INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelor's Thesis

Design and Evaluation of a Sign Language Dictionary leveraging Wikidata's Semantics

Eva Schambach

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelor's Thesis

Design and Evaluation of a Sign Language Dictionary leveraging Wikidata's Semantics

Eva Schambach

Task assigned by:	Prof. Dr. Dieter Kranzlmüller
Supervised by:	Maximilian Höb Dr. Constanze Schmaling Maximilian Kristen
Deadline:	April 19, 2024

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Munich, March 4, 2024

[illegible]

(Signature of the candidate)

Abstract

The work explores the utilization of the Wikidata platform and its standard tools as the backend for an online dictionary for Hausa Sign Language. In this context, there exists an official website providing an overview of Hausa Sign Language, alongside two separate applications for Android and iOS, each with its own database. This situation results in the aim of creating a common database to be implemented into a cross-platform application. Dart/Flutter was chosen as the cross-platform programming language. For the common database, Wikidata: Lexicographical Data was selected as an online dictionary. Due to its structure and regulations for sign language representation, the frontend had to be adapted to efficiently display the content of each entry. To retrieve specific entry components, the SPARQL system was employed for querying across diverse data sources, assisting in addressing certain regulations imposed by Wikidata. The interface for executing SPARQL queries was implemented using the FutureBuilder widget, which allows for the asynchronous construction of user interfaces while awaiting the completion of Future objects. Another crucial aspect was overcoming the lack of a local database to implement features like marking entries as favorites. The SharedPreferences package was employed to persistently store these entries. Although the implementation is currently in the prototype stage, the feasibility of the work could be demonstrated. While some requirements could not be fully verified, the overall potential of implementing Wikidata: Lexicographical was showcased, highlighting its importance over the current state with two separate applications. Additionally, identified disadvantages were complemented with proposed solutions, such as local data caching to mitigate internet dependency when using the application.

Contents

1 Introduction

1.1 Motivation

In an era characterized by the centrality of communication within interpersonal relationships, envisioning a world devoid of it seems implausible. Delving into the realm of communication, one naturally considers both verbal and non-verbal forms, such as gestures and facial expressions. Another form of communication is sign language, which is of particular significance for deaf people. The quantification of the deaf populace remains elusive, since disabilities are not reportable. Since it turns out to be quite complicated to differ *hard of hearing* people from *deaf*, the number of disabled people varies. [Kem14]. Nevertheless worldwide, the proportion of people with hearing loss is estimated at about 20%, according to WHO statistics. [Fou]
Mainly the communication of deaf people runs through sign language, which forms a large community characterized by its unique cultural identity. According to the UN one of around 300 documented Sign Languages around the world, is Hausa Sign Language, abbreviated HSL, which is the language of the Deaf community in Northern Nigeria. [UN] Conversely, the hearing community, speaks Hausa, the predominant language spoken across West Africa. [EB]
To meet the growing demand for online sign language dictionaries, a robust database is essential for easy accessibility. Implementing a structured and collaborative platform as the backend for a sign language dictionary could be a promising opportunity to improve access and safeguard sign language knowledge. By embracing this approach, the documentation and sharing of sign languages worlwide can be improved. Moreover, utilizing a shared platform has the potential to promote inclusivity and foster deeper understanding among diverse communities.

1.2 Research questions

The main purpose of this work is to prove the feasibility of implementing WD:LD, abbreviation of Wikidata:Lexicographical Data, as the backend for a prototype version of an online dictionary in form of an application.
A few years ago, an online dictionary comprising approximately 700 lexemes was developed and presented on a website and in an application, available for both Android and iOS, each with its own data set. However, they are outdated, so a new online dictionary is asked, cross-platform based and with a common data set.
When working on this purpose, following research questions are raised:
(i) The main task includes issues on how to create a cross-platform application to replace the current available applications and therefore make it easier to access the application no matter which operating system is used. The question of the usability of Flutter/Dart as a programming language for the implementation (of the announced demand) will be answered.

(ii) Another concern will be how the entries in the online dictionary are created, in other words which lexemes (for example HSL lexeme, Hausa lexeme, etc.) are implemented, what their syntax looks like and what further explanations, such as illustrations, can supplement an entry.
(iii) With regard to the editability of the entries and their use by queries, it is clarified how, a SPARQL query is created in Wikidata so that the entries are output in the application. To answer these questions, Wikidata's underlying project WD:LD is presented as the planned backend system, as well as SPARQL as the query system.
(iv) The last question that arises when using WD:LD concerns the possibility of implementing characters in the current Wikidata ontology system, and how to get around this if it is not possible. The definition of an ontology system and why it is important will be given.

2 Basics and related work

The following chapter provides insights into the basics, covering Hausa Sign Language and its structure, an introduction to some formal definitions and an overview to Wikidata. Based upon that, WD:LD is introduced as the underlying platform for the upcoming application. To contextualize related work, boundaries to other online sign dictionaries are defined.
Another aspect of this chapter involves presenting three sign language notations, which constitute an approach for formally writing signs.
Lastly, currently existing applications and the key distinctions from the forthcoming paper are presented.

2.1 Hausa Sign Language

The following chapter bases on the article named Hausa Sign Language by Constanze Schmaling [Sch15].
Hausa is the primary language used in Northern Nigeria and South Niger, and serves as a lingua franca in West Africa. However, alongside spoken Hausa, another language exists in this region known as HSL, which is primarily used by the Deaf community.
HSL has distinct rules and grammar that differentiate it from spoken Hausa. Deaf individuals acquire HSL through interactions with their parents, neighbors, or other members of the Deaf community. Interestingly, even members of the hearing community have the ability to communicate in HSL on a basic level, since HSL is recognized as a distinct language in its own right. Consequently, the broader community places a high value on communication with the Deaf through HSL.
HSL has different forms due to the diverse cultural backgrounds of the Deaf community, which can include factors such as age and place of origin.

Structure

The structure of HSL is categorized into manual parameters, encompassing handshape, orientation, location, and movement, and non-manual parameters.
Signs are executed with either one hand (usually the dominant hand) or both hands, in static or dynamic fashion. Dynamic handshapes involve finger play, such as the repeated movement of the fingers, and handshape change, signifying the transition from one shape to another.
Orientation denotes how to hold the palm and the extended fingers. It is further divided into static and dynamic groups. "Dynamic orientation involves some type of wrist movement." [Sch15, p. 370], which is more common than, for example, bending.
Location is the parameter that indicates where to place the hand: in space, on or next to the face/head, or, most commonly, in front of the chest with body contact.
The indicator movement, specifically path movement, pertains to changing the location of a handshape, the parameter described lastly, which "[...] can occur on the body or face, in

space, and from the body into space and vice versa.” [Sch15, p. 371] The type (for example straight, circle, etc.), the direction (for example horizontally, vertically, etc.), the manner and the number of movements can serve as specifications, as well as the combination of these four parameters. For instance, the manner of movement can be straight or circular, described by factors such as size, speed, and tension.

Examples of signs

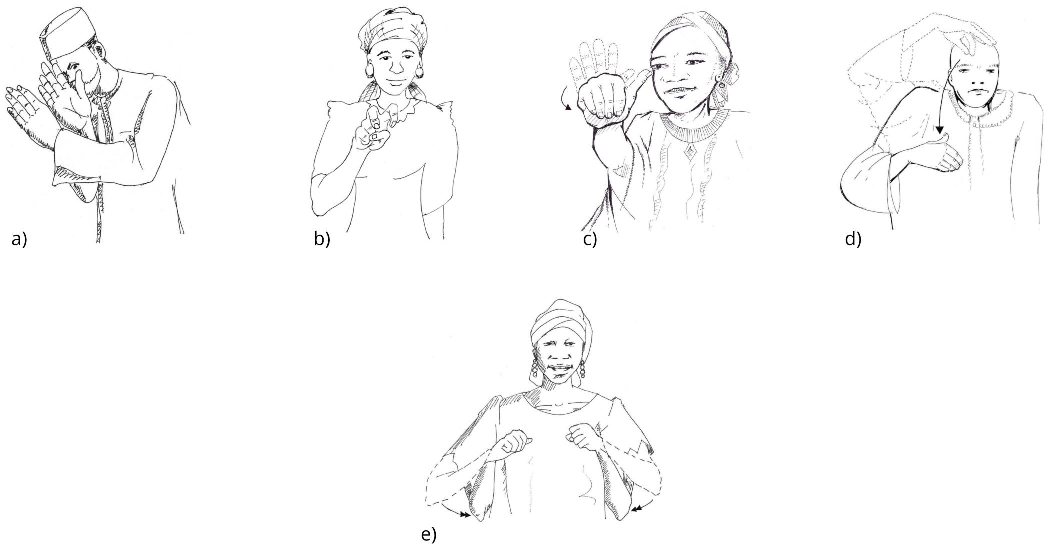


Figure 2.1: Examples for signs involving static handshapes (a, b) and dynamic handshapes (c, d); example for a sign including a non-manual parameter (e) [Sch]

In the following section, the previously illustrated examples of signs containing static and dynamic handshapes, obtained from the current Hausa Sign Language website, will be elucidated [Sch].
”Hide” for the Hausa lexeme B’UYA is depicted in Figure 2.1 (a), serving as an example involving a static handshape. In this case, both hands maintain a fixed position, with their orientation towards each other, while being held in front or beside the face and rotating in opposite directions. It is worth noting that shapes involving both hands, as shown here, typically represent a single morpheme.
Another example is MATA for ”wife” illustrated in Figure 2.1 (b). It exhibits a sign by holding the hand more freely in space, where the index and middle fingers are bent, and the thumb touches the ring finger and little finger.
As previously mentioned, dynamic handshapes represent the other fundamental category of handshapes, distinct from the movement parameter, presented in Chapter 2.1.
An example of a sign involving a dynamic, one-handed handshape is KIRA for ”call” as seen in Figure 2.1 (c). It depicts a straightened hand that is bent at a certain angle.

The sign in Figure 2.1 (d), DA KAINA, which translates to "myself" shows another dynamic handshape, accompanied by some movement. Initially, the hand is straightened, with the thumb touching the index finger, then the thumb separates from the index finger, and the hand is slightly bent forward. Here, the parameter of movement plays a role, as the handshape begins above the head and is later lowered down.
Non-manual parameters, such as head or shoulder movements, can also sometimes differentiate the expression of a word or phrase. For example, MURNA for "pleasure" is depicted by making a fist with both hands and bending the arms upwards, as shown in Figure 2.1 (e).

2.2 Formal definitions

To establish a foundational understanding of key terminology that will be referenced throughout the thesis, it is essential to provide clear definitions.

2.2.1 Semantic web

The Semantic Web is a framework for structuring and organizing data on the internet to make it more meaningful and interpretable by machines. It relies on standards like RDF, introduced in Chapter 2.2.3, to represent data and relationships between entities in a machine-readable format. This enables applications to understand and process information in a more intelligent and automated manner, leading to improved search, integration, and analysis of data across different sources. [HKRS07]

2.2.2 Ontology system

To address the question about whether and how characters can be represented in WD:LD, exploring an ontology system, rooted in ancient Greek philosophy, seems essential. (1.2 (ii)) Ontology aims to explain the existence and classification of objects, aligning with the study of being. [ONK] [EB]
Derived into computer science, the abstract of a chapter in a research named "Human Performance and Instructional Technology" declares, Ontology is "a data model that represents a set of concepts within a domain and the relationship between these concepts."[ZD09], describing the basic level of objects and classes. Moreover, they shed light on (common) attributes of objects and the relationships between objects. An example of an ontology system is WD:LD.

2.2.3 Resource Description Framework

An important tool for WD:LD is the format of semantic triples, supported by RDF (Resource Description Framework), as outlined in W3C recommendations. [PS08] RDF serves as a language for representing and exchanging data between different applications, ensuring that data retains its meaning. Therefore the URL is used to identify information alongside other metadata, such as the author's name or the title.
The structure of a RDF entry is similar to the structure of a Wikidata/WD:LD entry comprising a subject, a predicate, and an object. This standardized format offers flexibility, allowing for extensions and customization, which is particularly beneficial for Wikidata as a

database that is constantly being filled with new data or revising its existing entries. The other key advantage is its capability to link entities to similar or related entities within Wikidata itself or across other knowledge databases.

2.3 Wikidata

Wikidata is one of the main resources the bachelor thesis requires, as already mentioned in the introduction.
According to the website itself it is a "[...]free and open knowledge base that can be read and edited by both humans and machines." [Wik]
Wikidata is connected with various platforms, serving as the central repository for structured data originating from sources such as Wikipedia and Wikimedia etc. Mainly the platform consists of items, completed by details like descriptions and alias, a property and a value. The item, following the subject in RDF's scheme, is identified with a Q, followed by a sequential number, for example *Q34228* for *Sign Language*, whereas the property, alias the predicate in RDF, is identified by a P, suffixed by a sequential number as well, for example *P31* for *instance of.* The value of the property, representing the object in RDF, is again an item and in consequence identified by a Q and a sequential number.
Wikidata is divided into name spaces, selected by items, properties, etc., thus care should always be taken to ensure that the desired labels are set correctly when searching. Another specification of the searching query is the advanced search which sets more searching limits.

