /* Erstelldatum */
    "creatorID"             : creatorID*,       /*
Einzigartiger Fingerabdruck des ViSIT Partners um Upload zuweisen
zu können */

    "creatorName"          : string*,           /* Name des
Visit Partner um Upload zuweisen zu können */

    "rightholder"          : string*,           /*
Rechteinhaber / Urheber kann beim Upload angegeben werden */
    "uploader"             : string,           /* Name des MA der
```

```

der Datei hochgeladen hat */
    "MIMEtype"          : string,                /* Datei Typ e.g.
Jpg */
    "files" :
    {
        origin :                               /* Stufe 0
ist die Original Datei*/
        {
            "uploadDate" : date*,              /* Datum
wann die Kompression hinzugefügt wurde */
            "accessLevel" : accessLevel*,      /*
Sichtbarkeit vom Medieninhalt */
            "license" : string,                 /*
Lizenz (träger und Art) vom Medieninhalt */
            "fileSize" : long*,                 /*
Dateigröße (Summe aller Dateien) */
            "paths" : [Array von string]*,
/* interne Dateipfade */
            "fileTypeSpecificMeta" : json       /*
Weitere Metadaten für den Inhalt */
        },
        1 : { ... }
        2 : { ... }
    }
}

```

**TripleID:** String im Format 13 HexDez Zeichen (bsp.5bbdff6a6ce23)  
**TripleURL:** String im http://visit.de/data/5bab75474d51b  
**Date:** long – Unix Zeitstempel in Sekunden  
**accessLevel:** String entweder **public**, **private** oder **visit**  
**creatorID:** Syncthing ID. Format:  
SDS23M6-4BAKFJZ-DHR6WZW-VRUYYLJ-ZM2SWTS-N43IZBY-GAUNYKH-S5NMJQP

## Json “AlleMediaTripleData”

