The Kosh Suite: A Framework for Searching and Retrieving Lexical Data Using APIs

Francisco Mondaca[1], Philip Schildkamp[1], Felix Rau[2], Luke Günther[1]

[1] Cologne Center for eHumanities, University of Cologne
[2] Data Center for the Humanities, University of Cologne
E-mail: f.mondaca@uni-koeln.de, philip.schildkamp@uni-koeln.de, f.rau@uni-koeln.de, luke.guenther@uni-koeln.de

Abstract

This paper presents the Kosh Suite, an API-centric framework designed to efficiently manage and access lexical data. The Kosh Suite aims to address the challenges in working with XML and lexical data, providing a flexible and customizable solution. The Kosh Suite architecture features a backend powered by Elasticsearch, which forms the foundation for efficient data management and retrieval. This backend offers two APIs per dataset for accessing the lexical data - a REST API and a GraphQL API per dataset. In addition, the Kosh Suite includes a frontend implemented in form of a React-based user interface, ensuring a user-friendly experience and adaptability to various use cases. Deployment specifications are described for the backend, with reference implementations for FreeDict and Cologne Sanskrit Dictionaries (CDSD). Future enhancements include asynchronous request handling using FastAPI, integration with CSV files, and leveraging advancements in large language models (LLMs). These improvements have the potential to significantly enhance the system's performance and accessibility, promoting the integration of underrepresented languages into mainstream LLMs.

Keywords: api;rest;graphql;xml;elastic

1. Introduction

1.1 An API-Centric Framework for Lexical Data

Application Programming Interfaces (APIs) play a crucial role in enabling efficient data exchange and remote functionality invocation among distributed applications. APIs, through their well-documented, stable, and user-friendly services, present a sustainable alternative to both monolithic web applications, which frequently pose maintenance challenges, and data repositories, which necessitate computational processing for effective use. According to Amundsen (2020) APIs reduce computational time and cost, facilitate easier computations, and tackle previously unresolved issues. Importantly, APIs are intended not only for application integration but also for human interaction. As such, recognizing the target consumer and use case is of utmost importance. APIs accommodate a wide array of devices and software stacks, including Java-enabled smartphone applications and Python-based desktop programs. As a result, development efforts concentrate on the data output and functionality of the API. This emphasis indirectly boosts the sustainability of the underlying system supporting the API, as the computations take place within this system rather than prioritizing data presentation. In case of system errors, all consumers are affected, highlighting the necessity for a robust and dependable API infrastructure.

As of Kosh's[1] initial development (Mondaca et al., 2019b), and persisting until now, no frameworks have been exclusively dedicated to APIs for lexical data. Lexical data, with its inherent adaptability, can be applied in single-page dictionary applications, incorporated into corpora, and utilized in diverse NLP tasks. While alternative tools, such as Lexonomy (Měchura, 2017), permit users to create and publish dictionaries that can be integrated into the CLARIN network or have their data downloaded in XML format, these tools lack web API offerings. Despite the growing importance and application of APIs in the industry (Medjaoui et al., 2021), such focus has not yet been reflected in the academic advancements within the field of lexicography.

1.2 Background on XML and Lexical Data

XML (eXtensible Markup Language) is a popular serialization format for lexical data. The hierarchical and structured format of XML allows lexicographers to represent complex linguistic data in a clear and organized way, thus facilitating easier processing, query, and share between different systems and applications. In addition, the standardization of XML as a widely adopted format for data serialization lead to many tools and libraries being available for parsing and processing XML data, making it a reliable and well-supported choice for lexicographers. The extensibility of XML also makes it a beneficial choice for digital lexicography, as it enables lexicographers to customize lexical data structures to meet the specific needs of a given project. This allows for the creation of specialized lexicons and dictionaries that cater to specific domains or user groups, and can be used in a variety of contexts, such as natural language processing and machine translation. The development of the TEI (Text Enconding Initiative) has been a relevant endeavor in the digital humanities, as it provides a flexible model that can cover multiple needs of different communities working with digital data. However, the flexibility of the TEI Guidelines (TEI Consortium, 2023) for encoding dictionaries often results in inconsistent encoding, which hinders the processing, analysis, and sharing of lexical data across systems and applications. To address this challenge, TEI Lex-0 (Tasovac et al., 2018) was created to establish a standardized and interoperable format for encoding lexical data within a community of practice. TEI Lex-0 offers a baseline encoding and target format that ensures the interoperability of heterogeneously encoded lexical resources, providing a lightweight and standardized format for creating structured and machine-readable lexical resources. This makes it easy for lexicographers to use and adopt while still adhering to a consistent model for searching, visualizing, or enriching multiple lexical resources. The use of TEI Lex-0 enables the building and management of large-scale lexical infrastructures by facilitating the creation of high-quality and interoperable lexical resources that can be easily processed, analyzed, and shared across different systems and applications. XML-encoded lexical datasets are commonly used due to their flexibility, hierarchical structure, cross-platform compatibility, standardization, and extensibility. While TEI-encoded dictionaries are a relevant part of the digital lexicography scholarly landscape, there are also many other types of XML-encoded dictionaries used in academia and industrial contexts. These dictionaries can have a wide range of structures, and searchable fields can vary greatly between them. By initially focusing solely on XML as an input format, we could concentrate on designing a straightforward and efficient framework. Although data in other formats must be converted into XML to be used by Kosh, multiple tools are available for accomplishing this transformation.