2.3.1 Wikidata:Lexicographical Data

Now the underlying database for the application, WD:LD, is being examined.
It is part of Wikidata and stores "[...] words, phrases and sentences in many languages, described in many languages." [Wik], comparable to an online dictionary but with more functions, such as statements and other advantages facilitated by RDF, as introduced in Chapter 2.2.3.
In the following the content and style of a WD:LD entry is presented.
The information (words, phrases, etc.) is primarily stored in Lexemes, which are then subdivided into parts, such as Lemma, Forms and Senses. Figure 2.2 gives an overview of the structure of Lexemes.
The main structure of an entry of WD:LD follows the structure from Wikidata itself, thus a lexeme is then again identified by a sequential number and a prefixed L, for example *L14111* for the word *sign.* The boxes coloured in blue indicate that the respective instance is described only once, while all other boxes can be described multiple times. The section Lemma describes the standard and human readable form or dictionary form of the lexeme, for example *sign.* The Lexical Category indicates the grammar, referencing the respective item Q, thus for the announced example it is a *noun.* The Language is chosen by Wikidata and again referenced to the item Q; in the case of the chosen example *sign* it is evidently *English.* Statements can include several properties P, such as *usage example* or *described at URL* leading to a an external website. In the example of *sign*, WD:LD includes the property *homograph lexeme*, setting *sign* as the default value.
Forms is divided into three parts: the Representation as the string version of the given form, the Grammatical features which give information about tenses and more and the Statements

describing relations to other items, such as pronunciation audios. In the example of *sign*, only some different tenses are given, such as *present participle* and the value *signing*.
The last class Senses describing different meanings of the lexeme is again subdivided into two parts: the Gloss is about the use of natural language for describing the lexeme, while the class of Statements again combines the Sense to other Senses, in the form of translations or synonyms.

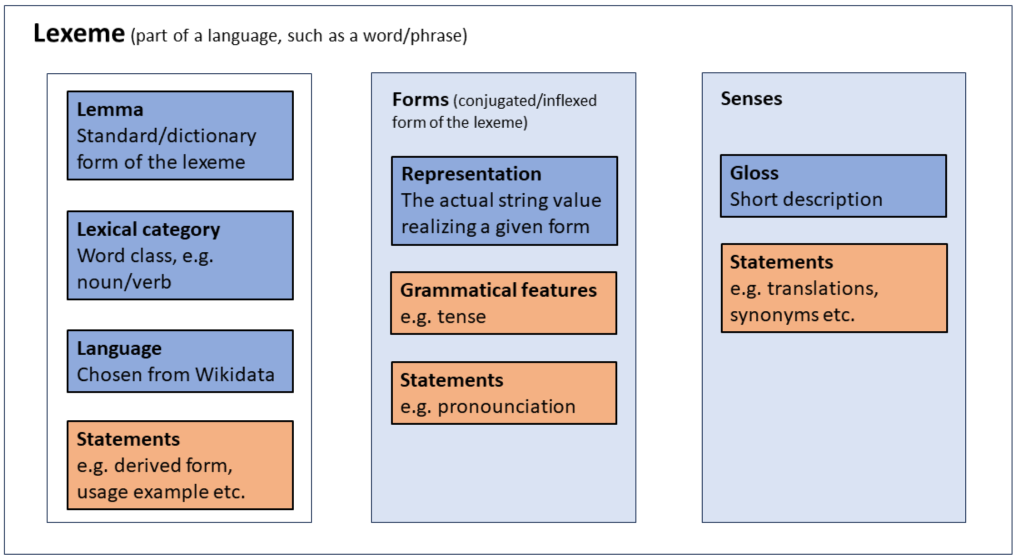

Lexeme (part of a language, such as a word/phrase)
Lemma
Standard/dictionary form of the lexeme
Lexical category
Word class, e.g. noun/verb
Language
Chosen from Wikidata
Statements
e.g. derived form, usage example etc.
Forms (conjugated/inflexed form of the lexeme)
Representation
The actual string value realizing a given form
Grammatical features
e.g. tense
Statements
e.g. pronounciation
Senses
Gloss
Short description
Statements
e.g. translations, synonyms etc.

Figure 2.2: Wikidata's documentation of the overview of the structure of lexemes [Wik]

2.3.2 Distinction of Wikidata from other online dictionaries

In order to understand the benefits of WD:LD towards other online dictionaries, the general structure of online sign dictionaries and two use cases are illuminated. Besides the sign itself, dictionaries often include the citation form of the sign and sometimes grammatical descriptions, for example *directional verb*. [Han04]
As an example of a sign dictionary, the SignDict platform will be introduced. [Sig17] Similar to Wikidata, it relies on the support of others and operates as an open platform, resulting in a wide range of alternative signs. To upload a new sign, a webcam is necessary to record the sign. Additionally the entries include the Sutton SignWriting, a system of writing sign languages that will be discussed later in the thesis.
Since the platform is only designed for German Sign Language and does not consider other sign languages, it is unsuitable for HSL.
Besides there is the platform named LinguaLibre providing tools for recording and sharing audios together with features for editing and annotating them. [Fra] The goal of this open-source project, created by Wikimédia France, is to create a diverse archive of minority languages, which are then made available for linguistic research and educational purposes.

The described approach is beneficial for sign languages overall; however, it may not be suitable for the intended use case, as it requires a written representation of the signs.

2.4 Notation systems for Sign Languages

One well-known approach of creating own lexemes for the signs is the system of writing sign languages, where signs are represented in writing. J. Hopkins provided an introduction to the basics and the functionality of this system. [Hop08]
For a more precise understanding, the Latin alphabet, along with the Greek alphabet, is often mentioned as exemplary for a comparison. These alphabets are used to write languages like English, German, Greek, and others, even though sign writing systems do not function in the same way as traditional alphabets. Furthermore it is essential to emphasize that these sign writing systems do not serve as a practical tool for the deaf community, as alphabets do for reading and writing as part of communication. In order to meet the needs of the deaf community, pictures and videos of signs are far more useful. Instead, the sign writing systems were originally developed to establish a formal means of representing and documenting sign languages in the form of a notation system. Meanwhile even that purpose is highly discussed, since it requires additional training and expertise in practise.
Over the years, several of these systems for sign language notation have been developed, all with the aim of establishing a standard using symbols and in consequence using them for research purposes. However these systems are not as widely adopted as, for instance, the Latin alphabet. The few systems finding practical application are discussed below.

2.4.1 Sutton SignWriting System

The official website of the Sutton SignWriting System, abbreviated to SSW, provides an overview of this notation. [Sut] SSW is a widely recognized standard for describing sign languages internationally. Unlike other systems, SSW focuses on representing the physical form of signs rather than analyzing their semantics. This approach has been embraced in education, research, and platforms like Wikidata.
The Brazilian Sign Language, for instance, utilizes SSW to represent its entries, leading to the current usage of SSW as the standardized form for sign representation in WD:LD. Even though in practise, the system can not be rendered without a speicific script included in one's own Wikidata account. [Wik]
SSW symbols are categorized into eight groups, covering aspects like handshapes, movements, and facial expressions, with a total of 652 symbols available for sign representation.
A closer examination is given to the way SignWriting is used for the expression "don't know" in Figure 2.3 (a). The circle is part of Category 4, which covers the position and movement of the head, indicating a side-to-side shake as shown by the arrows above. The half-painted pen touching the circle denotes a flat hand position while moving outward, as indicated by the overlaid arrows.
For a clearer understanding of the sign, the corresponding "don't know" American Sign Language (ASL) sign is presented in Figure 2.3 c).

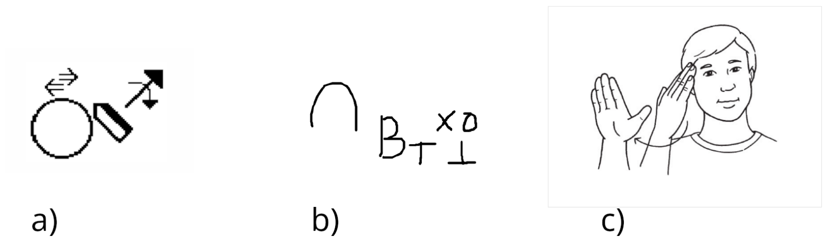


Figure 2.3: Illustration of the Sutton SignWriting (a), the Stokoe Notation (b) and the sign itself (c) "don't know"
[Mar00]

2.4.2 Stokoe Notation System

The article named "Towards a Unicode encoding for Stokoe Notation" discusses the Stokoe Notation System, primarily used in research for descriptive purposes in various publications. [Pri12] Created by William C. Stokoe, it includes ASCII and unique symbol sets, initially developed for ASL. [Sto65] Although adapted for other sign languages like British Sign Language, it has not been integrated into Wikidata. Initially, basic parameters such as location (known as tab), handshape (known as dez), and movement (known as sig) were ordered to describe signs, with later modifications by researchers to suit specific needs.
Stokoe describes, among others, the ASL example "don't know", which has been redrawn in Figure 2.3 (b). The handshape symbols consist mostly of Latin based characters, such as the B in "don't know", which is visualized by a flat hand. The location symbol (tab) is placed before the handshape symbol to indicate where the handshape should be placed; here $\bigcap$ is indicative of placement near the upper face. All symbols after the B represent combined movement symbols. While the subscript T indicates a movement towards the signer, X indicates a touch movement and $\perp$ indicates a movement away from the signer.

2.4.3 HamNoSys

The Hamburg Notation System (HamNoSys), introduced in 1987 by Thomas Hanke, provides a versatile notation system for all sign languages, but faces compatibility issues with Wikidata due to its Unicode encoding in the Private Use area. [Han04]
With 200 symbols for handshape, orientation, movement, and location, HamNoSys emphasizes intuitive glyphs over conventional alphabet or math symbols. Each sign description includes posture details and sequential or parallel actions, with an added parameter for two-hand signs' symmetry.
The system is best explained with an example analyzed for the meaning of the symbols. Figure 2.4 pertains to "woman" whose sign has already been depicted and explained in the

HSL chapter 2.1.
In Figure 2.4 the first symbol shows the index and middle fingers raised in a V shape, combined with the thumb held horizontally and the fingers bent forward, which can be checked in Figure 2.5 (a), the basic set for handshapes. Additionally, the symbols for hand orientation and palm orientation, as shown in 2.5 (a-1) and (a-2), respectively, include the second and third symbols from the example in Figure 2.4. These legends present varying viewpoints: the hand orientation symbols encompass three perspectives (signer's view, bird's view, and right-side view) regarding the direction of the index hand, whereas the palm orientation symbols encompass two perspectives relating to the index hand, such as *palm away from the body*. More specifically, in the chosen example, the second symbol indicates that the hand is facing forward and upward at the same time, while the palm is facing straight forward. The location symbol (see Figure 2.5 (b)) is also indicated by two components: while the first indicates the position within the frontal plane, the second determines the distance of the hand from the body. Unless specified otherwise, the sign is represented at a natural distance from the upper body, indicated by Ø, as in the given example of the word MATA.
Lastly, there are parameters for movements, often involving combinations of path movements, specified as targeted or relative movements. In Figure 2.5 c), some possible movements are displayed (from the signer's view, bird's view, and view from the right). Since the example MATA does not contain any action, this part will not be discussed further.

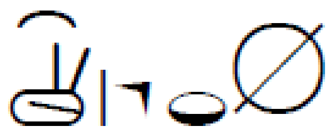


Figure 2.4: Illustration of the HamNoSys applied for the sign of MATA [Sch15]

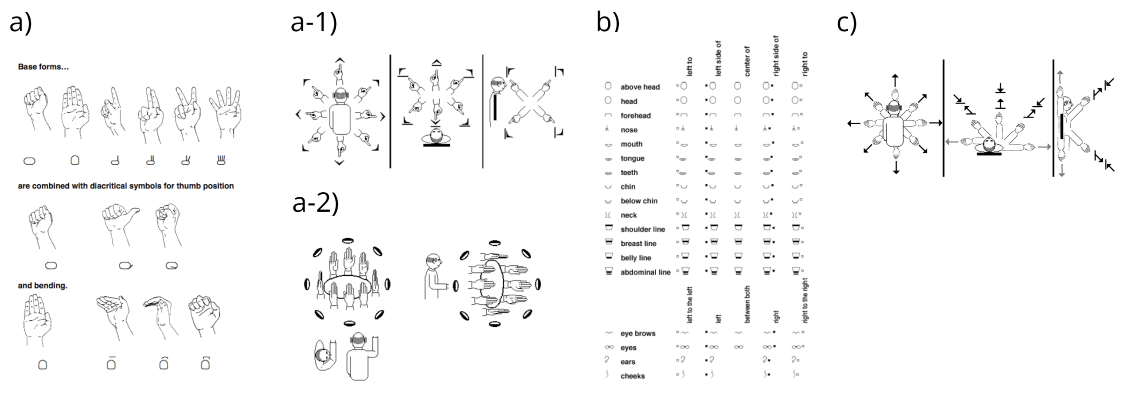
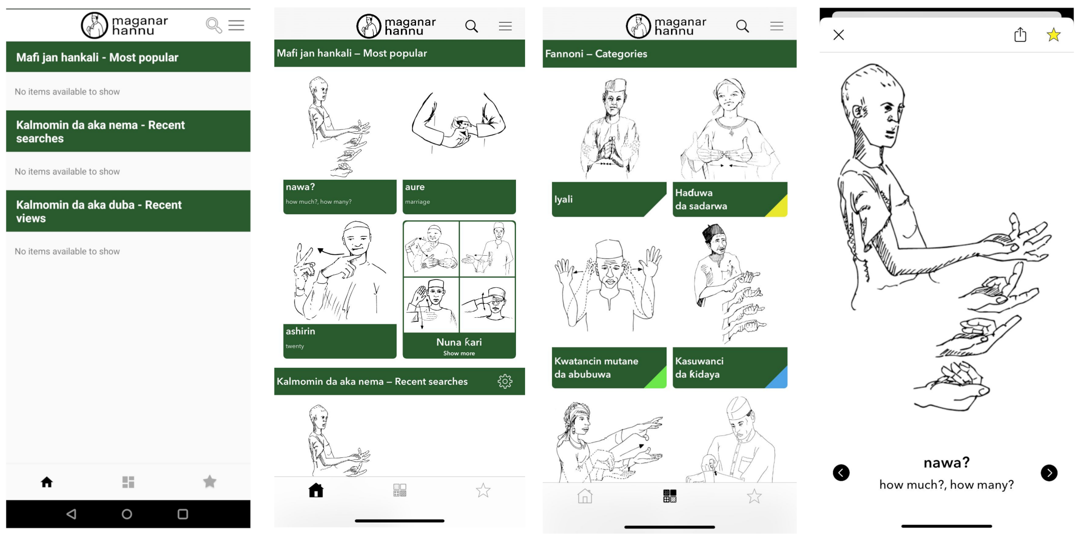