```

{
    "objectId" : TripleID,
    "digitalRepresentations" : [
        *Array von mediaTriples*
        =
        {mediaId: TripleID,
        technicalMetadata: *JSON siehe oben*}
    ]
}

```


# Was die Metadatenbank/WissKI **nicht** kann

Im Folgenden werden Anforderungen aufgeführt, die zunächst Grenzen der abgeklärten Implementierung der Metadatenbank aufweisen. Diese Features sind **planmäßig nicht im Visit-Projektrahmen umzusetzen**, können aber, für den Fall dass Ressourcen frei werden bzw. hinzugezogen werden, noch umgesetzt werden. Entsprechende Anforderungen werden ergänzt.

- Import von Metadaten per LIDO Standard oder Excel Datei. Dies ist prinzipiell per WissKI möglich, benötigt aber einen großen Konfigurationsaufwand.
- Ausführliche Provenance Informationen: WissKI unterstützt bisher nur die Anzeige des Daten-Erstellers. Dies ist im Moment jedoch noch fehlerhaft.

The screenshot shows the 'Edit New Architecture' form in the Visit Database interface. The form is divided into several sections, each with a title, a text input field, and an 'Add another item' button. The sections are: 'Name', 'Sakralbau', 'Profanbau', and 'Kirchliche Gliederung'. Each section also has a 'Show row weights' button. The 'Name' section has a placeholder text: 'Geben Sie hier den eindeutigen Namen des Architekturobjekts an, die Sie anlegen wollen, z.B. "Veste Oberhaus".' The 'Sakralbau' section has a placeholder text: 'Wenn es sich bei der Architektur um einen Sakralbau handelt, können Sie hier den Bautyp spezifizieren, z.B. Dom, Kapelle, Wallfahrtskirche usw.' The 'Profanbau' section has a placeholder text: 'Wenn es sich bei der Architektur um einen Profanbau handelt, können Sie hier den Bautyp spezifizieren, z.B. Ringburg, Festung, Kastell usw.' The 'Kirchliche Gliederung' section has a placeholder text: 'Handelt es sich um einen Sakralbau, geben Sie hier bitte die aktuelle Zugehörigkeit zu einem Bistum bzw. einer Landeskirche an.' On the right side of the form, there is a 'Tools' sidebar with a search bar and an 'Add content' button. The top navigation bar includes links for 'Home', 'Find', 'Navigate', 'Create', 'My account', and 'Log out'.

Bild: Erstellen eines Architecture Entities


[Visit Database](#)
[Home](#)
[Find](#)
[Navigate](#)
[Create](#)
[My account](#)
[Log out](#)

[Navigate](#) / [Object](#) / Ansicht von Dom und Residenz zu Passau

## Ansicht von Dom und Residenz zu Passau

[View](#)
[Edit](#)
[Delete](#)
[Triples](#)

**Titel**  
Ansicht von Dom und Residenz zu Passau

**Objektbezeichnung**  
Kupferstich

**Inskrift**  
**Text**  
Prospectus Ecclesie Cathedralis Passavienis ex parte postica et ? foro publico adspiciendus. - Prospect des hintern theils von der Dom Kirch in Passau, wie solche vom Markt oder Platz anzusehen ist

**Art**  
Text

**Anbringung**  
Vorderseite unten

**Art**  
Signatur

**Signatur**  
F.B. Werner delin

**Anbringung**  
Vorderseite unten links

**Art**  
Signatur

**Signatur**  
Martin Engelbrecht execud. Aug. Vindel

**Anbringung**  
Vorderseite unten rechts

**Inventarnummer**  
655


**Beschreibung**  
Das Blatt zeigt den Residenzplatz mit der Neuen fürstbischöflichen Residenz, dem Dom und dem Marschallhaus. Diese sogenannte Neue Residenz, d.h. die Erweiterung der Residenz nach Osten hin, erfolgte unter Fürstbischof Johann Philipp von Lamberg. Die Voraussetzungen für eine repräsentative Fassadengestaltung waren erst mit dem Erweiterungsbau auf einen offenen Platz hin geschaffen worden. Die Ausführung der kompletten Fassade wurde unter Fürstbischof Firmian im Rokoko-Stil circa 50 Jahre später, in der 2. Hälfte des 18. Jahrhunderts in Angriff genommen.

**Literaturangaben**  
[Passau - Mythos & Geschichte, , 2007](#)

**Institution/Eigentümer/Verwalter**  
[Oberhausmuseum](#)

**Standort (Museum)**  
[Oberhausmuseum](#)

**Herstellung**



### Tools

[Add content](#)

Bild: Betrachten eines Object-Entities: Ansicht von Dom und Residenz zu Passau

# Kompressions-Komponente

Die Kompressions-Komponente dient zur Kompression von Mediendaten, die im Dateimanagement auf dem LAS registriert sind. Die Kompressions-Komponente läuft innerhalb eines separaten Docker-Containers auf dem LAS. Eine direkte Interaktion des Benutzers mit der Kompressions-Komponente findet in der Regel nicht statt. Es wird jedoch eine Web-Oberfläche zur Administration und Konfiguration des Systems bereitgestellt.

## Hardwarevoraussetzungen

Die Anforderungen an die Ressourcen, die sich für den LAS bzw. den Docker-Container der Kompressions-Komponente ergeben, hängen in erster Linie von der Größe der zu komprimierenden Mediendateien ab. Während die Prozessorleistung primär die Dauer der Kompressions-Prozesse bedingt, beschränkt der zur Verfügung stehende Arbeitsspeicher die maximale Größe der Eingabedaten nach oben.

