easyLAC

Inhaltsverzeichnis

Quickstart

Überprüfe, dass node.js installiert ist.
Öffne den Terminal/Konsole.

1. Klone das Repo

git clone https://github.com/pbd84/eLAC.git

2. Gehe in den Ordner

cd eLAC/elac

3. Installiere alle Module als Dependencies in package.json (einmalig)

npm install

4. Builds und serves die eLAC-APP.

ng serve

Kurzbeschreibung

Das Projekt easyLAC (eLAC) entstand im Rahmen des Seminars “Digital Humanities und Informatik der Geisteswissenschaften” im Wintersemester 2019/2020 und wurde von Anne Gerlach und Nils Geißler in Zusammenarbeit mit dem Data Center for the Humanities (DCH) und dem Regionalen Rechenzentrum Köln (RRZK) umgesetzt. Grundlage für das Projekt war der Wunsch nach einem einfacheren Zugriff auf im Language Archive Cologne (LAC) hinterlegte Daten für Probanden des Projekts “Variation and Language Attitudes in the Yurakaré: Towards a Language-Comparative Perspective” des Instituts für Linguistik der Universität zu Köln.

Im Laufe des Projektes wurden Anforderungen an die zu erstellende Softwarelösung festgelegt, Möglichkeiten zur Nachnutzung technischer Infrastrukturen ermittelt und schließlich in mehreren Phasen ein funktionsfähiger Prototyp entwickelt.

Die so erstellte Web-App “eLAC” nutzt die vorhandene Infrastruktur des LAC, einem Backend mit offenen Schnittstellen für verschiedene Aufgaben, und dient damit als eine Art Proof-of-Concept.

Der im Projekt entstandene Code teilt sich auf zwei Repositorien auf:

- Transformationen zur Erzeugung multilingualer Metadaten
- Angular Web-App

Die Dokumentation ist auch online verfügbar:

- Dokumentation

Was ist das LAC?

Die Selbstbeschreibung des LAC vom 09.03.2020:
(https://dch.phil-fak.uni-koeln.de/bestaende/repositorien/language-archive-cologne):

> *“Das ‘Language Archive Cologne’ (LAC) ist ein Forschungsdatenrepositorium für die Sprachwissenschaften und alle geisteswissenschaftliche Disziplinen, die mit audiovisuellen Daten arbeiten. Das Archiv bildet einen Cluster des Data Center for the Humanities in Kooperation mit dem Institut für Linguistik der Universität zu Köln.*
>
> *Wir unterstützen Forschung und Lehre, indem wir qualitativ hochwertige digitale Sprachressourcen archivieren und einen webbasierten, offenen Zugang zu den Forschungsdaten ermöglichen. Wir helfen damit den Fachdisziplinen, einen nachhaltigen Umgang mit ihren Daten nach guter wissenschaftlicher Praxis und unter Beachtung maßgeblichen Standards zu realisieren und leisten konkrete technische und methodologische Beratung im Forschungsdatenzyklus – von der Erhebung der Daten, über deren Aufbereitung und Archivierung, bis zur Publikation und Nachnutzung.”*

eLAC: über das Projekt

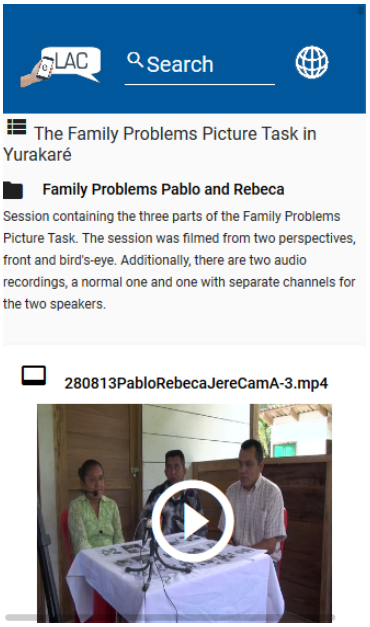
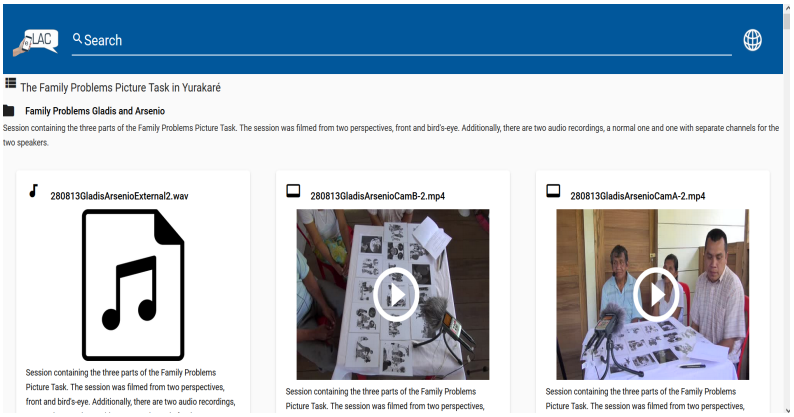
Das Projekt entstand aus dem Wunsch, einen vereinfachten Zugriff zu im LAC hinterlegten Mediendateien zu bieten. Zwar ist der Zugriff über das LAC auf diese Dateien möglich, aber der Weg dorthin ist verglichen mit üblichen Ansätzen, wie später in einem kurzen Abschnitt über einen Usability Test verdeutlicht wird, umständlich. Zudem bietet das LAC bisher nur einen englischsprachigen Zugang sowie englischsprachige Metadaten. Beide Begebenheiten sind durch den wissenschaftlichen Hintergrund begründet und nachvollziehbar. Nichtsdestotrotz wird dadurch der Zugang zu den Daten erschwert. Zudem ist davon auszugehen, dass die Zielgruppe der neuen Softwarelösung vor allem mit Mobilgeräten auf die Daten zugreifen wird.

Es wurden zwei wichtige Punkte zur Erleichterung des Zugriffs definiert:

1. Eine vereinfachte Präsentation der Daten (ähnlich gängigen Portalen wie YouTube etc.), optimiert für mobile Endgeräte.
2. Die Implementierung von Mehrsprachigkeit.

Darüber hinaus wurde entschieden, einen möglichst allgemeinen Ansatz zu wählen, um eine Nachnutzbarkeit für die Daten anderer Projekte zu ermöglichen.

Auf dieser Grundlage entstand eine Web-App, die mit Hilfe von Angular 9 implementiert wurde. Durch diesen Ansatz wird eine Responsivität der Ansicht gewährleistet. Angular ist zudem das zur Zeit für das Frontend des LAC verwendete Framework. Durch den modularen Aufbau (Components) des Frameworks werden sich die im Rahmen von eLAC erstellten Bestandteile bei Bedarf leichter in das bestehende Frontend übertragen lassen. Wie die Responsivität umgesetzt wurde, ist in Abbildung 1 zu erkennen. Hier sieht man die Ergebnisliste der Mediendateien der Yurakare-Collection.