Figure 2.5: Handshape symbols in a), hand orientation symbols in a-1), palm orientation symbols in a-2), location symbols in b), movement symbols in c) [Han04]

2.5 Current applications

In 2018, two bachelor theses were published focusing on the development of an application for a HSL dictionary. [Bol18] One thesis worked on the iOS version for the front-end, while the other tackled the Android version, each with its own database. The application bears the name MAGANAR HANNU (Hausa), which translates to "language of the hands" in English.
The design of both applications is predominantly identical, as evidenced by the subsequent screenshots in Figure 2.6, taken from two different phones: one running on Android (a)) and the other on the iOS platform (b), c), d)). Screenshots a) and b) depict the initial screens upon launching the application. Navigation across different screens is facilitated by symbols for different applications: a house for the Home Screen (Start Screen), a grid-based layout for Categories, a star for Favourites, a magnifying glass for Search, and a menu bar for Settings. The Start Screens include sections like Most Popular, Recent Searches, and Recent Views, depicted as unfilled in a) and filled in b). The backend dictionary consists of approximately 700 expressions organized into thematic categories such as Family or Everyday activities, offering expression suggestions aligned with the selected thematic context, as shown in Figure 2.6 c).
The screenshot in d) displays the content and style of the exemplary entry for NAWA? (in English "how much?"). This entry can either be suggested within one of the categories or sections or selected via the search function, enabling users to query specific expressions or words. The entry features the Hausa lexeme, English translations, and a corresponding picture illustrating the sign. Users can click the star icon in the right corner to add the entry to their favorites, and it also offers an option for export.


maganar hannu
Mafi jan hankali - Most popular
No items available to show
Kalmomin da aka nema - Recent searches
No items available to show
Kalmomin da aka duba - Recent views
No items available to show
maganar hannu
Mafi jan hankali – Most popular
nawa?
how much?, how many?
aure
marriage
ashirin
twenty
Nuna ƙari
Show more
Kalmomin da aka nema – Recent searches
maganar hannu
Fannoni – Categories
Iyali
Haduwa da sadarwa
Kwatancin mutane da abubuwa
Kasuwanci da kidaya
nawa?
how much?, how many?

a) Android Start Screen b) iOS Start Screen c) iOS Categories Screen d) iOS Entry Screen

Figure 2.6: Screenshots of the different screens in the current applications [Wik]

2.6 Distinction between the related work and the subsequent thesis

There are some significant differences between the Bachelor Thesis from 2018 and this Bachelor Thesis, such as compatibility. The following version will be compatible across all platforms, regardless of which version (iOS/Android) the user has installed. Therefore, Flutter is used as the main development kit, coded with the Dart programming language.
Another major difference is the underlying database. The author of the iOS application utilized MySQL for the backend and SQLite for the frontend. In contrast, this work employs a single shared database, namely WD:LD, to provide the data.
Since the thesis does not focus on the development of a new application but especially on the integration of Wikidata, this can only be a prototype version.

3 Design of the application

This chapter will be divided into the following components: the user interface, its non-functional and functional requirements and the underlying architecture that aligns with the user interface. Together, these components comprise the application's design. Certain parts will be adapted from the existing applications, as there is no need for extensive changes. Furthermore methods for the implementation, involving the programming language and the interface to the backend are presented.

3.1 Requirement Analysis

Before elaborating the user interface and defining the underlying architecture, requirements have to be set.
A differentiation is established between functional, primarily technically based, and non-functional requirements.

3.1.1 Non-functional requirements

Through the review of previous bachelor theses and an analysis of the resulting current applications, the following non-functional requirements have been defined.

a) **Navigation to different screens**:
Upon launching the application, the user is directed to the Start Screen, where he can access different screens by pressing the corresponding buttons.

b) **Views for the most popular entries, recent views and recent searches**:
Besides those buttons, the Start Screen should include a view for the most popular entries, as well as a view for the recent views and the recent searches.

For the individual screens, more requirements are needed:

c) **Selecting languages from the Settings Screen**:
The Settings Screen should provide an option for selecting the language used for the application's guidance, specifically English and Hausa.

d) **Adoption of the categories from the current application**:
The categories from the Categories Screen are taken over from the classification of the current applications, namely for example Family, Education, etc.

e) **Implementation of the functionality of the Favorite entries through an icon star**:
The Favorite entries section is populated only when users mark entries as their favorites by selecting the star icon.

f) **Navigation through a magnifying glass icon**:
Upon pressing the magnifying glass icon, the Search Screen should open, and as the user types, the application should provide suggestions for entries.

3.1.2 Requirement of an entry

One of the primary objectives is to establish a standardized framework for both the content and visual presentation of entries within WD:LD, as well as for their representation within the application. Therefore part of the non-functional requirements is the design of an entry which is later checked by showing an exemplary entry.

a) **Requirement of an easy adaption of an entry**:
It is required that a new entry can be easily created, as well as edited later on.

b) **Presentation forms for the sign, such as illustrations or videos**:
Several presentation forms for the sign itself shall be included. Therefore an illustration of how to perform the sign is required. Optionally, a video could provide a more detailed demonstration.

c) **Implementation of a gloss or a spoken translation of the lexeme**:
Furthermore a label is required which is namely a gloss, a representation form of the sign in a spoken language. [Man06].
Optionally a spoken translation of the lexemes can be added.

d) **Requirement of an HSL lexeme**:
Last but not least the sign requires its own lexeme, known as an HSL lexeme, which can be compiled using one of the presented sign writing systems in chapter 2.4.

Exemplary entry for a Hausa Sign lexeme in the application

Based on the non-functional requirements outlined in the previous chapter, the following idea of a design for displaying an entry within the application has been created, see Figure 3.1. In addition to the mentioned content, which includes the Hausa and English gloss, the sign illustration, the video, and the sign writing system HamNoSys, the user has two interaction options: they can navigate back to the previous page by pressing the arrow in the top left corner, and/or they can mark it as one of their favorite entries by selecting the star.

3.1.3 Functional requirements

The functional requirements primarily pertain to the backend and can be partially derived from the non-functional requirements, as the backend must ultimately be able to accept and process the requirements from the frontend.
Furthermore the following requirements have been established:

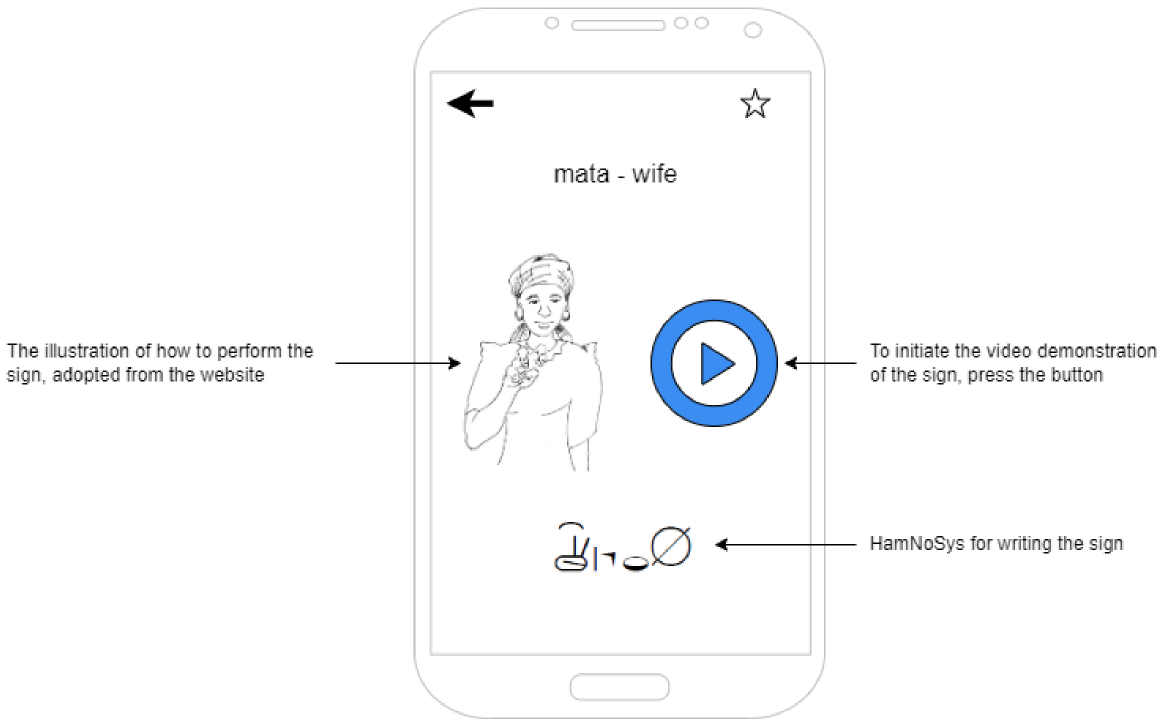

mata - wife
The illustration of how to perform the sign, adopted from the website
To initiate the video demonstration of the sign, press the button
HamNoSys for writing the sign

Figure 3.1: First draft of an entry in the frontend

a) **Usability across platforms**:
The new application must be usable across platforms, necessitating the use of a cross-platform programming language. This requirement is based on the idea of replacing the currently two existing applications, as briefly discussed in Chapter 2.5.

b) **Update through an URL link**:
Requirement 3.1.3 a) results in the requirement that the current applications can either be updated or replaced by displaying a note with a URL link to download the new application.

c) **Database efficiency**:
The backend must utilize a database that provides quick and easy functionality for updating the entries within the application.

d) **User-friendliness**:
The user-friendliness has to be maintained, means the interface must have a simple and intuitive operability. This can be measured by testing and gathering user feedback.

e) **Minimization of the memory consumption**:
In addition, it is important for the application to minimize memory usage, particularly since mobile phones have limited storage space, similar to other hardware. This consideration extends to both the programming code and the database containing signs and other related data.

f) **Reliability and stability of the application**:
Last but not least, the application needs to be reliable and stable in terms of accurate

screening and technical aspects, such as error handling, etc.

3.2 Architecture

The architecture of a system requires various perspectives. In the following, a detailed examination is conducted on the different models forming the foundation of a system's architecture.
Subsequently, the architecture of the backend, as well as the one of the frontend system, will be presented. Additionally, the process of data ingestion into Wikidata and its subsequent retrieval will be explained, focusing on the format of SPARQL queries, which will be introduced accordingly.

3.2.1 Models for the foundation of a system's architecture

For modelling the architecture of the application, first of all a detailed look, based on the online platform IP Insider is taken onto the two main different systems Peer-to-Peer and Client-Server architecture. [Lub20b] [Lub20a] The Peer-to-Peer architecture is organised in a decentralised way, that means the application is not or hardly dependant on the server. The peers, the instances of the communication, are directly and equally connected to each other and can either provide services or use services. They behave autonomously so that no additional instances are required to organise the system.
In contrast, the client-server architecture is centrally organised so that different requests can be made to the server or to the client. Server provide services that can be asked and used by one or more clients. The communication between these instances is controlled by protocols and is always initialised by the client.
The underlying backend, also known as the server, of the online dictionary is WD:LD; delivering the data for the application. In order to meet this requirement, the architecture must be a Client-Server architecture, illustrated in Figure 3.2.

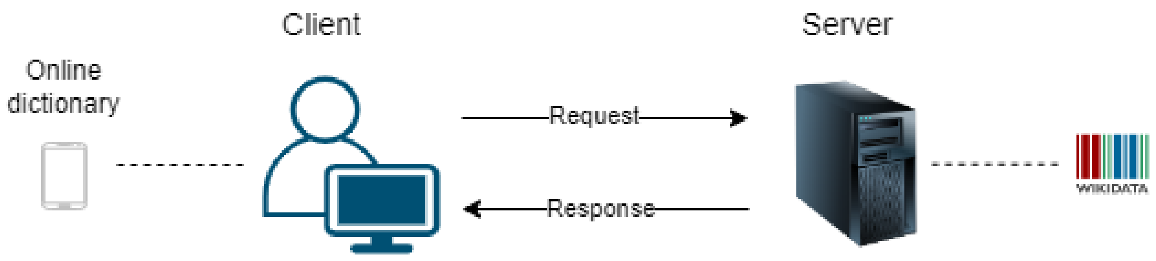


Figure 3.2: Server-Client Architecture depicted in a chart

3.2.2 Architecture of the backend

Since the backend system WD:LD, known as part of Wikidata, does not need to be developed and is fixed in its architectural model, it is only briefly highlighted.
Figure 3.3 displays a chart that provides an overview of the architecture's structure, based on a diagram published by Wikidata. [Wik]
Access to MediaWiki, the software on which Wikidata is built, and to other extensions, such

as Wikibase (a storage of editable information), is managed through varnish-cashing, the main purpose of which is load balancing. [Var20]
This so-called reverse proxy is necessary due to problems with the performance of a dynamic website, which occur with more users and higher complexity. While the server upstream of the reverse proxy processes a request, the reverse proxy caches it for renewed requests, which are then answered by loading it via the Varnish Cache.
The aforementioned load balancing function manages multiple concurrent accesses by treating them as threads that are processed in order. A fixed limit determines the number of active threads that can be processed simultaneously, while the remaining threads are transferred to a queue.
The Varnish Cache acts as a server between Wikidata and users, tools and other parties. Other services, such as Graphite as part of WMF Analytics for live monitoring, also interact with Wikidata but are not depicted in the diagram.
Last but not least, Wikidata is linked to storage technologies that facilitate data management, such as MySQL (more specifically MariaDB Service) for primary metadata. However, it's crucial to note that Wikidata itself is not a relational database.