[1] https://kosh.uni-koeln.de

1.3 Motivation for the Kosh Suite

Since its inception in 2019, Kosh has been employed in various research and community projects, including the Cologne Digital Sanskrit Dictionaries (2019), VedaWeb (Kiss et al., 2019), Zoroastrian Middle Persian Corpus and Dictionary (MPCD) (Mondaca et al., 2022), and FreeDict[2]. The VedaWeb project was the first to employ Kosh, linking every token of the RigVeda to a lemma of Grassman's dictionary (Grassmann, 1873) through a Kosh API and its corresponding information (Mondaca et al., 2019a). The primary impetus behind Kosh's development was to foster the decentralization and increased utilization of lexical data pertaining to datasets that are typically underrepresented in the digital realm for various reasons. Although APIs enable developers and researchers to craft bespoke applications that harness these APIs, it has been observed that numerous scholars possessing XML-encoded lexical data often lack the necessary resources to devise user-centric client applications tailored for data exploration and retrieval. To address this issue, we have developed a client application as part of the Kosh Suite, which consumes and visualizes data provided by the Kosh APIs, thus enabling users to search through the data. By providing a user-friendly interface for accessing data, we aim to make consuming and serving XML-encoded lexical data more accessible to researchers and scholars who may not have the resources or expertise to develop their own applications for using the Kosh APIs.

2. Architecture

2.1 Overview

The Kosh Suite is a comprehensive software framework designed to manage and access lexical data in XML format. The structured nature of XML, facilitated by the use of tags, allows for easy identification and navigation of different elements of the data. The framework relies on Elasticsearch, a search engine that can be used to index lexical data, making it easily searchable. Kosh provides two APIs per dataset: a REST (Representational State Transfer) API (Fielding, 2000) and a GraphQL API (GraphQL, 2021). The REST API allows for read operations on the indexed data and is easy to use with a wide range of programming languages and frameworks. The GraphQL API allows clients to request exactly the data they need and retrieve multiple resources in a single request. Using these APIs, the Kosh frontend offers a user-friendly interface for searching and filtering the indexed data. The Kosh Suite's frontend is developed using React[3] and Tailwind CSS[4], offering a web-based user interface to search through the lexical data provided by the Kosh APIs. Users have the ability to perform complex searches, filter results based on various criteria, and view detailed information about the lexical entries. Additionally, users have the option to configure the fields they wish to search on through a JSON file, ensuring that the indexed data remains easily searchable.

[2] https://freedict.org/
[3] https://react.dev
[4] https://tailwindcss.com

2.2 Backend

2.2.1 Elasticsearch

Elasticsearch[5] is an open-source distributed search engine that is specifically designed to handle large datasets with high performance and scalability. It offers advanced features such as full-text search, faceting, and geospatial search, which make it a popular choice for indexing and searching large datasets. Elasticsearch is built on top of Apache Lucene[6], a powerful and widely used search library that provides advanced search capabilities. The Elasticsearch platform is broadly used in various applications, including digital lexicography, due to its ability to handle large datasets in near real-time. The full-text search capabilities of Elasticsearch enable users to search for relevant data using natural language queries, making it an ideal choice for indexing and searching textual data. In the context of the Kosh Suite, Elasticsearch serves as the backend for indexing and searching lexical data in XML format. The system's ability to index large amounts of data quickly is crucial for digital lexicography, which often deals with massive datasets. Elasticsearch's full-text search capabilities and real-time search enable complex searches and filtering based on multiple criteria. Additionally, Elasticsearch's distributed architecture facilitates easy scaling of the system, enabling high availability and fault tolerance. The system can be deployed across multiple nodes, providing redundancy and load balancing, making it suitable for large-scale projects. Elasticsearch also offers APIs, making it easy to integrate with other systems and applications, enhancing its flexibility and versatility.