Smartphone-Ansicht	Desktop-Ansicht
	

Abbildungen 1 (Smartphone) und 2 (Desktop): Vergleich der responsiven Darstellungsvarianten

Zur sprachunabhängigen Darstellung wurde eine Icon-basierte Präsentation gewählt, die sich an etablierten Lösungen orientiert (Motto: “steal from the best”). Das heißt statt sprachlichen Ausdrücken werden Icons verwendet und Metadaten wie Titel oder Beschreibung werden deutlich voneinander getrennt angezeigt.

Für die Mehrsprachigkeit der Metadaten selbst wurde eine Pipeline entwickelt, die es Wissenschaftlern ermöglicht, ihre Metadaten in zusätzliche Sprachen zu übersetzen und an die bestehende Struktur anzuhängen.

Die Web-App eLAC entstand in Zusammenarbeit mit dem Data Center for the Humanities (DCH) und dem Regionalen Rechenzentrum Köln (RRZK). Das RRZK leistete dabei Hilfestellung im Umgang mit den in der Infrastruktur des LAC enthaltenen APIs. Zudem wurden vom RRZK die Voraussetzungen für die Mehrsprachigkeit der Metadaten in die Wege geleitet und in Absprache mit dem DCH umgesetzt. Das DCH half außerdem bei der Implementierung beziehungsweise Übertragung der ersten Version des realisierten Prototypen in Angular in die bestehende Struktur des LAC.

Beim derzeitigen Stand (Mitte März 2020) ist das Feature der Mehrsprachigkeit aufgrund von nötigen Veränderungen im Backend noch in Entwicklung.

Arbeitsabläufe

(Mehrsprachigkeit, Technologien: BLAM, Oxygen bzw. XSLT/XML, XProc, Python, OCFL, REST APIS)
Der zu diesem Kapitel erstellte Code befindet sich im Repository Transformationen zur Erzeugung multilingualer Metadaten.

Die mehrsprachigen Metadaten erfüllen zwei zentrale Funktionen:

a. QUERY: mehrsprachige Suchanfragen werden ermöglicht.
b. FRONTEND: im Frontend kann ein multilinguales Feature eingebaut werden, welches auf die übersetzten Metadaten (“title” und “description”) zugreift.

Um die mehrsprachigen Metadaten zu verwenden, müssen diese zunächst generiert und dann auf den Server des LAC geladen werden. Dafür gelten folgende Restriktionen; erstens müssen die Daten dem vom LAC verwendeten BLAM Schema (Basic Language Archive Metadata) entsprechend valide sein und sie müssen in die OCFL (Oxford Common File Layout) Struktur als Resources integriert werden.

1. Multilinguale Metadaten generieren

a. TranslationPattern: Vorlage zur Übersetzung erstellen

Für die Übersetzungen dienen die im LAC liegenden englischen Metadaten-Dateien als Vorlage. Diese sind letztlich Duplikate der englischen Ausgangsdatei mit entsprechenden Übersetzungen. Beide müssen nach dem BLAM Schema valide sein. Die englische

Metadaten-Dateien können pro Bundle jeweils über folgenden Curl-Befehl abgerufen werden (siehe Datei *getblambundles.txt* für die Yurakare Collection).

curl -H "Accept: application/x-cmdi+xml" -X GET
https://lac.uni-koeln.de/bundle/**11341/00-0000-0000-0000-1BF6-F**>
Family_Problems_Gladis_and_Arsenio.xml

Hierfür muss das Handle des Bundles angegeben werden (siehe in bold), für welches die Metadaten Datei heruntergeladen werden soll. Als Name wurde in diesem Beispiel derselbe gewählt, in dem die Datei bereits vorliegt (siehe in italic). Die Handles sind auf der Seite des LAC auffindbar. Die heruntergeladenen Dateien liegen unter *bundle_blams_eng*.
Die englische Ausgangsdatei (*Family_Problems_Gladis_and_Arsenio.xml*) dient als Vorlage für die mehrsprachigen Dateien, weshalb aus dieser die Elemente extrahiert werden, die übersetzt werden sollen. Mit der Pipeline ***generatetranslationsheets.xpl*** wird dann eine Translation Pattern generiert und unter *translate_pattern* gespeichert.

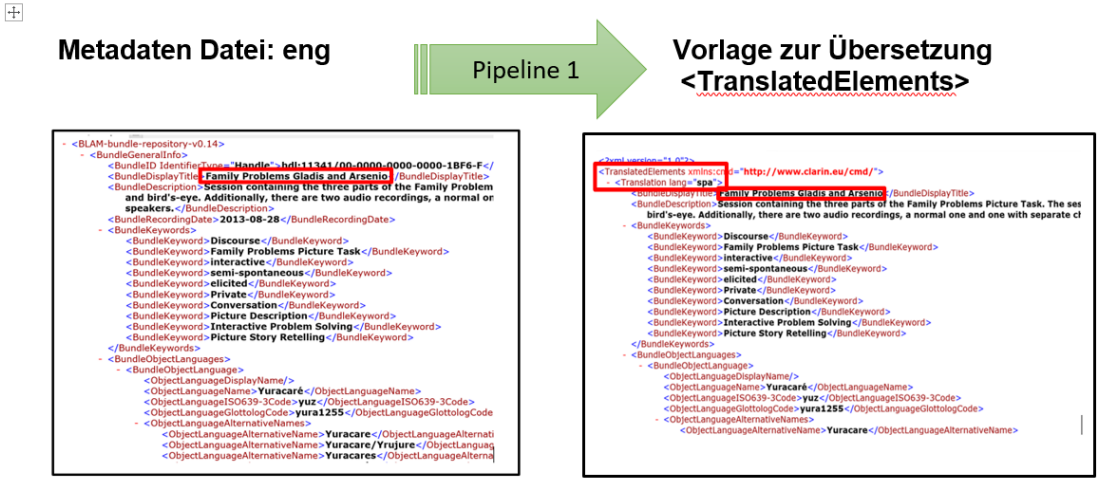


Abbildung 3: Erstellung der Übersetzungsvorlagen

Das Translation Pattern trägt denselben Namen wie die englische Metadaten-Datei, aber den Beisatz "_translate" (*Family_Problems_Gladis_and_Arsenio_translate.xml*). Pro Sprache, die für die Mehrsprachigkeit verfügbar sein soll, enthält diese Datei beispielsweise einen Abschnitt mit dem Tag <*Translation lang = "spa"*>. In dem verwendeten Beispiel folgt anschließend der Abschnitt <*Translation lang = "yuz"*>, welches den zweiten Übersetzungsteil ins Yurakare einleitet. Beide Abschnitte enthalten dieselben zu übersetzenden Elemente. Das Translation

Pattern kann nun an den Ersteller der Metadaten zurückgeschickt werden, der die Elemente in der Vorlage je nach Sprache übersetzt.

b. TranslatedPattern wieder zu Metadaten zusammenfügen

Die fertigen Translation Pattern (bspw. *Family_Problems_Gladis_and_Arsenio_translate.xml)* im Ordner *translated_pattern* werden dann im zweiten Schritt mit einem Python-Skript ***mergeblamengandtranslation.py*** wieder in das gültige BLAM-Schmema zurücktransformiert, nur dass nun keine englischen, sondern multilinguale Metadaten enthalten sind. Pro Sprache wird eine Metadaten Datei erstellt. Im vorliegenden Beispiel bedeutet dies, dass aus dem Translation Pattern, welches Elemente aus dem Spanischen und dem Yurakare, je eine Datei pro Sprache generiert wird (s. Ordner *bundle_blams_multiling*: *Family_Problems_Gladis_and_Arsenio_spa.xml* & *Family_Problems_Gladis_and_Arsenio_yuz.xml*).