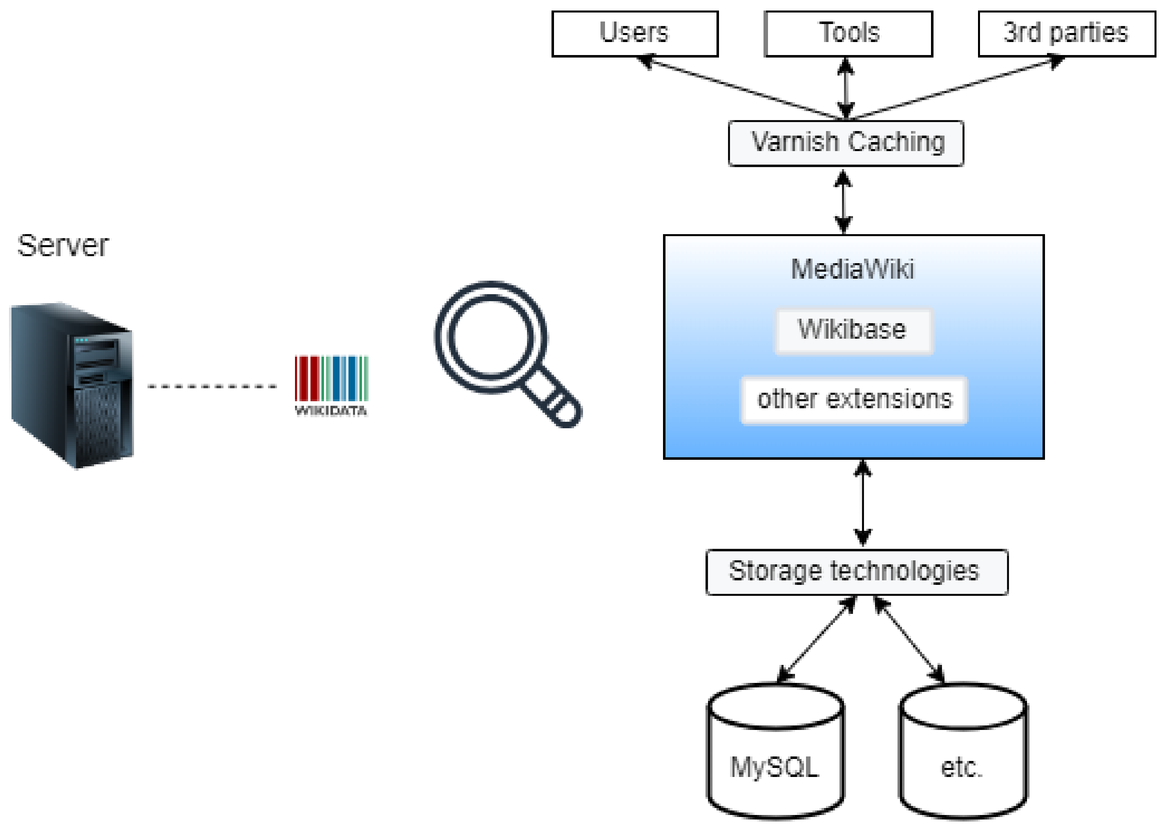


Figure 3.3: Diagram of Wikidata's architecture
[Wik]

3.2.3 Structure of WD:LD's data regarding the representation of signs

In order to create a standard of an entry for the Hausa lexemes, a detailed look should be taken at the current representation of signs in Wikidata.
Therefore, an overview of the currently existing sign languages is provided in Figure 3.4 based on a SPARQL query, see Chapter 3.2.5 for an introduction. [SPA] In total, 32 different sign languages are represented, which is a relatively low number compared to the documented 300 sign languages worldwide. [UN] The overview is organized based on the quantity of corresponding lexemes, and as evident, the largest dataset belongs to Brazilian Sign Language with a total of 456 lexemes, whereas HSL currently contains only five lexemes. This illustrates the underdevelopment of WD:LD in terms of entering data for sign languages.
Besides there are some extensions of existing Wikidata entries, such as *December* (Q126), where the property label in sign language is accompanied by a corresponding sign language, such as Catalonian sign language. However, since this resembles more of a translation and is still not directly included in WD:LD, it is not considered in this overview. [Wik] The goal should be to input unique lexemes for the signs to maintain the integrity and cultural significance of sign languages as independent linguistic entities.

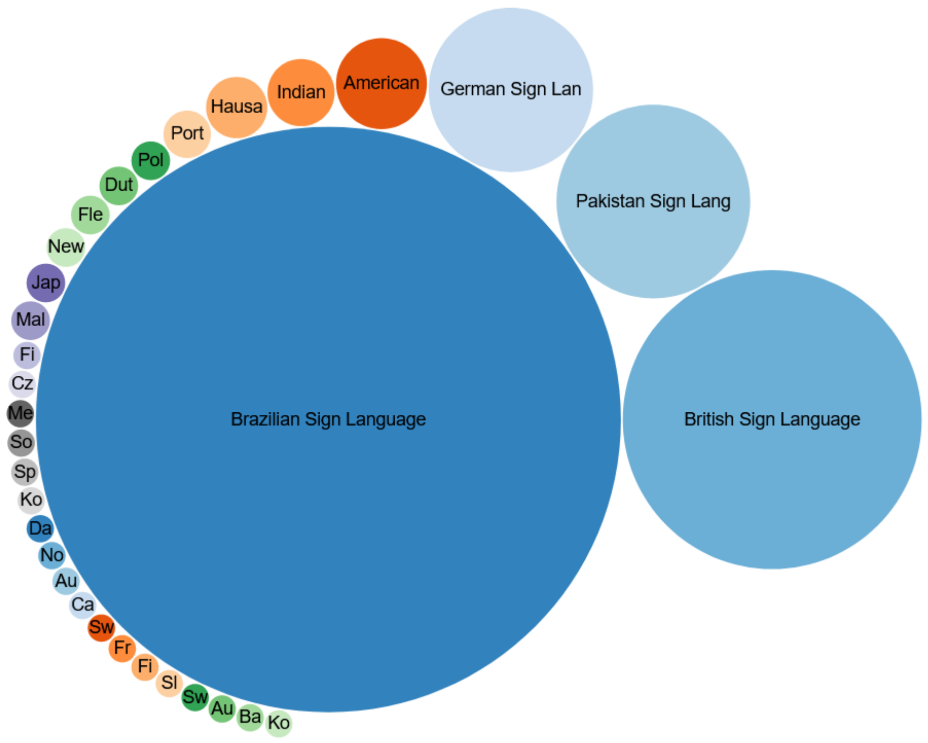


Figure 3.4: Distribution of the sign languages in WD:LD
[Wik]

The current practice within the Wikidata community involves using SSW as the standard method for declaring lexemes associated with sign languages. However, this approach poses challenges for adding new lexemes, since it requires knowledge of the corresponding alphabet. [Wik]
When analyzing the lexemes, their information is traditionally represented as strings of characters. However, they can also be depicted through visual media such as pictures or videos associated with a property.
There are two properties of significance: *label in sign language* (P2919) and *signed form* (P3969). The *label in sign language* property typically includes a media file, as exemplified by the Latin letter *Ö*, demonstrating the handshape of this letter in the manual alphabet of the German sign language. Additionally, other items, such as *word* (Q8171), also include short videos. Both variations can be seen in Figure 3.5.

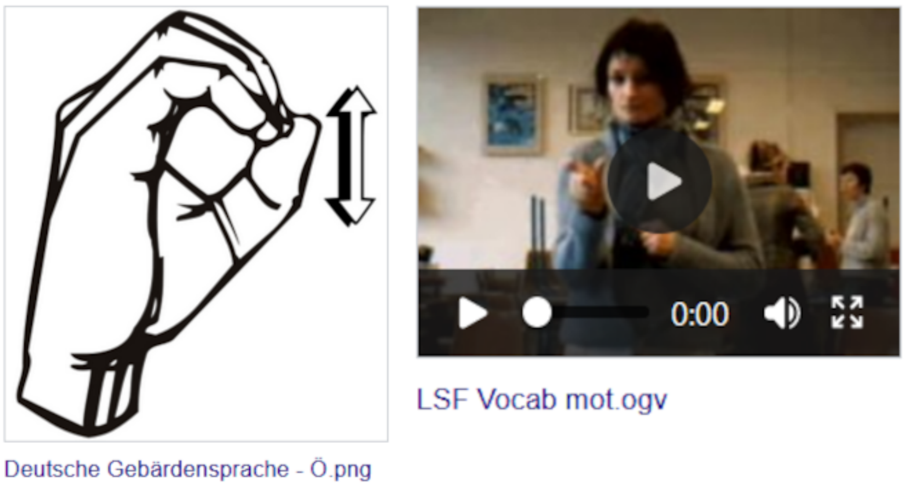
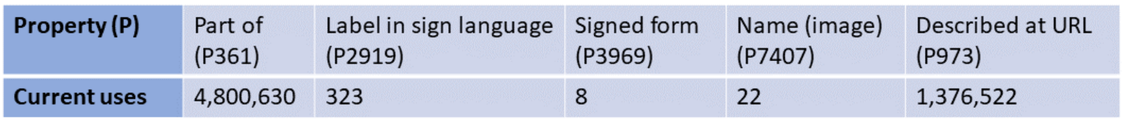


Figure 3.5: Variations for the property *label in sign languages* [Wik]

In comparison to other properties, such as *part of* (P361), both properties P2919 and P3969 are rarely and inconsistently used, as evidenced in Table 3.6. While the *part of* property is used 4,800,630 times, *label in sign language* is used only 323 times, often for items containing letters. The *signed form* property is applied only 8 times, which raises questions about its relevance. There is a discussion in the associated namespace property talk about whether it should be deleted. [Wik]
In addition, there are approaches to circumvent the direct implementation of signs, such as the use of the property (P7407) *name (image)* and the property (P973) *described at URL*. *Name (image)* serves as a useful option for including images when the presentation form is not supported by Wikidata, such as Unicode and is included 22 times in total. Conversely *described at URL* offers an approach for implementing signs in Wikidata by linking to websites, such as sign language dictionaries that display the signs. While P7407 is used only 22 times, P973 is quite familiar, with a total count of 1,376,522 uses.

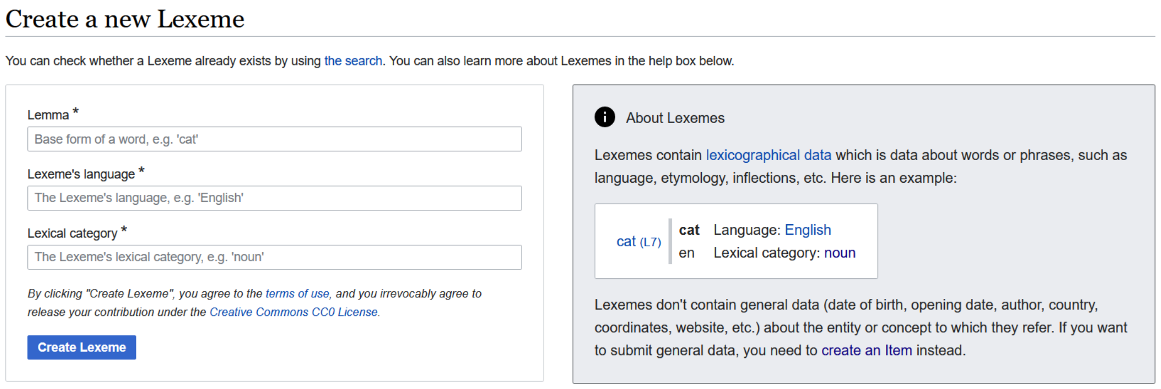


Property (P)	Part of (P361)	Label in sign language (P2919)	Signed form (P3969)	Name (image) (P7407)	Described at URL (P973)
Current uses	4,800,630	323	8	22	1,376,522

Figure 3.6: Current uses for selected properties
[Wik]

Data input

To enter a lexeme, an entry needs to be created, referring back to the initial inquiry into how data is integrated into WD:LD.
The screenshot (Figure 3.7) illustrates the process of creating a new lexeme, starting with entering the lemma's name, followed by specifying the language and the lexical category, such as *noun*. As HSL is not currently registered as a distinct language, *Hausa* is entered instead.



Create a new Lexeme

You can check whether a Lexeme already exists by using the search. You can also learn more about Lexemes in the help box below.

Lemma *
Base form of a word, e.g. 'cat'

Lexeme's language *
The Lexeme's language, e.g. 'English'

Lexical category *
The Lexeme's lexical category, e.g. 'noun'

By clicking "Create Lexeme", you agree to the terms of use, and you irrevocably agree to release your contribution under the Creative Commons CC0 License.

Create Lexeme

About Lexemes

Lexemes contain lexicographical data which is data about words or phrases, such as language, etymology, inflections, etc. Here is an example:

cat (L7)	cat	Language: English
	en	Lexical category: noun

Lexemes don't contain general data (date of birth, opening date, author, country, coordinates, website, etc.) about the entity or concept to which they refer. If you want to submit general data, you need to create an Item instead.

Figure 3.7: Creation of a new lexeme
[Wik]

3.2.4 Architecture of the Frontend

The architecture model can be demonstrated by the Model-View-Controller pattern, see Figure 3.8. [Buc09] Each component has specific responsibilities, briefly described as follows:

1. The Model stores data, primarily from the current state of the application, such as the version number. Additionally, the Model can interact with other modules, such as the database by allowing access and is responsible for the communication between the server and the client.

2. The View represents the application's interface, meaning it visualizes the data contained in the Model, for example the representation of a dictionary's entry. Furthermore, in the case of user input, the View forwards it to the Controller.

3. The Controller manages the request flow, which involves retrieving processed data from the View (and other sources), cleansing and normalizing it before forwarding it to the Model. An example of use would be processing the information, in this case the lexeme a user has searched for in the search field, provided for this purpose.