Da der Algorithmus zur Kompression von 3D-Modellen kaum Parallelisierungs-Möglichkeiten bietet, werden die Berechnungen derzeit nicht auf mehrere Prozessorkerne verlagert. Es wird jedoch nicht ausgeschlossen, dass in Zukunft die Funktionalität zum parallelen Ausführen mehrerer (kleinerer) Kompressions-Aufträge implementiert wird, wobei diese Berechnungen dann auf mehrere Kerne verlagert werden können.

Bisherige Experimente zeigen, dass Modelle mit ca. 8 Mio. Dreiecken auf einem aktuellen Bürorechner (Intel-Core-i7-Prozessor, 32 GB RAM) komprimiert werden können. Konkrete Aussagen können erst nach Experimenten mit den implementierten Algorithmen gemacht werden.

## Softwarevoraussetzungen

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

- Java Runtime Environment<sup>5</sup> (aktuell installiert 1.8.0)
- ImageMagick<sup>6</sup> (aktuell installiert 7.0.8)

## 3D-Kompressions-Algorithmen

Zur Kompression von 3D-Modellen werden zwei verschiedene Algorithmen angeboten:

- Target-Size-Kompression: Bei diesem Kompressions-Algorithmus wird das Eingabemodell auf eine vorgegebene Anzahl an Vertices komprimiert. Für sehr

---

<sup>5</sup> <https://java.com/de/download/>

<sup>6</sup> <https://imagemagick.org/script/index.php>

kleine Zielgrößen kann die Kompression fehlschlagen, wenn das Eingabemodell zu komplex ist. Der Algorithmus basiert auf Edge-Collapse-Operationen, die anhand von Quadriken bewertet werden.<sup>78</sup>

## Administration & Konfiguration

Die Kompressions-Komponente beinhaltet einen Webserver, über welchen die Administration und Konfiguration des Systems komfortabel über eine Web-Oberfläche möglich ist.

Diese gibt einerseits einen Überblick über die ausstehende Kompressions-Aufträge und erlaubt andererseits ein Pausieren oder Stoppen des Systems. Außerdem lassen sich im Archiv die letzten ausgeführten Kompressions-Aufträge einsehen. Überdies hinaus lassen sich folgende Konfigurations-Parameter festlegen:

- Maximale Länge der Auftragsliste
- Portnummer der API-Schnittstelle
- Automatisches Abarbeiten der Auftragsliste beim Serverstart
- Zugriffsberechtigte IP-Adressen
- Zu erstellende komprimierte Modelle (Anzahl der Vertices bei Target-Size-Kompression, Erstellung eines Modells mittels progressiver Kompression), wenn beim Absetzen des Auftrags nicht anders angegeben
- Konfiguration der Textur-Kompression:
  - Festlegung von Schwellwerten bzgl. Vertex-Anzahl, durch welche Intervalle an Geometrie-Komplexität definiert werden
  - Angabe der Größe der Textur pro Intervall

---

<sup>7</sup> <https://www.cs.cmu.edu/~garland/Papers/quadrics.pdf>

<sup>8</sup> <https://www.cs.cmu.edu/~garland/Papers/quadric2.pdf>





Bild: Web-Oberfläche der Kompressions-Komponente - Überblick über durchgeführte Aufträge im Archiv

## Abgrenzungskriterien

Das Absetzen eines Kompressions-Auftrags wird in jedem Fall von der Dateimanagement-Komponente angestoßen. Eine Benutzeroberfläche zum manuellen Absetzen von Kompressions-Aufträgen wird nicht notwendigerweise zur Verfügung gestellt.

An Medientypen werden Bilder und 3D-Modelle unterstützt. Andere Medientypen können mit dieser Komponente nicht komprimiert werden. An Bilddaten können alle Bildformate komprimiert werden, die von der Software ImageMagick unterstützt werden. An 3D-Modellen können texturierte und nicht-texturierte Modelle komprimiert werden, die im OBJ-Format gespeichert werden. Andere Dateien müssen zunächst manuell durch Drittsoftware in das OBJ-Format übersetzt werden. Pro Modell ist maximal eine Texturdatei zulässig. Genauere Informationen über die für ein 3D-Modell notwendigen und möglichen Dateien sind in folgender Tabelle dargestellt.

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

Tabelle: Notwendige und mögliche Dateien für 3D-Modelle

Zu Beachten ist der Arbeitsspeicher des Servers → Siehe [Hardwareanforderungen für die Kompressions - Komponente](#)

Bei 3D-Modellen wird vorausgesetzt, dass die Eingabedaten den Voraussetzungen eines *Manifold Mesh*<sup>9</sup> genügen. Für Modelle, welche diese Eigenschaft nicht aufweisen, wird keine Garantie bezüglich des Ergebnisses des Kompressionsvorgangs gegeben. Mögliche Konsequenzen sind ein Fehlschlagen des Kompressionsvorgangs oder unbrauchbare Resultate. Außerdem darf das Modell ausschließlich aus Dreiecken bestehen. Faces höherer Ordnung sind nicht zulässig.

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

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

---

<sup>9</sup> <https://pages.mtu.edu/~shene/COURSES/cs3621/SLIDES/Mesh.pdf>