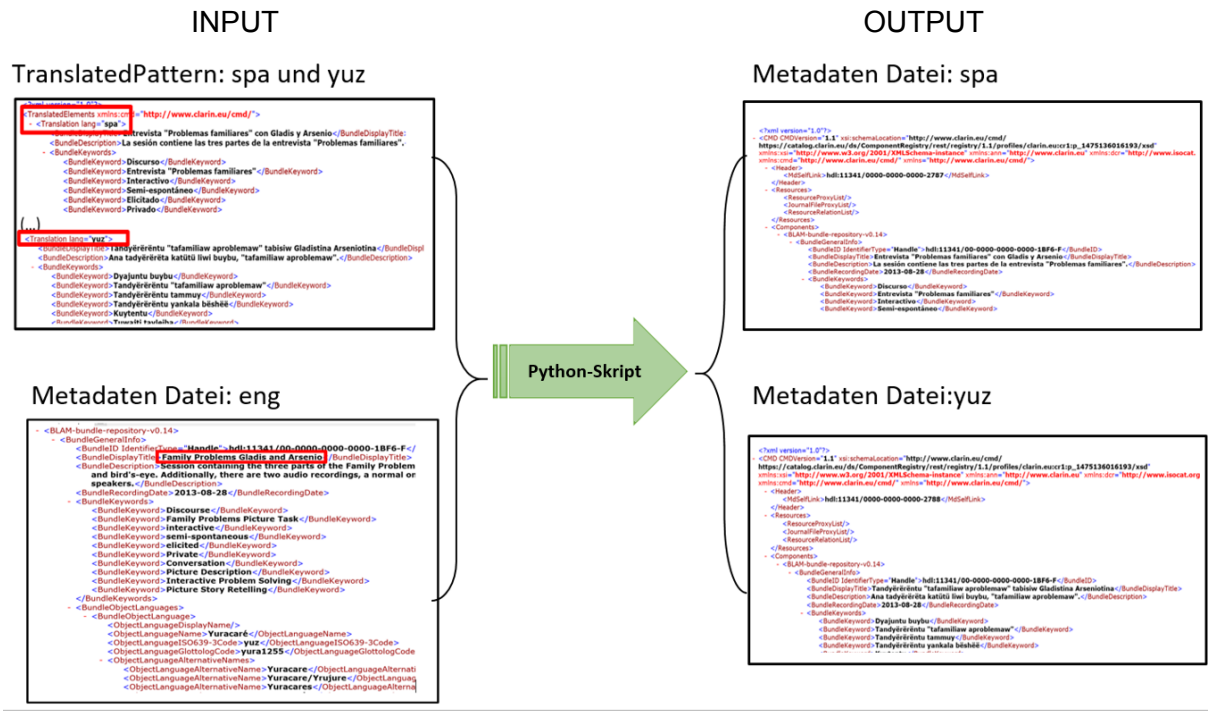


Abbildung 4: Zusammenführen der Übersetzungen und englischen Vorlagen

Letztlich erhält man pro englischer Metadaten-Datei eines Bundles, jeweils die Übersetzungsdateien: Im Yurakare liegen somit 21 Metadateien vor, da es bei sieben Bundles jeweils eine englische, eine spanische und eine Yurakare Metadaten-Datei gibt.

Solution: field “AdditionalMetadata”

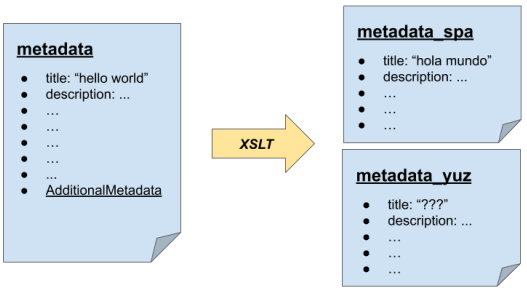


Abbildung 5: Feld “AdditionalMetadata”

Die Pipeline und das Python-Skript können auf Ordner Ebene für die Bundles einer Collection ausgeführt werden. Das Python-Skript ***mergeblamengandtranslation.py*** ist auf den Anwendungsfall Yurakare besonders zugeschnitten und bei einem anderen Testcase müsste das Skript weiter modifiziert werden. In den Yurakare Metadaten gibt es beispielsweise immer nur ein <*BundleObjectLanguage*> unter <*BundleObjectLanguages*>. Hier bezieht sich das Skript zur Zeit lediglich auf das eine Element.

2. Metadaten an das LAC übergeben

Die fertigen Dateien können nun an das LAC übergeben werden. Dies betrifft nicht nur die mehrsprachigen Dateien, sondern auch die englische Ausgangsdatei, da diese ebenfalls verändert wurde. Die Zuständigen der Metadatenkuration überprüfen die Dateien erneut und generieren neue PIDs und ergänzen im englischen Ausgangsdokument den Tag <*additional Metadata*>, in dem das spanische und Yurakare Duplikat festgehalten wird.

```
<BundleAdditionalMetadataFile>
  <FileName>family_problems_gladis_and_arsenio_spa.xml</FileName>
  <FilePID>hdl:11341/0000-0000-0000-2787</FilePID>
  <MimeType>xml</MimeType>
  <IsMetadataFor>hdl:11341/00-0000-0000-0000-1BF6-F</IsMetadataFor>
</BundleAdditionalMetadataFile>
<BundleAdditionalMetadataFile>
  <FileName>family_problems_gladis_and_arsenio_yuz.xml</FileName>
  <FilePID>hdl:11341/0000-0000-0000-2788</FilePID>
  <MimeType>xml</MimeType>
  <IsMetadataFor>hdl:11341/00-0000-0000-0000-1BF6-F</IsMetadataFor>
</BundleAdditionalMetadataFile>
```

Abbildung 6: XML-Tag “BundleAdditionalMetadataFile”

Die fertigen Dateien werden dann auf den Server geladen und in der OCFL-Struktur als Resources behandelt (s.u). Das RRZK hat bereits die Query-API angepasst und es kann im

LAC mit entsprechenden spanischen und Yurakare Begriffen gesucht werden. Die mehrsprachigen Dateien werden im Top-Level als *BundleTranslation* gekennzeichnet.