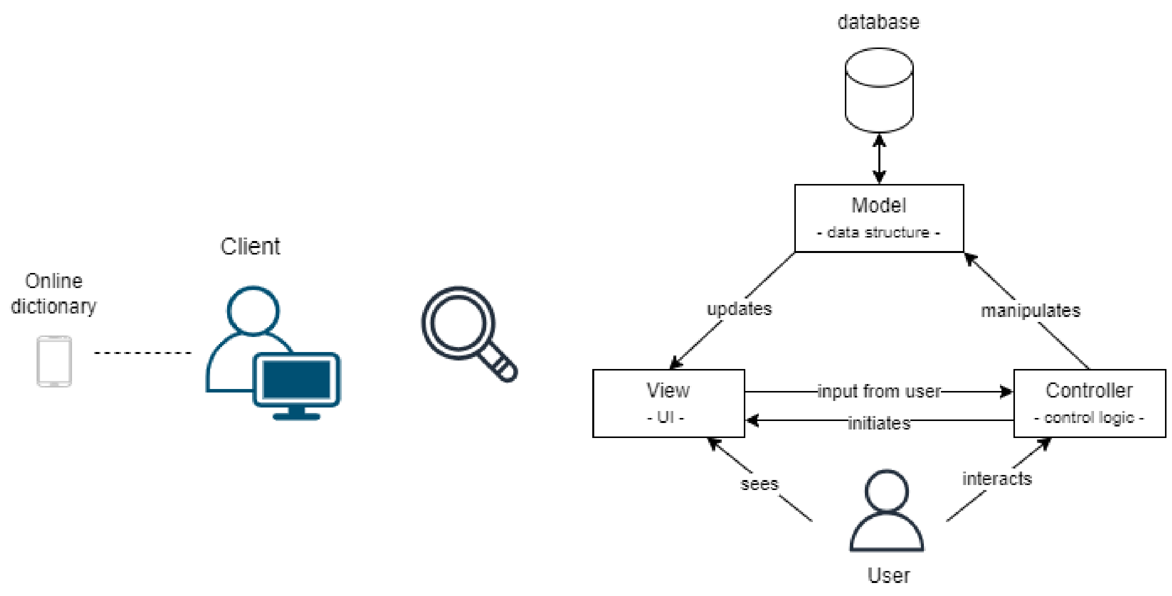


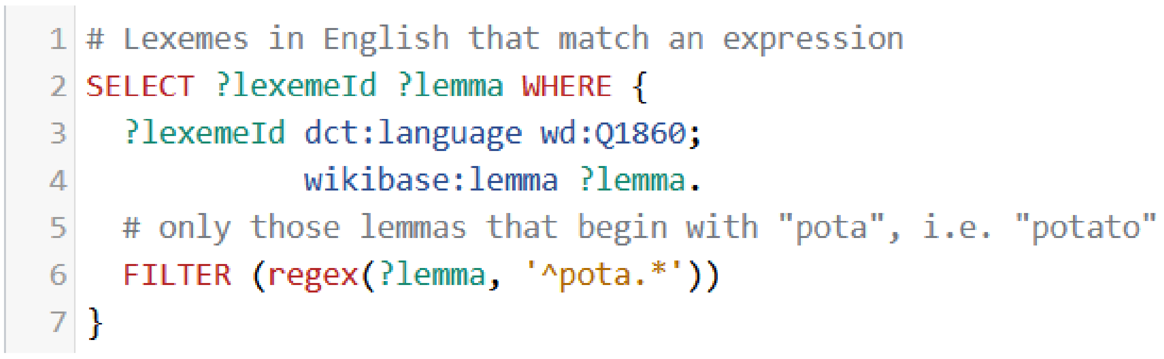
Figure 3.8: The client's architecture
[Buc09]

3.2.5 Data output

How the data from Wikidata gets into the application is one of the main questions and requires the introduction of the interface SPARQL.

SPARQL

A form of query is needed to search for a particular element in WD:LD; more specifically, a SPARQL query. According to the W3C recommendations, "[...] SPARQL can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware [...]". [PS08] While the system shares similarities with the SQL (Structured Query Language) scheme, the underlying formats are distinct. SQL is primarily used for relational databases. The functionality of a SPARQL query will be explained by the example *Lexemes in English corresponding to an expression* in Figure 3.9.
The `SELECT` element is intended to return variables and their bindings, such as `lexemeID` and `lemma` in this case. Sometimes a * is added to the end of `SELECT` to get all the variables in an entry.
The item `WHERE` checks the comparison between the basic graph pattern (which contains the variables subject, object and predicate) and the data graph, also known as the database



```
# Lexemes in English that match an expression
SELECT ?lexemeId ?lemma WHERE {
  ?lexemeId dct:language wd:Q1860;
            wikibase:lemma ?lemma.
  # only those lemmas that begin with "pota", i.e. "potato"
  FILTER (regex(?lemma, '^pota.*'))
}
```

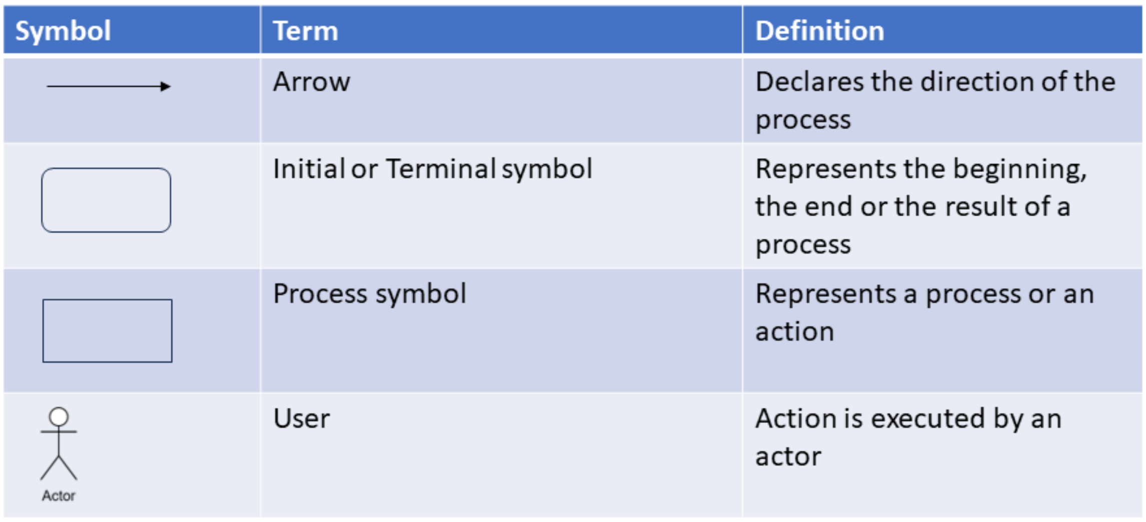
Figure 3.9: The query for *Lexemes in English that match an expression* [Wik]

of Wikidata. "A basic graph pattern matches a subgraph of the RDF data when RDF terms from that subgraph may be substituted for the variables and the result is RDF graph equivalent to the subgraph."[PS08]
In Figure 3.9, the language of the `lexemeID` is determined using *value of entity* (wd:) with the value Q1860 representing the language *English*. Also, the query is restricted to the wikibase lemma, which is the dictionary form of lexemes.
Last but not least, the query is filtered by the `FILTER` element to output all lemmas that start with a `pota`. `Regex` here announces a regular expression which is as follows: `^pota.*`. The functionality of regular expressions is not discussed in detail.

3.3 UML Diagrams

The subsequent sections elucidate the app's structure and its design using Unified Modelling Language diagrams (abbreviated to UML), which depict a blueprint of the application and apply the non functional requirements, defined in chapter 3.1.
Figure 3.10 provides an overview of the employed symbols and notations.

Since the primary framework of the existing applications is intended to remain unaltered, the ensuing diagram 3.11 corresponds to the bachelor thesis version of 2018. [Bol18] Upon initiating the application, the Start Screen will be loaded automatically. By clicking any of the buttons – namely, the Settings Button, Search Button, Categories Button, or Favorites Button – the respective screen will be displayed accordingly.



Symbol	Term	Definition
→	Arrow	Declares the direction of the process
▢	Initial or Terminal symbol	Represents the beginning, the end or the result of a process
□	Process symbol	Represents a process or an action
Actor	User	Action is executed by an actor

Figure 3.10: Explanation of used symbols in the subsequent UMLs

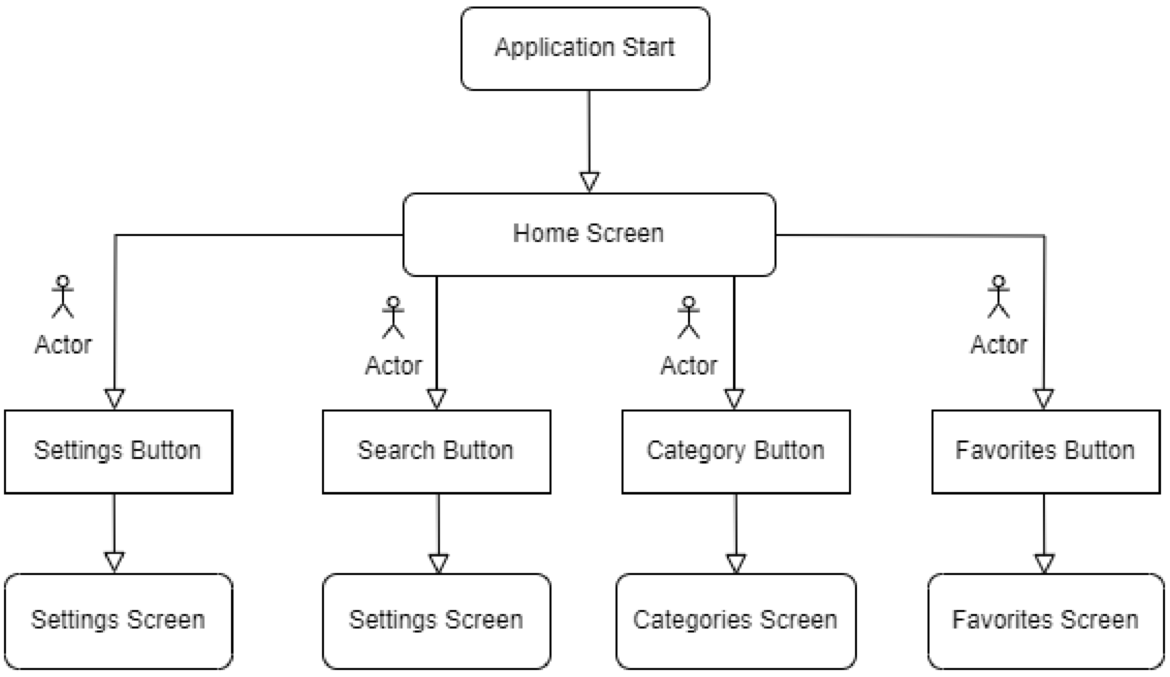


Figure 3.11: Fundamental functionality of the application

In addition to the presented buttons, which direct users to the various screens, an option to switch between English and Hausa languages is provided.

3.4 Methods

For the implementation some methods will be presented.

3.4.1 Cross-platform programming

This section and the following one about Flutter/Dart is based on the German book "App-Entwicklung mit Dart und Flutter 2".[Mei21]
The idea of programming cross platform is to make software available on different devices and platforms no matter the operating system. As a result it is not necessary to program an application twice, instead the code can be deployed on several platforms.
Basically there are two approaches to cross-platform programming.
The first approach deals with web technology, as websites are embedded in the application and the whole screen is filled by a browser component; Java as the programming language and an API as the connection between JavaScript and the hardware. However, the performance is determined by the browser application, as well as the limiting functions that must be available in the application and on the website.
The other and better approach is to use unspecified languages that are then adapted to the environment through compilation, such as Xamarin, React, Ionic, Flutter/Dart etc. In the following, only Xamarin and Flutter/Dart will be discussed and compared.

Introduction to Xamarin

According to the official Microsoft.NET website, Xamarin, established in 2011, is an open-source platform for developing applications for iOS, Android, and Windows using .NET. [Mic] Approximately 80% of the code is written in C# within the Visual Studio Code environment to share business logic, while the remaining code, responsible for the native UI, is developed separately for each platform, allowing them to retain their unique characteristics. .NET offers benefits such as memory allocation, garbage collection, and interoperability with other platforms.

Introduction to Dart

The concept of adapting the language to the environment is also applied within the Dart programming language as part of the Flutter framework.
Dart was originally developed by Google in 2013 and has since evolved to its second version, making it compatible with mobile devices, consoles, and web applications. It was only with the introduction of Flutter in 2018 that Dart could be employed in an object-oriented style similar to Java and C. Mobile devices equipped with Flutter feature a virtual machine with a just-in-time compiler to execute Dart. Upon the application's release, the code undergoes a second execution phase with an ahead-of-time compiler, resulting in the generation of native machine code, namely ARM for Android or x86_64 for iOS.
Dart has become a popular choice for cross-platform development, thanks to its extensive collection of packages and tools. Additionally, Dart's unique UI toolkit offers developers flexibility in design. Another standout feature of Dart is its Hot Reload functionality, allowing developers to make real-time changes to their code, significantly speeding up the development process.
Based on these reasons, it is decided to use Dart as the programming language.

For the development environment, Android Studio was chosen because of the option of running the code on a emulator. [And]

3.4.2 Programming concepts

The concept of Dart is based on widgets, which are "[...] the central class hierarchy in the Flutter framework." [Dar] That means everything the user sees on the display, from windows and interfaces to text boxes and the application itself, is composed of widgets. These widgets can be converted into elements that manage the underlying render tree. The state of a widget cannot be modified, so the standard state is set as `final`. As an alternative, StatefulWidget is offered, whose states can be changed multiple times during the widget's lifetime. Additionally, there is the option of a StatelessWidget, created recursively from other widgets, which makes it easier to describe the interface. Lastly, there is InheritedWidget, used for passing information from one class to another.
A widget always needs to be defined by the `build()` method, which returns a widget or, in most cases, a tree of widgets that are cached.
Another important feature is the Scaffold class, which is the top-level widget that implements the basic structure for Material Design. It allows other features to be directly implemented, such as the AppBar, which functions like a fixed menu created from buttons leading to various actions.
The rest of the Dart code is similar to CSS; a body is returned containing children with texts, and everything can be individually designed using parameters like `colorScheme`, `padding`, and more.

3.4.3 Interface to the backend

One of the main aspects of this thesis is to implement the data from WD:LD to the application. Therefore an interface has to be integrated which leads to the following options:

1. The SPARQL query is available in Python code which could be initialised through packages, such as the Chaquopy package, a plugin to run Python code on Android. [Cha]

2. The code serving for the backend server could be written in Flask, a web framework for building web applications and APIs. Dart then communicates with Flask through HTTP requests, enabling the application to send and receive data from the Flask server. [Pyt]

3. The function of a FutureBuilder which is a widget and allows asynchronous functions through the use of Future objects, enabling developers to write non-blocking, responsive applications. The Builder function is called when the Future completes. [Dar]

Considering the advantages of a FutureBuilder, particularly its flexibility and asynchronous rendering, the decision is made to utilize it as the interface to Wikidata. To elaborate further, the FutureBuilder can be extended and customized to suit a range of use cases and scenarios, thereby improving code reusability and maintainability. Additionally, it seamlessly integrates with other user interface components and widgets, enabling asynchronicity in the application without interrupting the overall user interface flow. In Chapter 4.1.1 the applied function is presented in a more detailed form.

4 Implementation

Following the exploration of fundamental content relevant to the application, such as HSL knowledge, the functionality of WD:LD, or the design of the application along with its requirements and implementation methods, now the implementation phase itself is proceed. Therefore the requirements defined in Chapter 3.1 have to be considered, first and foremost the one about the usability across platforms, as well as the interface WD:LD which intends to serve as the backend.

4.1 Functions

For the interface to the backend Wikidata, two basic functionalities were considered.
The FutureBuilder, demonstrated in the upcoming Chapter 4.1.1 basically stores the current state of the Future and displays them by asking.
The other tool for accessing the backend involves using SPARQL queries. While the basic functionality of SPARQL is introduced in Chapter 3.2.5, its specifics will be revisited in the upcoming code-related discussions.

4.1.1 Function: Future Class

In the following, the code of the `fetchSparql` function, see 4.1, involving a Future Class is examined step by step.
By `fetchSparql` the function is initialised together with the input of the Search query (explained below) being later initiated. The `async` parameter introduces the asynchronous code containing the endpoint Url and the SPARQL query followed by a `try` and `catch` loop for catching any exception that may occur during the search. A `http.get` request is raised to the Wikidata SPARQL endpoint with the query checking if the response is okay (`==` `200`). In the case of a positive answer the JSON response is parsed using `jsonDecode`, which is then passed to the `processSparqlResults` function to extract relevant information and convert it into a list of `lexemeEntry` objects.
In the case of a negative response, again an exception is thrown.

```
Future<List<LexemeEntry>> fetchSparql(String searchQuery) async {
  final endpointUrl = "https://query.wikidata.org/sparql";
  final query = '''
    ...
  ''';

  try {
    final response =
    await http.get(Uri.parse('$endpointUrl?format=json&query=$query'));

    if (response.statusCode == 200) {
      final parsedResponse = jsonDecode(response.body);
      final searchResults = processSparqlResults(parsedResponse);
      return searchResults;
    } else {
      throw Exception('Failed to load search results');
    }
  } catch (e) {
    // Handle any exceptions that may occur during the search.
    throw Exception('Search failed: $e');
  }
}
```

Listing 4.1: Future Class

4.1.2 Function: SPARQL query

To retrieve data from Wikidata, a SPARQL query is utilized which is now presented.
The `SELECT` clause specifies the properties to be obtained, including `lexemeId`, `lemma`, `wird_beschrieben_in_URL` (which leads to the corresponding entry of the official HSL website), `full_work_at` (which leads to the URL of the underlying image) and `gloss`.
The query structure is designed to facilitate the addition of more parameters, for example *label in sign language*.
The `WHERE` clause filters the query, implementing pattern matching. Specifically in `?lexemeId dct:language wd:Q3915462;` the item Q3915462 restricts the results to the HSL language. `?lexemeId wikibase:lemma ?lemma` ensures that the lexeme has a lemma, whereas `?lexemeId ontolex:sense ?sense` links the lexeme to its sense.
`p:P973 _:b10. _:b10 ps:P973 ?wird_beschrieben_in_URL; pq:P953 ?full_work_at.` extracts URLs related to the lexeme. A blank node _:b10 is created defining a statement with the property P973 (described_at_URL). In a next step the blank node is specified by `described_at_URL` and `full_work_at`.
`?sense skos:definition ?gloss` retrieves the gloss (definition) of the sense associated with the lexeme, filtered by `FILTER(LANG(?gloss) = "en"` for including only the English glosses.
The FILTER clause at the end imposes an additional condition on lemmas, requiring them to contain the `searchQuery`. The use of `lcase` ensures case-insensitive matching by converting the lemma to lowercase.
The focus currently centers on the mentioned four properties and the presented conditions, but the query structure allows for easy extension with additional parameters.

```
SELECT ?lexemeId ?lemma ?wird_beschrieben_in_URL ?full_work_at ?gloss WHERE {
  ?lexemeId dct:language wd:Q3915462;
    ontolex:sense ?sense;
    wikibase:lemma ?lemma.
    p:P973 _:b10.
  _:b10 ps:P973 ?wird_beschrieben_in_URL;
    pq:P953 ?full_work_at.
  ?sense skos:definition ?gloss .
  FILTER(LANG(?gloss) = "en")

  FILTER (contains(lcase(?lemma), "${searchQuery.toLowerCase()}")).
}
```

Listing 4.2: SPARQL query

4.1.3 Function: Search

The search function is a critical component for enhancing user-friendliness, and its implementation is detailed below.
The function called after its main purpose `search` is responsible for initiating a search operation based on the user input, fetching the results asynchronously using the `fetchSparql` function, see 4.1 and then navigating to a new screen to display the obtained search results. First of all it takes a `BuildContext` named context as its parameter. `Async` indicates that the function involves asynchronous operations, such as network requests. Then the search query is extracted from a `searchController`, resulting into a text input field for the search query.
The `await` keyword is used to perform another asynchronous operation: the `fetchSparql` function is called with the extracted `searchQuery`. This function is expected to return a Future that resolves to a list of search results.
Once the search results are obtained, the `Navigator.push` method is used to navigate to the `SearchResultsScreen`.
The `MaterialPageRoute` is used to define the destination screen and includes a builder function that takes the context and returns an instance of `SearchResultsScreen`, passing the retrieved `searchResults` as a parameter.

```
void search(BuildContext context) async {

    String searchQuery = searchController.text;
    final searchResults = await fetchSparql(searchQuery);
    Navigator.push(
      context,
      MaterialPageRoute(
        builder: (context) => SearchResultsScreen(searchResults:
   searchResults),
      ),
    );
  }
```

Listing 4.3: Search Function

4.1.4 Function: Displaying an entry

Last but not least, the function for displaying an entry is shown.
The `if (snapshot.hasData)` condition checks if the asynchronous operation has completed and data is available. If data is available (`snapshot.hasData` is true), the `snapshot.data` property is accessed to extract the data obtained from the asynchronous operation.
The extracted data is then used to populate a `ListView` widget with various child widgets, including Text widgets to display the `lexemeId`, `lemma`, and `gloss`, and an Image widget to display an image fetched from the URL provided in the `full_work_at` property of the data object.

```
if (snapshot.hasData) {
            final data = snapshot.data; // Extract the data here

            return ListView(
              children: <Widget>[
                Column(
                    children: <Widget>[
                      Text(data!.lexemeId.replaceAll("http://www.wikidata.org/entity/", '')),
                      Text(data!.lemma),
                      Text(data!.gloss),
                      Image.network(data!.full_work_at),
                    ],
                  ),
```

Listing 4.4: Search Function

4.1.5 Function: Favorites

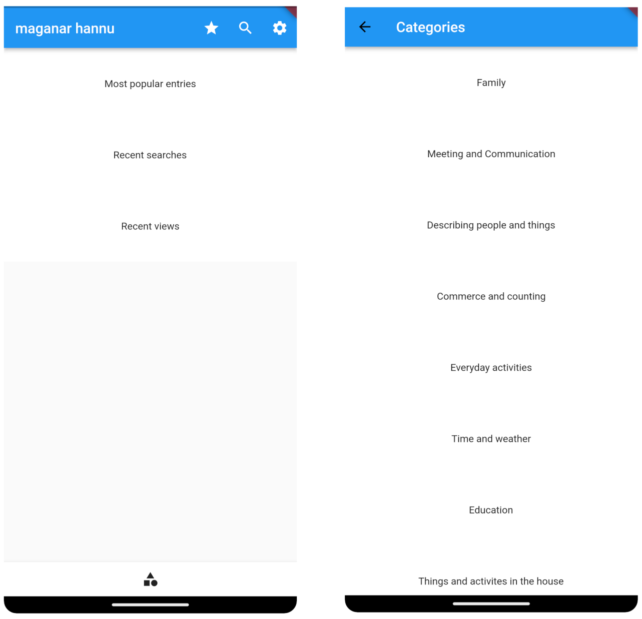
For implementing the function of adding entries to the Favorites Screen, the SharedPreferences package is utilized because there is no local database storing the entries. This package allows the application to persistently store and retrieve data by the lexemeId of the entries marked as favorites by the user. By using SharedPreferences, the application can remember the user's preferences even after the app is closed and reopened.
The function `onLikePressed` defining the process of clicking the Add to Favorites button , takes the defined `likedEntry` containing the specific lexemeId and prints it to the console. The function then retrieves an instance of `SharedPreferences` using the `getInstance` method, which allows the application to store and retrieve key-value pairs persistently. Using the `setStringList` method of SharedPreferences, the lexemeID of the liked entry is added to a list of favorites stored `Favorites`. If no list exists yet, a new one is created. The whole function is embedded in a FutureBuilder, used to asynchronously retrieve data from the SharedPreferences, which stores the list of liked entries. Once the data is fetched, the builder function of the FutureBuilder is called, allowing the user interface to be updated with the retrieved data.

```
Future<void> onLikePressed(String likedEntry) async {
  print(likedEntry);
  final SharedPreferences prefs = await SharedPreferences.getInstance();
  await prefs.setStringList('Favorites', <String>[likedEntry]);
}
```

Listing 4.5: Search Function

4.2 Presentation of the frontend

After highlighting the functions, the prototype of the application itself will be presented. Since this is a prototype version, the design was adopted from the existing basic settings of the emulator of the environment Android Studio. [And] The Start Screen, see Figure 4.1 a), offers three main categories, enveloped the Most popular entries, Recent searches and Recent views. Besides at the top of the bar there is the option to navigate to the Favorite entries in form of a the star icon search option. The magnifying glass leads to the Searching Screen, while the gear icon leads to the Settings Screen. At the bottom of the bar a grid-based layout icon initialises the Categories Screen, which is demonstrated in Figure 4.1 b). The categories were adopted by the currently existing application, namely Family, Meeting and communication, Describing people and things, Commerce and counting, Everyday activities, Time and weather, Education, Things and activities in the house and Religion.



maganar hannu

Most popular entries

Recent searches

Recent views

Categories

Family

Meeting and Communication

Describing people and things

Commerce and counting

Everyday activities

Time and weather

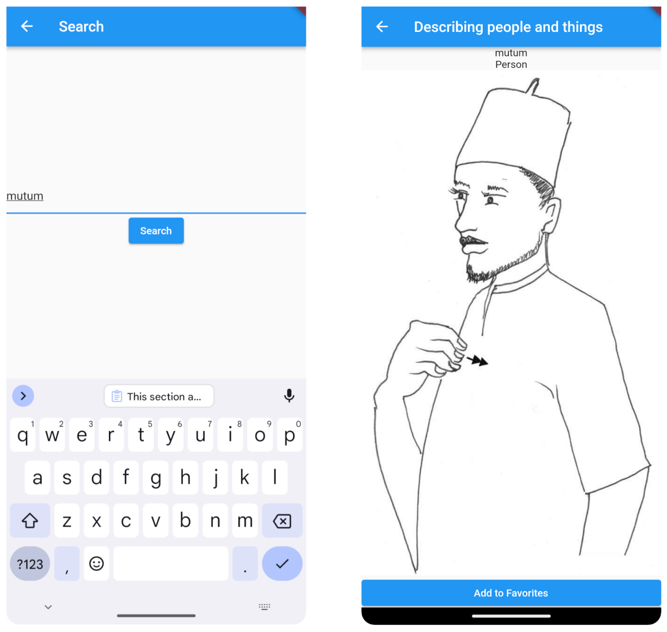
Education

Things and activites in the house

a) Start Screen b) Categories Screen

Figure 4.1: Screenshots of the implementation in the prototype application

The screenshot shown in Figure 4.2 a) illustrates the Search Screen, where users can input any lexeme and initiate the search by clicking the Search button. In Figure 4.2 b), the exemplary entry MUTUM is displayed. It appears when MUTUM is entered into the search bar and is also displayed when selected to be shown in one of the sections or categories, such as Describing people and things.
MUTUM serves as the lemma, while the subsequent line provides the gloss ”person”.
The sketch below is adapted from existing sketches and is loaded through the URL of the official HSL website [Sch]. Users can scroll the screen to access the Add to Favorites button, which changes its color to red when clicked and is subsequently added to the Favorites Screen.