Diese Art der Transformation der Metadaten bezieht sich auf Daten, die bereits im LAC liegen. Dies kann bei dem Wunsch nach Mehrsprachigkeit auch weiterhin so gehandhabt werden. Da das LAC langfristig aber ein Formular zur Eingabe der Metadaten vor dem Hochladen anbieten möchte, wäre auch eine zusätzliche Spalte im Formular denkbar, in der die Mehrsprachigkeit direkt eingetragen werden kann.

Der nächste Schritt für das eLAC wird es sein, die Auflistung der Audio und Videofiles in Bezug zu den mehrsprachigen Daten zu setzen. Dann wird auch eine Suchanfrage mit dem spanischen Begriff “problemas familiares” die Audio- und Videodateien des LACs zu “family problems” ausgeben. Außerdem soll im Frontend noch das Feature der Mehrsprachigkeit implementiert werden. Die Grundlage für beides ist zur Zeit schon gegeben.

Technische Dokumentation

LAC

eLAC basiert auf der Infrastruktur des LAC, dessen REST APIs die Suche nach und die Darstellung von Dateien ermöglichen.
Die in eLAC verwendeten APIs sind:

- Query API (Durchsuchen des Index: Volltextsuche)
- Object API (Zusammenhänge und Abhängigkeiten der Dokumente)
- Media API (Wiedergabe von Mediendateien, bspw. Thumbnails oder Streams)

Das LAC bietet noch weitere APIs an, diese werden aktuell (Stand Mitte März) jedoch nicht in eLAC verwendet. Weitere Informationen finden sich in der API-Dokumentation des LAC: https://ka3.uni-koeln.de/apidoc/index (09.03.2020). Die Struktur der Metadaten wird durch das “Basic Language Archive Metadata” (BLAM) Format erfasst. Dieses eignet sich besonders für die Dokumentation linguistischer Daten in Repositorien. Die Datenorganisation des LACs richtet sich nach der “Oxford Common File Layout” Spezifikation (OCFL) zur nachhaltigen Strukturierung von Datensätzen. Im Folgenden ist ein

Beispieldatensatz aus dem LAC abgebildet und es ist sichtbar, wie die hierarchische Ordnerstruktur die Daten in *Collections* und *Bundles* einordnet.

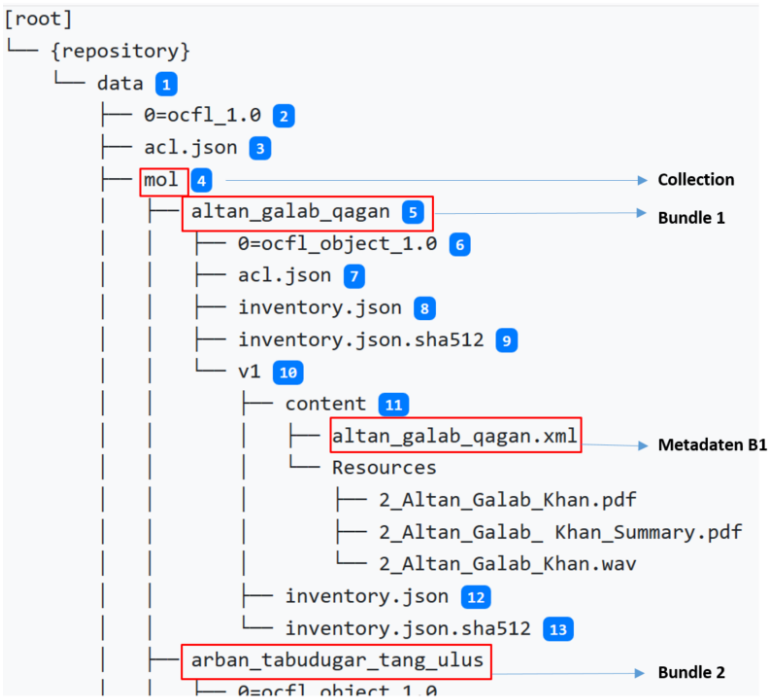


Abbildung 7: Struktur des LAC gemäß OCFL (Quelle: https://ka3.uni-koeln.de/doc/fileSystemStructure)

Jede Collection (hier am Beispiel “mol”) kann aus einem oder mehreren Bundles bestehen (bspw. eine Session). Die Bundle zugehörigen Metadatendateien (eng) liegen unterhalb des Ordners *content* eines jeweiligen Bundles. Die eigentlichen Mediendateien liegen unter dem Ordner *Resources*. Die, das eLAC betreffende Collection “Yurakare”, ist ebenfalls so aufgebaut. Unter der Ebene *content* darf lediglich eine Metadatendatei liegen, weshalb die im eLAC-Projekt erstellten mehrsprachigen Metadaten als Resources behandelt werden. Dadurch wird die OCFL-Spezifikation beibehalten und eine eindeutige Hierarchie etabliert, welche die englische Datei als Metadatendatei und die multilingualen Dateien als Resources kennzeichnet.

eLAC

Die App ist responsiv und für mobile Endgeräte optimiert. Eine voll funktionsfähige Ansicht via Computer beziehungsweise Laptop sollte problemlos möglich sein.
eLAC ist in Angular 9 implementiert und verwendet das darin enthaltene Material Design. Eine öffentlich zugängliche Instanz der Seite existiert noch nicht.

Mehrsprachige Metadaten

Zur Bereitstellung mehrsprachiger Metadaten wurde für bereits bestehende Bundles eine XPROC-Pipeline (unter “Arbeitsabläufe” genauer beschrieben) entwickelt, die per XSLT aus den englischen Metadaten Vorlagen zur Übersetzung erstellt. Diese Vorlagen können dann (nachdem sie übersetzt wurden) mit einem Python-Skript zu modifizierten Kopien der englischen Metadaten transformiert werden. So entstehen XML-Dateien, die von den verwaltenden Personen mit einem entsprechenden Editor bearbeitet werden können. Diese zusätzlichen Metadaten können dann im englischsprachigen Original als “additionalMetadata” angehängt beziehungsweise verknüpft werden. Nachdem die Daten auf den Server geladen wurden, ist eine Suche nach den Bundles über bspw. spanische Begriffe möglich.

Dies gilt für die Metadaten, die bereits im LAC archiviert sind und im BLAM-Format vorliegen. Sollten neue Collections ins LAC überführt werden, können die mehrsprachigen Metadaten entweder rückwirkend erstellt werden oder direkt während der Erstellung der BLAMS bspw. im ExcelSheet/Formular notiert werden.

Zugriff auf REST APIs

Die REST APIs des LAC ermöglichen ein simples Abrufen der hinterlegten Daten über HTTP Requests (JSON-Response). Die so erhaltenen Daten werden in eLAC über Angular-Components in im Browser darstellbares HTML umgewandelt.
Die Abfrage der Metadaten zu einem Video erfolgt beispielsweise zunächst über die Query API, um über den eingegebenen Suchbegriff an die gewünschten Ergebnisse zu gelangen. Die von der API erhaltenen Daten (JSON) werden nach den handles der Bundles durchsucht und diese an die Object API zurückgeschickt. Die Object API übermittelt strukturelle Informationen zu den Daten im LAC. Diese Informationen zu den Bundles enthalten den Wert *parentOf*, der aus einer Liste aus handles besteht, die auf die Mediendateien verweisen. Wieder werden strukturelle Informationen (wie Titel und

contentType) durch die Object API abgefragt, nur diesmal mit den handles der *children* bzw. der Mediendateien. Abschließend folgt die Abfrage an die Media API mit den handles der children. Hier können die Medien entweder direkt gestreamt oder Thumbnails generiert werden.

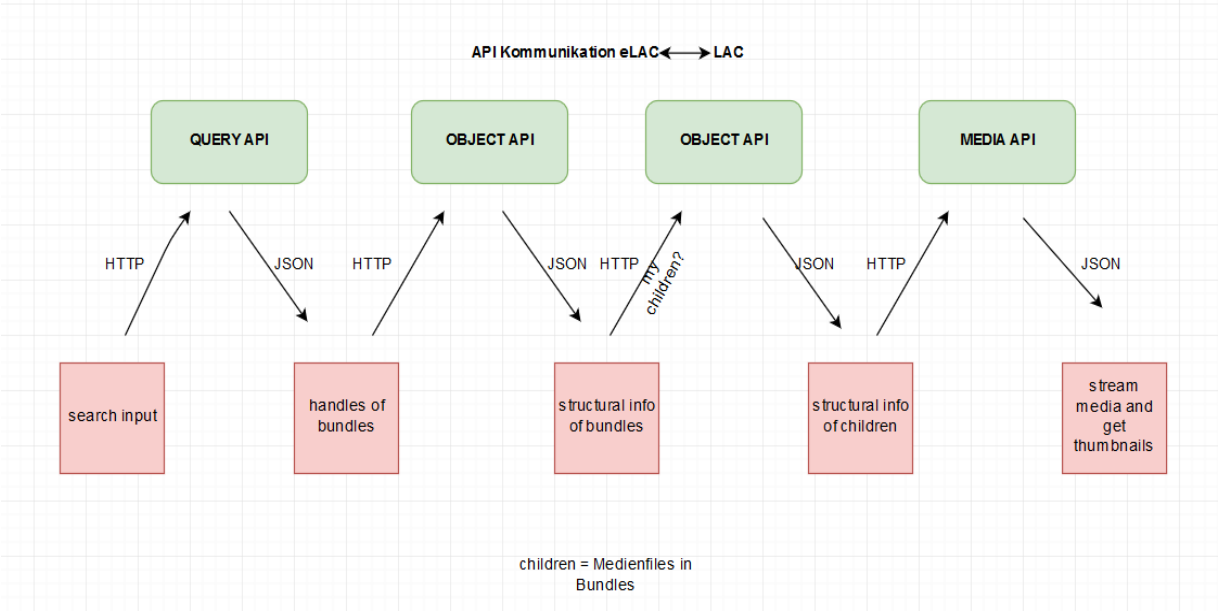


Abbildung 8: API Kommunikation eLAC und LAC

Implementierung in Angular

Das eLAC wurde in Angular implementiert, um eine spätere Einbindung ins LAC-Frontend (Angular) zu gewährleisten. Weiter bietet Angular gerade für mobile Endgeräte ein dynamisches und responsives Framework. Die Organisation in Components ermöglicht austauschbare und nachhaltige Funktionen. Für das eLAC wurden sechs Components generiert: für die Startseite die Component *elac-main*, für die Suche die Component *elac-search,* für die Ergebnislisten *elac-collection und elac-search-hits* und die Player-Components *elac-audio* und *elac-video.*

Bis auf *elac-search* (in *header*) sind alle Components über das Routing-Module eingebunden und miteinander verlinkt. Dies ist in der *app-component.ts* ersichtlich, in der die grundlegende Struktur und Organisation der Components festgelegt wird. Der Header und der Footer werden immer angezeigt und in *main* <*div*>, ist das Routing-Module angegeben, welches zwischen den verschiedenen Components verlinkt und die Ansicht der Components umschaltet. Als Default-Anzeige ist die *elac-main* Component festgelegt, die die Liste an durchsuchbaren Collections des LAC anzeigt (s. Abbildung 9: Ebene 1):

eLAC-Startseite (Ebene 1):