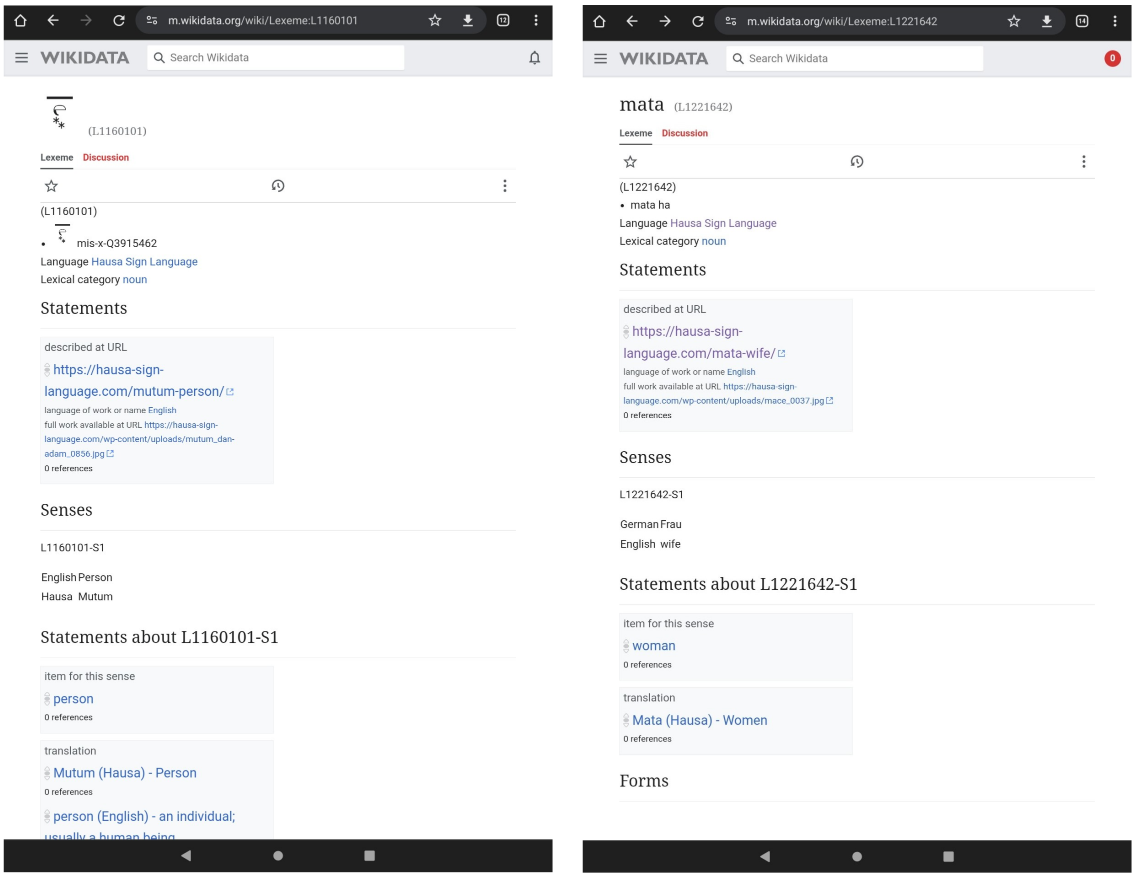
a) Search Screen b) Entry Screen

Figure 4.2: Screenshots of the implementation in the prototype application

4.3 Presentation of the backend

Last but not least, the exemplary WD:LD entry will be executed. Figure 4.3 shows the two exemplary entries created, MUTUM, written in SSW, and MATA. As previously mentioned, Wikidata conventionally labels sign language entries using SSW. Since MUTUM has already been overwritten in accordance with this convention, the application requests the English gloss to circumvent its display.
By creating an entry, an ID is automatically generated (namely L1160101 and L1221642), while the language and the category of the lexeme have to be specified, as already explained in Chapter 3.2.3; in the following *HSL* and *noun*. The abbreviation *ha* is for Hausa, as currently HSL is not saved and consequently has no abbreviation of its own.
In the Statements section, the source of the lexeme is linked via *described at URL*, also known as the official website of HSL [Sch]. The image for the sign itself is directly linked via *full work available at URL*.
The Senses section is comprised of a brief description, also known as the gloss, and a translation, which in both cases do not differ due to their simplicity, namely *Person* and *Woman*. The entry of MUTUM clarifies the meaning of the reference, as the Statements of Sense S1 (person) are linked beneath the Senses section. Consequently, the definition of a person as one option of the property P5972 *translation* is included as part of this entry.

Figure 4.4 displays the Statements section of MUTUM's entry, providing a closer view from the perspective of a logged-in user. Edit buttons, along with options to add references, values, and statements, are enabled and accessible.


m.wikidata.org/wiki/Lexeme:L1160101

WIKIDATA Search Wikidata

(L1160101)

Lexeme Discussion

(L1160101)

- mis-x-Q3915462

Language Hausa Sign Language

Lexical category noun

Statements

described at URL

https://hausa-sign-language.com/mutum-person/

language of work or name English

full work available at URL https://hausa-sign-language.com/wp-content/uploads/mutum_dan-adam_0856.jpg

0 references

Senses

L1160101-S1

English Person

Hausa Mutum

Statements about L1160101-S1

item for this sense

person

0 references

translation

Mutum (Hausa) - Person

0 references

person (English) - an individual; usually a human being

m.wikidata.org/wiki/Lexeme:L1221642

WIKIDATA Search Wikidata

mata (L1221642)

Lexeme Discussion

(L1221642)

- mata ha

Language Hausa Sign Language

Lexical category noun

Statements

described at URL

https://hausa-sign-language.com/mata-wife/

language of work or name English

full work available at URL https://hausa-sign-language.com/wp-content/uploads/mace_0037.jpg

0 references

Senses

L1221642-S1

German Frau

English wife

Statements about L1221642-S1

item for this sense

woman

0 references

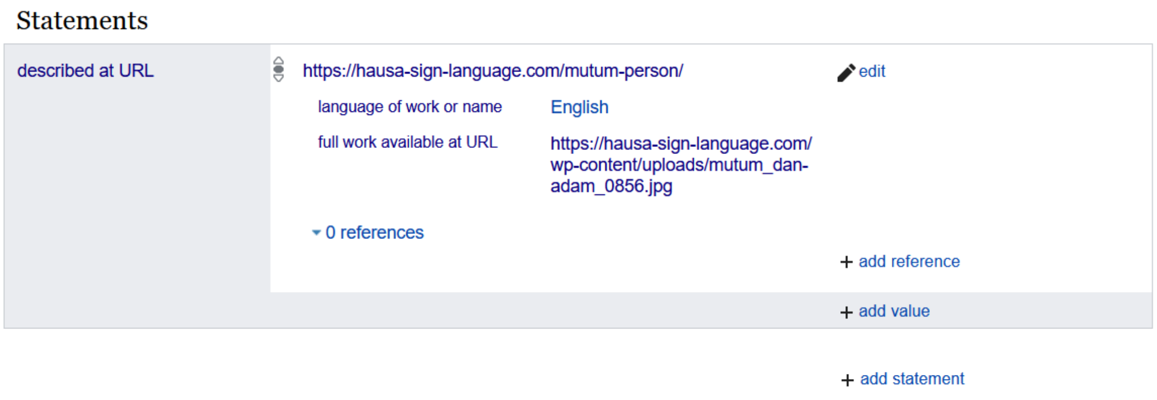
translation

Mata (Hausa) - Women

0 references

Forms

Figure 4.3: The WD:LD entries for MUTUM and MATA


Statements

described at URL

https://hausa-sign-language.com/mutum-person/ edit

language of work or name English

full work available at URL https://hausa-sign-language.com/wp-content/uploads/mutum_dan-adam_0856.jpg

0 references

+ add reference

+ add value

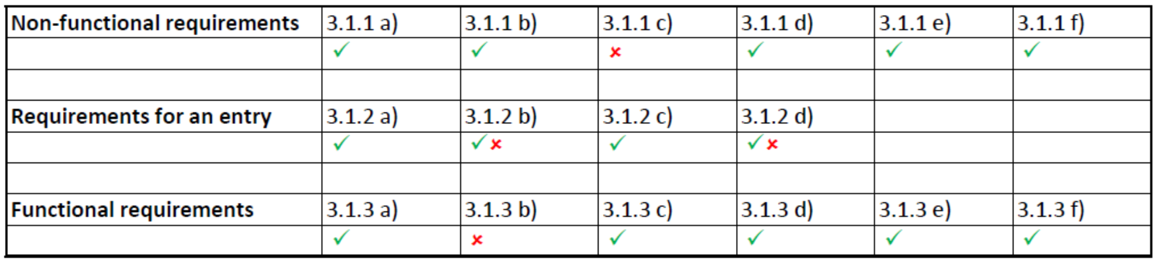
+ add statement

Figure 4.4: Zoom for the WD:LD entry MUTUM

5 Evaluation

When evaluating the application, two tools are employed. The first part involves checking the requirements outlined in 3.1. The second part addresses the task description, explores its advantages, examines potential disadvantages, and proposes approaches to mitigate them.

5.1 Review of the requirements



Non-functional requirements	3.1.1 a)	3.1.1 b)	3.1.1 c)	3.1.1 d)	3.1.1 e)	3.1.1 f)
	✓	✓	✗	✓	✓	✓
Requirements for an entry	3.1.2 a)	3.1.2 b)	3.1.2 c)	3.1.2 d)		
	✓	✓✗	✓	✓✗		
Functional requirements	3.1.3 a)	3.1.3 b)	3.1.3 c)	3.1.3 d)	3.1.3 e)	3.1.3 f)
	✓	✗	✓	✓	✓	✓

Figure 5.1: Overview of the (not) fulfilled requirements

3.1.1 a) **Navigation to different screens**
The user has the option to choose from buttons such as Settings, Search and Categories, see Figure 4.1 a). When selecting one of these buttons, the corresponding screen is opened, as exemplified in Figure 4.1.

3.1.1 b) **Views for the most popular entries, recent views and recent searches**
Boxes using the Widget scheme (presented in Chapter 3.4.2) were created which can be clicked on to get a navigation to recent views, favorite entries and most popular entries, see Figure 4.1 a). The individual screens cannot be populated yet due to the absence of a local database. However, this limitation can be overcome by leveraging the SharedPreferences package, which was introduced in Chapter 4.1.5.

3.1.1 c) **Selecting languages from the Settings Screen**
This requirement can be fulfilled by utilizing a placeholder for all the text within the application. A file could then replace the placeholder with the word corresponding to the language selected by the user.

3.1.1 d) **Adoption of the categories from the current application**
The categories, such as Family, Education, etc., are adopted, as evidenced by Figure 4.1 b). Once again, the Widget scheme was utilized to implement them.

3.1.1 e) **Implementation of the functionality of the Favourite Entries through an icon star**

Instead of an icon star, a button labeled by Add to Favorites was inserted which then calls the function to insert the corresponding entry to the Favourite entries class, compare to Figure 4.2 b). The implementation of this function is not as straightforward as usual due to the absence of a local database, but it is executed exemplary to validate its feasibility, compare to Chapter 4.1.5.

3.1.1 f) **Navigation through a magnifying glass icon**
By clicking on the magnifying glass, a new screen is opened, see Figure 4.2 a). Upon clicking the Search button, the screen displays the corresponding entry, as shown in Figure 4.2 a) and b). Suggestions during the typing process are not provided due to the SPARQL query, which is executed only once upon completing the search.

Following that, the requirements defined for the entry in the frontend, as well as in the backend, must be checked.

3.1.2 a) **Requirement of an easy adaption of an entry**
The structure of the WD:LD entry, as described in Chapter 3.2.3 for creating an entry and in Chapter 4.3 for editing an entry, enables straightforward operation. Upon logging into an account, the "edit" button is activated, along with the option to add statements, values, etc., as depicted in Figure 4.4.

3.1.2 b) **Presentation forms for the sign, such as illustrations or videos**
The illustrations were adapted from the current application and are parsed through the *full work available at URL* link in the Statements section. Therefore the `Image.network (data!.full_work_at)` function was used. However videos need to be created first in order to implement them, which is not part of this work.

3.1.2 c) **Implementation of a gloss or a spoken translation of the lexeme**
The Sense section, as shown in Figure 4.3, includes the glosses in English, namely "Person" and "Woman" as well as in Hausa, namely MUTUM and MATA. Both serve as the spoken translation for the lexemes. These glosses are also requested in the frontend, as illustrated in Figure 4.2 b).

3.1.2 d) **Requirement of an HSL lexeme**
Wikidata requires SSW as the running declaration system for Sign Language lexemes. Therefore, they are called through SSW, see 4.3 on the left side MUTUM. Since that system of writing signs is not practical for users (mainly the deaf community), it is not displayed in the frontend.

Last but not least, the functional requirements will be verified.

3.1.3 a) **Usability across platforms**
Dart as the chosen programming language ensures the usability across platforms, as presented in Chapter 3.4.1.

3.1.3 b) **Update through an URL link**
Since the application is in its prototype stage, integrating a URL link for updates is unnecessary. Once the application reaches a publishable level, accessing the code of the current version will be necessary for fulfilling this step.

3.1.3 c) **Database Efficiency**
The entries are loaded in real time through a SPARQL query from the backend WD:LD, serving as the database, containing all entries that can be updated easily. This ensures that the data coming from the backend is always up to date. For verification purposes, Chapter 4.1.1 is referenced to assess the functionality of the SPARQL request.

3.1.3 d) **User-friendliness**
While the full extent of user-friendliness may not be fully verified in a prototype, ensuring user-friendliness in its design and functionality is crucial for providing a positive user experience. Features like intuitive navigation, clarity, consistency, accessibility across different platforms, and a functional search are essential components of user-friendliness and have been demonstrated in other requirements, such as in the requirement 3.1.1 f).

3.1.3 e) **Minimization of the memory consumption**
Since the application is currently only a prototype version, this requirement cannot be fully verified. Nevertheless, it can be confirmed that the largest source of storage memory, the images, do not consume any storage memory until they are loaded in real-time via the SPARQL query.

3.1.3 f) **Reliability and stability of the application**
This requirement cannot be fully verified until the application is no longer in the prototype stage. To ensure this requirement is met, rigorous testing and user feedback will be necessary once the application is fully developed and deployed. Additionally, continuous monitoring and updates based on user experiences will be essential to ensure that the application maintains its reliability and stability over time.