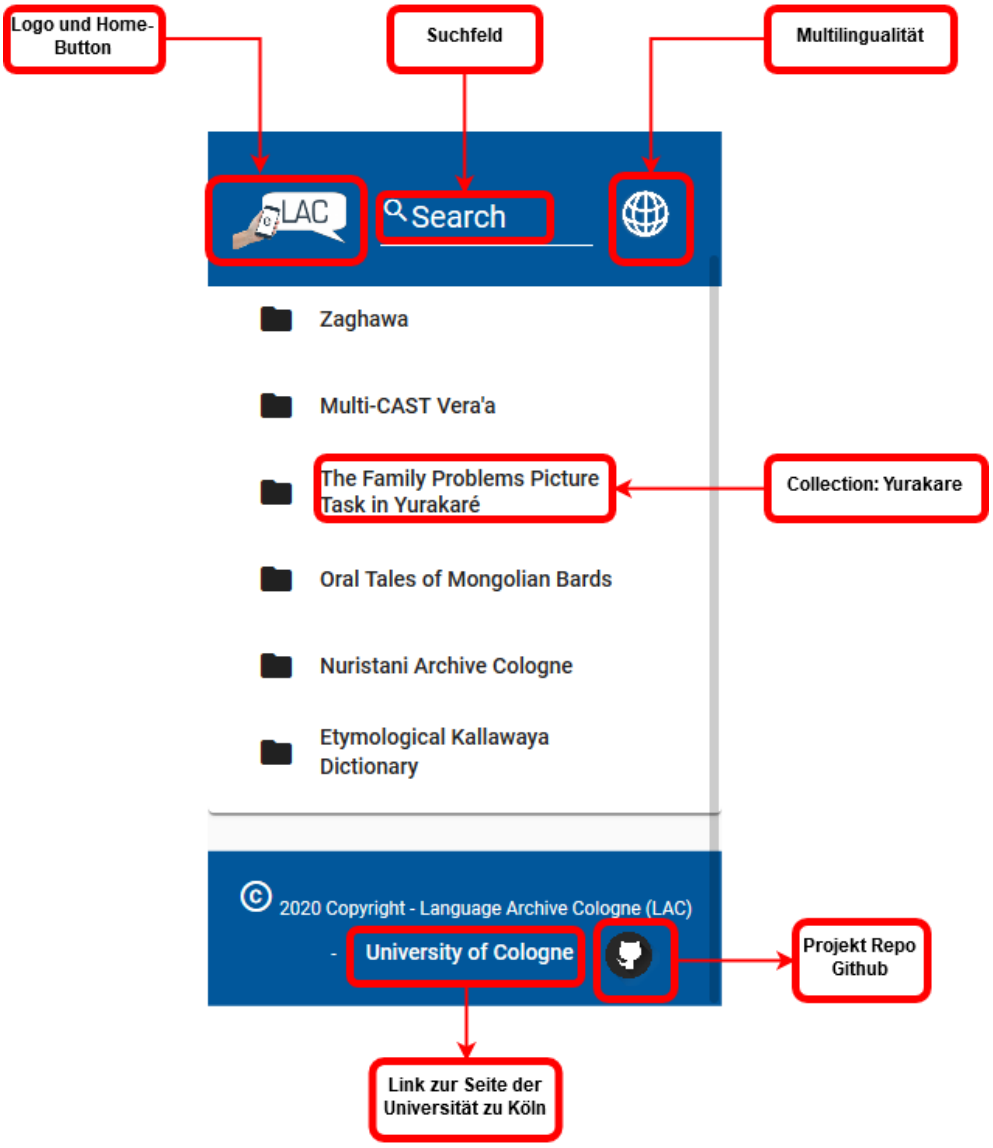


Abbildung 9: eLAC, Startseite mit Erklärungen

Routing-Module:

Alle Pfade, die zu den jeweiligen Components führen, sind in app-module.ts notiert. Diese Pfade können als Links ins HTML einer Component eingebunden und aufgerufen werden. Durch die Benennung der Links können zudem Daten von der den Link enthaltenen Component an die Aufzurufende mitgegeben werden, wie s.u. zu erkennen. Dies ist allerdings auf sehr geringe Datenmengen begrenzt, da die Pfade möglichst eindeutig und kurz gehalten werden sollen.

```
const routes: Routes = [
 { path: '', component: ElacMainComponent },
 { path: 'collection/:name/:prefix/:id', component: ElacCollectionComponent },
 { path: 's', component: ElacSearchHitsComponent },
 { path: 'videoplayer/:prefix/:id/:contentType/:suffix', component: ElacVideoPlayerComponent },
 { path: 'audioplayer/:prefix/:id/:contentType/:suffix', component: ElacAudioPlayerComponent },
 { path: '', redirectTo: '/', pathMatch: 'full' },
];
```

Abbildung 10: Pfade des Routing-Module (Screenshot)

Die eigentliche Datenweitergabe zwischen Components erfolgt über den *ElacDataService*.

Shared Services und Models:

Dies sind Funktionen und Klassen, die von allen Components gleichermaßen genutzt werden können. Andersrum können bestimmte Services auch Daten von Components entgegennehmen und an andere Components weiterreichen.

a. *ElacApiService*: Der API-Service ist für die Abfragen an die APIs des LACs zuständig. Da bei der Query- und Object-API größere strukturelle Informationen zurückerhalten werden, werden diese der Klasse *ElacApiEntity* zugeordnet, wodurch die weitere Verarbeitung der Daten erleichtert wird.
   Dies gilt nicht für die Media-API, da hierüber die Mediendateien gestreamt werden und die Daten nicht weiterverarbeitet werden müssen.
b. *ElacDataService*:
   Die aus den API-Anfragen erhaltenen Daten sollen von einer Component an die nächste weitergegeben werden bspw. elac-collection an elac-audio, damit das ausgewählte Audiofile auch abgespielt wird.

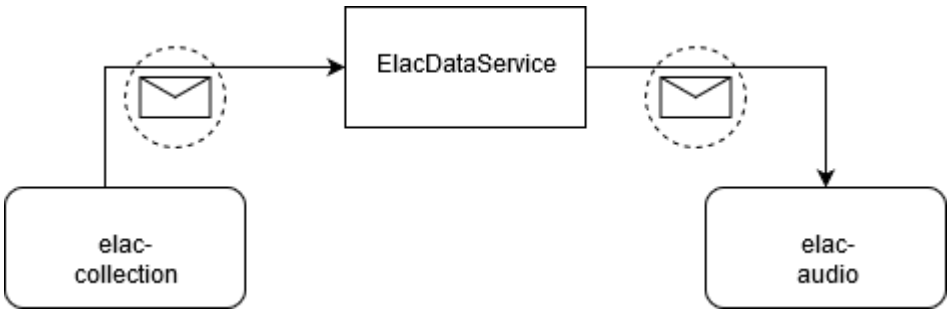


Abbildung 11: Datenfluss zwischen Components

Der ElacDataService kann Daten empfangen und weitergeben.

Kommunikation aller Instanzen im eLAC

Im Folgenden wird erläutert, wie das Zusammenspiel der Components und ihre Verknüpfungen durch Routing und Services im eLAC funktionieren. Das eLAC besteht letztlich aus drei Ansichten oder Ebenen. Die erste Ebene ist die Startseite (mit der elac-search Component immer im header), auf der zweiten Ebene werden die Ergebnisse an Mediendateien aufgelistet und die dritte Ebene umfasst die Audioplayer- und Videoplayer-Components.