5.2 Evaluation summary

The main research question of this thesis, represented in Chapter 1.2, is demonstrating the feasibility of the implementation of Wikidata as the backend for a sign language dictionary. This requirement is met, as evidenced by the presented function of a FutureBuilder in Chapter 4.1.1, the corresponding SPARQL query in Chapter 4.1.2 and the Figures in Chapter 4.2. Nevertheless questions are raised about benefits and disadvantages towards other solutions, thus evaluating the overall feasibility.
First, the key advantages of the proposed solution are summarized.

1. The benefits of Wikidata:Lexicographical towards other online sign dictionaries are presented in Chapter 2.3.1. Utilizing systems like RDF, which provides a standardized approach to describing resources and their relationships, the application can continuously exchange and update data.

2. Furthermore, the server infrastructure does not require maintenance as in other applications with their own databases, such as those in the current HSL applications. Instead the service is already built and maintained by Wikidata.

3. Computing capacity presents fewer challenges, given that the data, including a lot of images, is outsourced.
4. The SPARQL interface between the frontend and the backend is a customizable tool that can be modified and adapted at any time according to the requirements of the content of an entry in the frontend.
5. The application is compatible with all devices regardless of their underlying operation system, facilitated by the use of Dart as the programming language, which fulfills this requirement.

During the course of the work, certain deficiencies were identified and addressed. These issues will be discussed, along with potential solutions.

1. Since WD:LD is not a local database, one component of the interface as the SPARQL query had to be defined. As already mentioned before, the SPARQL query then returns the results in-time which can lead to time delay, also known as the network latency. This delay may arise from several factors, including physical distance between the client and server or the processing time required for data transmission and routing through intermediary network devices.
2. The application is evidently dependent on internet connection.

Both issues are critical, especially considering that the sign dictionary is primarily used in northern Nigeria, an area with poor internet connectivity.
In order to circumvent the presented dependency problem, the data could be cached locally during opening the application the first time, repeated periodically for getting updates.
The reliance on external regulations poses another challenge when working with databases not developed in-house.

3. For instance, the ontology system in WD:LD for titling the entries for sign languages is managed using SSW, which is not the preferred written form for sign languages. As previously noted, it is not utilized by the deaf community and, therefore, proves ineffectiveness for on-screen display. Furthermore, the system poses a barrier to the development of sign languages in Wikidata, as it requires expertise for this type of declaration. Moreover, as mentioned in Chapter 2.4.1, it is challenging to render the SSW system for the average user, diminishing its utility.
   In 4.3 the entry MUTUM is declared by SSW and is exemplary for the described issues.

For the time being, the solution involves displaying the Gloss, a representation form of the sign in a spoken language in the application instead of the declaration lexeme, namely the SSW symbol, as outlined in Chapter 5.1 for the Entry Requirement of an HSL lexeme (d).
In the future, the focus should be on completing lexemes across different sign languages and enhancing the diversity of sign languages represented. One potential approach to achieve this is by adopting a declaration system that utilizes glosses, among other methods. Additionally, linking of items should be further explored to enhance the connectivity and interrelation

between different concepts and entities within the WD:LD framework. Lastly, prioritizing the direct implementation of images depicting signs is essential, rather than relying on external properties like *described in the URL*.
This shift could streamline the process and improve accessibility.

6 Conclusion and future work

This chapter summarizes the paper and provides an outlook on potential options for the application.
In the context of this bachelor thesis, the feasibility of developing an application for a cross-platform online sign language dictionary has been demonstrated, utilizing WD:LD as the backend. Due to the incorporation of two unfamiliar subject areas, namely the sign language Hausa and Wikidata, only a prototype demonstrating approximate functionality was ultimately developed.
At the beginning the structure of HSL and some sign examples were highlighted, alongside basic definitions pertaining to Wikidata, Wikidata as a platform, and its underlying dictionary WD:LD. The main advantage of WD:LD is the structure of an entry offering multiple options defined by properties, with additional features, such as translations, captured in the Forms and Senses sections. Currently, WD:LD represents sign language entries using SSW, a formal transcription system for writing sign languages. However, as it is not practical, especially for the deaf community, an alternative approach is to focus on providing the gloss, or spoken translation, instead. The main content of the applications, as for example the categories, being now available intends to stay the same, so it was adapted.
Before the application was implemented, requirements were set. The main non-functional requirement was to maintain the original functionality, such as navigating to different screens through buttons and views. The main functional requirement was to ensure usability across platforms. Then the architecture's structure was highlighted. Since the backend architecture was established with Wikidata, the client-server architecture was chosen for the system model. An essential component of the backend is the Varnish Cache, which serves as a service between Wikidata and other users and tools, managing load balancing. The underdevelopment of the WD:LD platform in terms of sign representation was highlighted, particularly concerning the low usage of certain properties within the framework. Demonstrating data input and output was another significant aspect. For data input, a template is provided by WD:LD, while data output requires SPARQL queries. These queries follow the SQL scheme, including `SELECT`, `WHERE`, and `FILTER` clauses. Additionally, various methods were introduced, such as Dart, the underlying programming language for cross-platform programming, and the interface to the backend, which includes the function of a FutureBuilder allowing asynchronous programming. Other programming concepts within Dart were introduced, primarily focusing on the system of widgets, which arranges each component from a display through a window to an interface.
The subsequent chapter detailed the implementation, highlighting key functions including the FutureBuilder, the SPARQL query, the Search function, the display entry function and the Favorites function. The main elements in the SPARQL query included the items to be potentially displayed, such as `lexemeId`, `lemma`, `wird_beschrieben_in_URL` and `full_work_at`. The latter two parameters both involved URL links, with `full_work_at` containing the URL of the sign sketch. Later, the function responsible for displaying an entry can selectively choose the specific items to be shown.

The Favorites function was highlighted as a workaround for the lack of a local database, leveraging the SharedPreferences package for persistent storage of key-value pairs.
The frontend presentation was demonstrated through screenshots captured on a emulator within the Android Studio environment, showcasing the user interface. Meanwhile, the backend implementation was illustrated by a screenshot depicting an example entry created for demonstration purposes.
Following this, the initial requirements were thoroughly assessed and reviewed. With the exception of the language selection requirement, all other requirements were either completely met or partially fulfilled.
In a Evaluation summary, the main advantages and disadvantages were summarized.
The benefits of Wikidata:Lexicographical over other online sign dictionaries were outlined, including its use of RDF for continuous data exchange and updates without requiring additional maintenance. However, drawbacks of using Wikidata as the backend were identified. These include dependency on internet connection and reliance on external regulations. Caching data locally and periodic updates were proposed as solutions. In response to regulations like WD:LD's ontology of declaring lexemes in sign languages as SSW, requesting the gloss instead was suggested as an alternative approach.
In this context, it was highlighted that WD:LD requires further development, particularly concerning the presentation of sign languages. One approach could involve the direct implementation of images that demonstrate signs, rather than relying on external URLs for outsourcing them.

For future work, several points are emphasized.
It is important to extend the implementation, which includes implementing and verifying the remaining functions such as recent views and recent searches. The current solution using SharedPreferences has limitations in storing a significant number of entries locally, indicating that alternative approaches may need to be explored.
Another aspect of future work could involve refining the current approach to address the load balancing of WD:LD's content. Caching data locally at application startup and updating it periodically may not be the final solution.
In conclusion, the development of the structure of WD:LD's entries is a critical aspect that may impact SPARQL queries and overall system performance. This aspect should be taken into consideration when finalizing the WD:LD entries.

List of Figures

Bibliography

[And] *Android Studio.* https://developer.android.com/studio. – Online; accessed 03-March-2024

[Bol18] BOLAT, Emre: *Entwicklung einer Client-Server Anwendung für ein Gebärdensprachen-Wörterbuch.* 2018

[Buc09] BUCANEK, James: Model-view-controller pattern. In: *Learn Objective-C for Java Developers* (2009), S. 353–402

[Cha] *Chaquopy.* https://chaquo.com/chaquopy/. – Online; accessed 03-March-2024

[Dar] *Flutter.* https://docs.flutter.dev/. – Online;accessed 15-October-2023

[EB] ENCYCLOPAEDIA BRITANNICA, The E.: *Sign language - communications.* https://www.britannica.com/topic/sign-language. – Online; accessed 22-May-2023

[Fou] FOUNDATION, Starkey H.: *Deafness and hearing loss.* https://www.who.int/health-topics/hearing-loss#tab=tab_1. – Online; accessed 22-May-2023

[Fra] FRANCE, Wikimédia: *Lingua Libre.* https://lingualibre.org/wiki/LinguaLibre:About/de. – Online; accessed 03-March-2024

[Han04] HANKE, Thomas: HamNoSys-representing sign language data in language resources and language processing contexts. In: *LREC* Bd. 4, 2004, S. 1–6

[HKRS07] HITZLER, Pascal ; KRÖTZSCH, Markus ; RUDOLPH, Sebastian ; SURE, York: *Semantic Web: Grundlagen.* Springer-Verlag, 2007. http://dx.doi.org/10.1007/978-3-540-33994-6. http://dx.doi.org/10.1007/978-3-540-33994-6

[Hop08] HOPKINS, Jason: Choosing how to write sign language: a sociolinguistic perspective. (2008). http://dx.doi.org/10.1515/IJSL.2008.036. – DOI 10.1515/IJSL.2008.036

[Kem14] KEMMERY, Megan A.: *Are you deaf or hard of hearing? Which do you go by: perceptions of students with hearing loss*, dissertation, 2014

[Lub20a] LUBER, Stefan: *Was ist das Client-Server Modell?* https://www.ip-insider.de/. Version: 2020. – Online; accessed 15-August-2023

[Lub20b] LUBER, Stefan: *Was ist das Peer-to-Peer Modell?* https://www.ip-insider.de/. Version: 2020. – Online; accessed 15-August-2023

[Man06] MANGAN, Rick: Signs their Glosses. In: *Bellevue College* (2006). https://www2.bellevuecollege.edu/artshum/materials/lang/Mangan/Winter2006/103/GlossExplained.pdf. – Online; accessed 01-September-2023

[Mar00] Martin, Joe: A linguistic comparison. In: *Western Washington University* (2000). https://www.signwriting.org/archive/docs1/sw0032-Stokoe-Sutton.pdf. – Online; accessed 03-March-2024

[Mei21] Meiller, Dieter: *App-Entwicklung mit Dart und Flutter 2: Eine umfassende Einführung*. Walter de Gruyter GmbH & Co KG, 2021. http://dx.doi.org/10.1515/9783110753080. http://dx.doi.org/10.1515/9783110753080

[Mic] *Xamarin*. https://dotnet.microsoft.com/en-us/apps/xamarin. – Online; accessed 03-March-2024

[ONK] *Philosophie Fakultät Ontologie*. https://www.philosophie.uni-muenchen.de/fakultaet/schwerpunkte/ontologie/index.html. – Online; accessed 25-May-2023

[Pri12] Priest, Lorna A.: Towards a Unicode encoding for Stokoe Notation. In: *ScriptSource* (2012). http://unicode.org/L2/L2012/12133-toward-stokoe.pdf. – Online; accessed 03-March-2024

[PS08] Prud'hommeaux, Eric ; Seaborne, Andy: *SPARQL query language for RDF*. https://www.w3.org/TR/rdf-sparql-query. Version: 2008. – Online; accessed 30-May-2023

[Pyt] *Flask*. https://flask.palletsprojects.com/en/latest/quickstart/. – Online; accessed 03-March-2024

[Sch] Schmaling, Halima C.: *Hausa Sign Language*. https://hausa-sign-language.com. – Online; accessed 25-May-2023

[Sch15] Schmaling, Constanze H.: 14 Hausa Sign Language. In: *Sign languages of the world: A comparative handbook* (2015), S. 361

[Sig17] *SignDict*. https://signdict.org/. Version: 2017. – Online;accessed 18-September-2023

[SPA] *List of sign languages with count of the number of lexemes*. https://query.wikidata.org/index.html#%0A%23%20List%20of%20languages%20with%20count%20of%20the%20number%20of%20lexemes%0ASELECT%0A%20%20%3Fnumber_of_lexemes%0A%20%20%3Flanguage%20%3FlanguageLabel%20%3FlanguageDescription%0A%20%20%3Flabel_in_sign_language%0AWITH%20%7B%0A%20%20SELECT%20%3Flanguage%20%28COUNT%28%2a%29%20AS%20%3Fnumber_of_lexemes%29%20WHERE%20%7B%0A%20%20%20%20%5B%5D%20dct%3Alanguage%20%3Flanguage%20.%0A%20%20%20%20%3Flanguage%20wdt%3AP31%20wd%3AQ34228%3B%0A%20%20%7D%0A%20%20GROUP%20BY%20%3Flanguage%0A%7D%20AS%20%25languages%0AWHERE%20%7B%0A%20%20INCLUDE%20%25languages%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%2Cen%22.%20%7D%0A%7D%0AORDER%20BY%20DESC%28%3Fnumber_of_lexemes%29%0A%20%20. – Online; accessed 03-March-2024

[Sto65] Stokoe, William C.: *A dictionary of American sign language on linguistic principles.* Gallaudet Press, 1965

[Sut] *Sign Writing.* `https://www.signwriting.org/`. – Online; accessed 04-June-2023

[UN] *Sign languages unite us!* `https://www.un.org/en/observances/sign-languages-day`. – Online; accessed 25-May-2023

[Var20] *Varnish Cache: Performance-Boost für dynamische Webprojekte.* `https://www.ionos.de/digitalguide/hosting/hosting-technik/varnish-cache-alle-fakten-zum-web-beschleuniger/`. Version: 2020. – Online; accessed 08-February-2024

[Wik] *Wikidata.* `https://www.wikidata.org/wiki`. – Online; accessed 03-March-2024

[ZD09] Zheng, Robert Z. ; Dahl, Laura B.: *Human Performance and Instructional Technologie.* SCOPUS, 2009. `http://dx.doi.org/10.4018/978-1-60960-503-2.ch716`. `http://dx.doi.org/10.4018/978-1-60960-503-2.ch716`