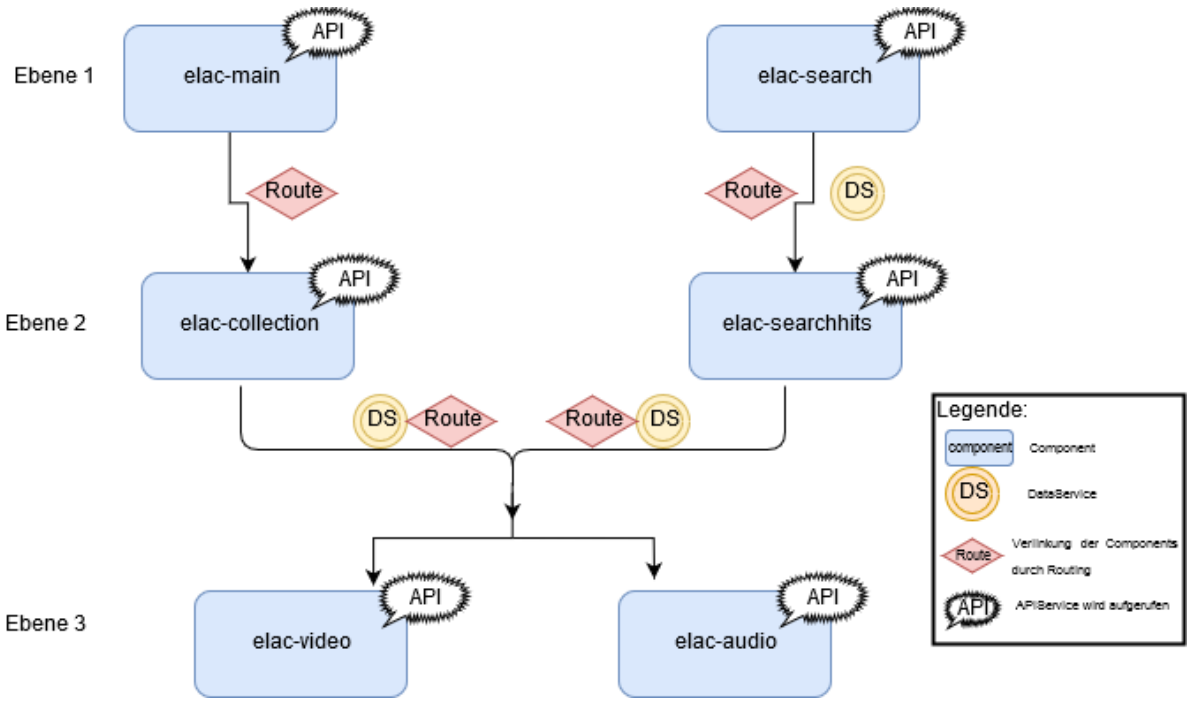


Abbildung 12: Kommunikationsabläufe im eLAC

Ebene 1

Wie zu Beginn des Kapitel ausführlich erläutert umfasst die erste Ebene die Startseite des eLACs. Hier wird die *elac-main* Component als *default* über das Routing-Modul angezeigt, die die Auswahl an Collections im LAC darstellt. implementiert. Die Informationen zu diesen Collections werden über den *ElacApiService* abgefragt und die Auswahl einer Collection über den Routing-Link an die Component *elac-collection* weitergegeben.

Die *elac-search* Component ist im Header eingebunden und der Input-Wert wird ebenfalls über den *ElacApiService* abgefragt. Hier werden die Daten aber nicht über das Routing an die *elac-search-hits* Component weitergeben, sondern über den *ElacDataService*. Das Routing von *elac-search* zu *elac-search-hits* wird bei Eingabe eines Wertes in das Textfeld ausgelöst (s. Abbildung 12).

elac-main und elac-search

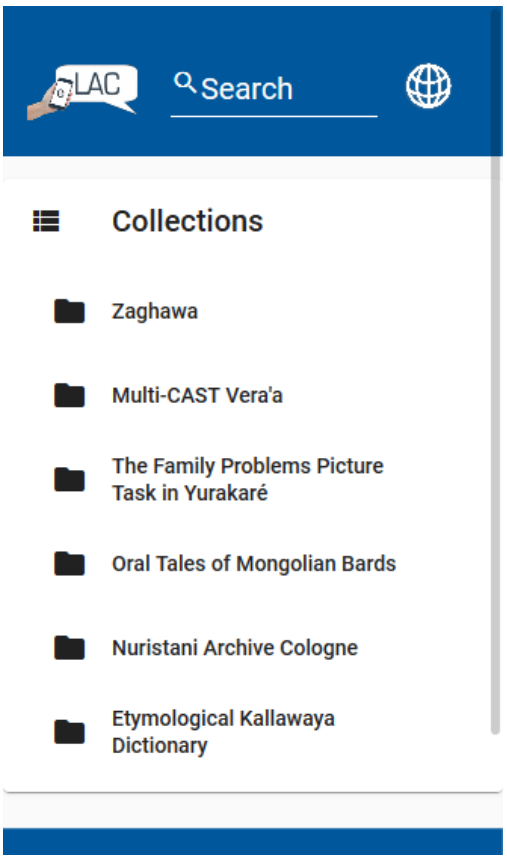


Abbildung 13: eLAC, Ebene 1

Ebene 2

Die Mediendateien können über zwei Wege aufgelistet werden. Entweder wird eine Collection in *elac-main* ausgewählt, welche dann über einen Routing-Link zu der elac-collection führt oder der eingegebene Suchbegriff (*elac-search*) wird über den *ElacDataService* an die Component *elac-search-hits* weitergegeben und löst den Routing-Link zu *elac-search-hits* aus. In beiden Fällen erhält man eine Auflistung der Mediendateien, welche ihre Daten durch den API-Service abfragen und Thumbnails generieren.

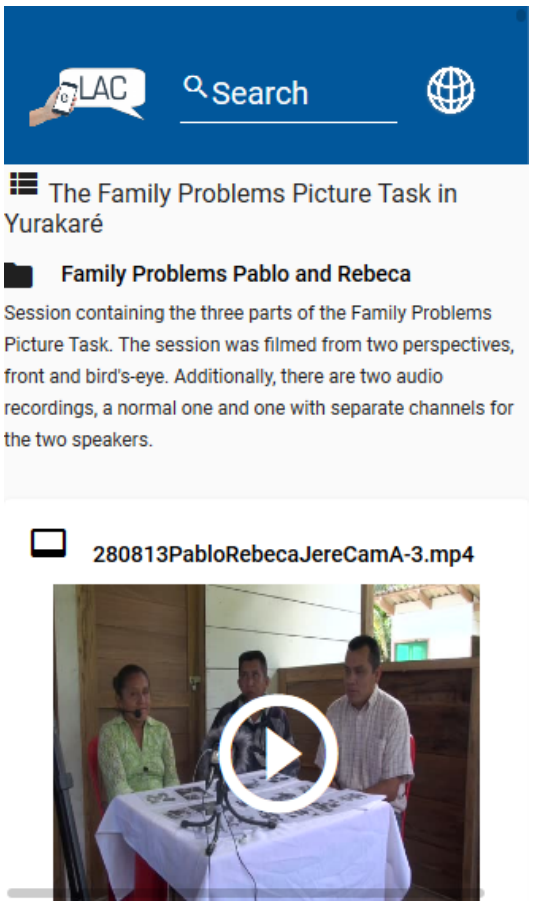
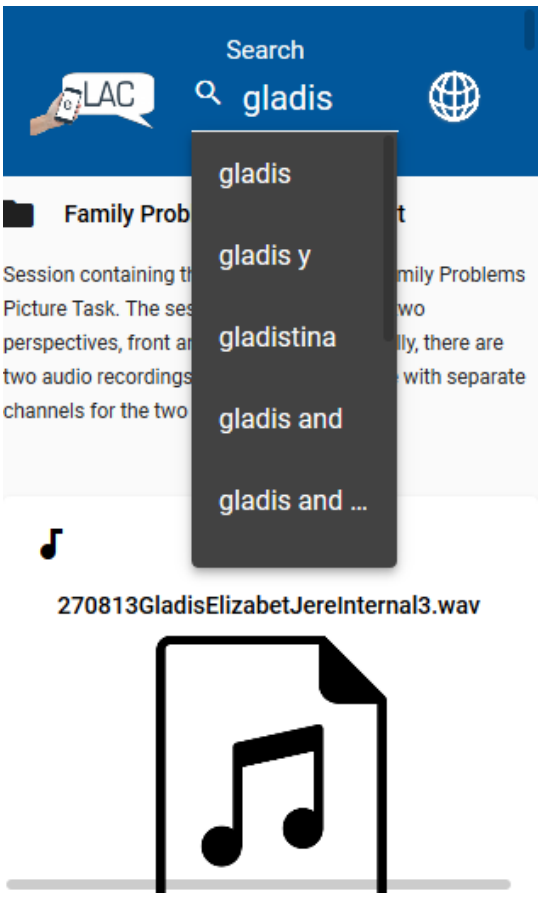
elac-collection	elac-search und elac-search-hits
	

Abbildung 14 und 15: eLAC, Ebene 2

Ebene 3

Ebenso verhält es sich mit der dritten Ebene. Wird ein Element aus der Liste (*elac-collection* oder *elac-search-hits*) ausgewählt, leitet der Routing-Link entweder zum Audioplayer (*elac-audio*) oder zum Videoplayer (*elac-video*). Die Daten des ausgewählten Mediendateien werden über den *ElacDataService* an die neue Component weitergegeben. Die Player-Components schicken dann eine Anfrage an den API-Service (Media-API), um die Mediendateien zu streamen und Thumbnails zu generieren.

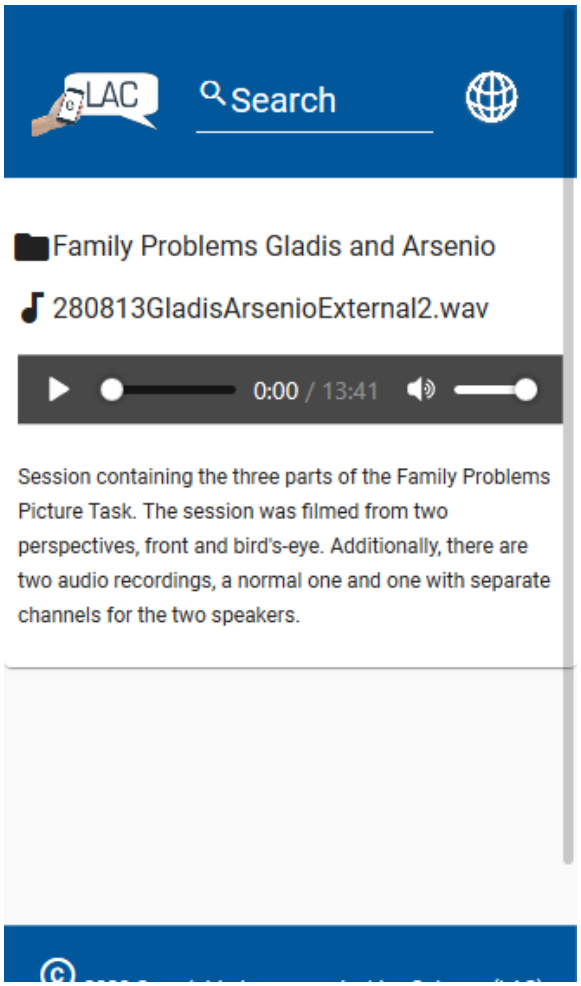
Audio-Player: *elac-audio*	Video-Player: *elac-video*
	

Abbildung 16 und 17: eLAC, Ebene 3

Usability Test

Im Rahmen einer Übung zur Usability von Software wurde ein für eLAC erstelltes Mockup mit der Anfang 2020 öffentlichen Webpräsenz des LAC verglichen.

Abbildung 18: Balsamiq-Mockup für eLAC

Das Ergebnis des Vergleichs muss vor den Hintergrund gestellt werden, dass das Mockup nur sehr beschränkte Funktionalitäten bot und das Finden der gesuchten Seiten dadurch erleichtert wurde. Zudem wurde der Test der Webpräsenz des LAC gezielt von Mobilgeräten aus durchgeführt.

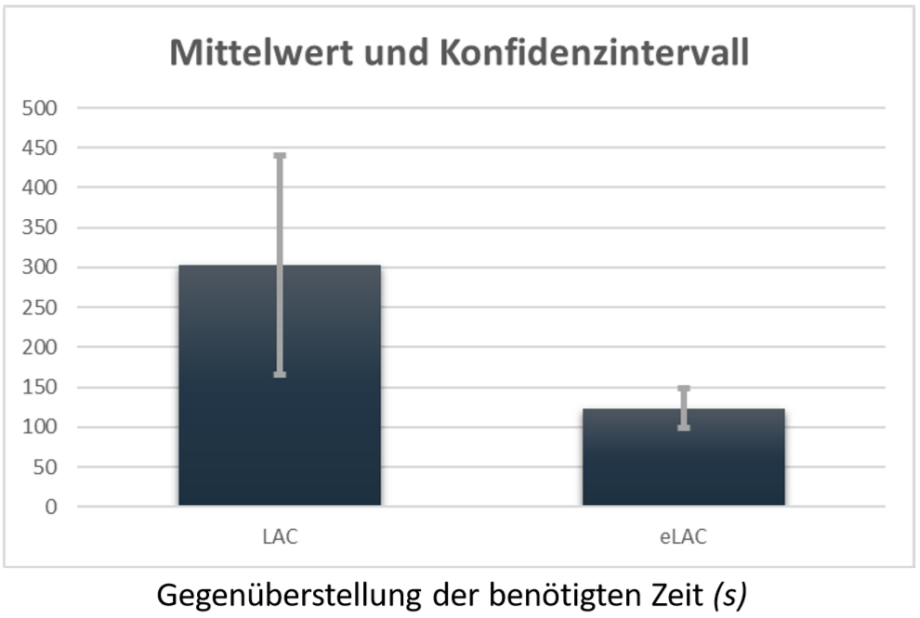


Abbildung 19: Auswertung Usability Test (Quelle: Poster Usability DHCon2020)

Bei der Gegenüberstellung der zwei Gruppen mit jeweils sechs Probanden zeigten die für die gestellten Aufgaben benötigten Zeiten eindeutig einen Vorteil durch das eLAC-Mockups.

Beteiligte Personen und Zusammenarbeit

Projekt-Team

Nils Geißler & Anne Gerlach, Universität zu Köln

Zusammenarbeit mit

Regionales Rechenzentrum Köln (RRZK)

Data Center for the Humanities (DCH)

Institut für Linguistik (IfL)

Quellenverzeichnis

- Data Center for the Humanities (DCH). “Language Archive Cologne”. https://dch.phil-fak.uni-koeln.de/bestaende/repositorien/language-archive-cologne (17.03.2020).
- KA^3. “File System Structure”. https://ka3.uni-koeln.de/doc/fileSystemStructure (17.03.2020).
- KA^3. “REST API Documentation”. https://ka3.uni-koeln.de/apidoc (17.03.2020).

Abbildungsverzeichnis