2.2.2 REST API

A REST API is an interfacing standard for creating web services that enables communication between different systems. It follows a client-server architecture and allows for read and write operations on the indexed data, making it easy to use with a wide range of programming languages and frameworks. In the context of the Kosh Suite, the REST API enables read-only access to the indexed lexical data in XML format. Kosh only accepts HTTP GET requests on the REST API, as it is used for reading the indexed data. Users can perform complex searches and filter results based on various criteria using the REST API. It is designed to be easily integrated with other systems and applications, further enhancing its versatility. OpenAPI[7] provides a user-friendly interface for developers to interact with the Kosh APIs. It offers a visual representation of the API's endpoints and parameters, making it easier to understand how to use the API. Additionally, OpenAPI offers interactive documentation, allowing developers to test the API's endpoints and view the results in real-time. This feature saves time and improves efficiency as it eliminates the need to manually test each endpoint using external tools.

2.2.3 GraphQL API

GraphQL is a query language that provides a more flexible approach to retrieving data compared to traditional REST APIs. One of the key benefits of GraphQL for managing lexical data in Kosh is its ability to allow clients to request only the data they need. This

[5] https://www.elastic.co/elasticsearch/
[6] https://lucene.apache.org
[7] https://www.openapis.org

means that clients can retrieve precisely the data they require without being limited by the constraints of a fixed data structure. As a result, GraphQL provides greater flexibility and reduces the amount of data that needs to be transferred over the network, which can improve the efficiency of data retrieval. In the context of digital lexicography, this flexibility is particularly valuable, as it allows lexicographers to create more complex data structures without worrying about the impact on data retrieval performance. This can enable the development of more advanced and customizable search interfaces for lexical data. Additionally, the ability to perform nested queries enables clients to retrieve related data in a single request, reducing the number of requests required to access all the necessary data. The use of GraphQL in Kosh provides a powerful and flexible tool for managing and retrieving lexical data.

2.3 Frontend

2.3.1 React-based User Interface

React is a widely-used and popular JavaScript library for building user interfaces (UIs). Its component-based architecture enables high modularity and reusability in building UIs, while its virtual DOM feature provides performance advantages by selectively updating only the modified parts of the UI. Moreover, React boasts a vast ecosystem of libraries and tools that facilitates rapid development of complex applications. When combined with a GraphQL API, React offers various advantages that can enhance the performance of web applications that manage large datasets, such as lexical data. By allowing clients to request only the necessary data, GraphQL minimizes the volume of redundant data transferred over the network, leading to faster data retrieval and improved application performance. GraphQL's capability to execute nested queries simplifies data fetching, reducing the number of requests necessary to retrieve data. These benefits are especially relevant when the network data is limited, as GraphQL streamlines data retrieval, resulting in more efficient resource utilization, faster data loading times, and improved application responsiveness. React's component-based structure also promotes code efficiency and reusability, supporting the development of scalable and efficient applications.

2.3.2 Tailwind CSS for UI-Customization

Tailwind CSS[8] is a utility-first framework that offers more customization and flexibility than alternatives like Bootstrap[9] or Foundation[10]. Its low-level utility classes can be composed to create unique designs, and it offers a vast collection of predefined classes for easy modification. It also provides responsive design classes for optimization across screen sizes and devices. In the Kosh Suite, Tailwind is used for frontend development, offering advantages such as easy customization, and responsive design options. This ensures a user-friendly interface that remains accessible and usable across devices and platforms while giving developers greater control over design and layout.

[8] https://tailwindcss.com
[9] https://getbootstrap.com
[10] https://get.foundation

3. Deployment

3.1 Backend

Kosh is designed to offer minimal prerequisites and an uncomplicated setup. It can be deployed on Linux systems or through Docker. For deployment, Kosh requires a Kosh dotfile, a JSON file with mappings, and it processes files in XML format.

3.1.1 Kosh Dotfile

This file provides details on: (i) the name of the dataset's index; (ii) the location of XML files containing lexical information; (iii) the location of the configuration JSON file, utilized for parsing and configuring Elasticsearch; (iv) the title of the dataset; (v) any other, additional metadata, that should be made available through the Kosh API.

3.1.2 JSON Mappings

The JSON file referenced from within a Kosh dotfile contains information about XML nodes and their subnodes, specified in XPath 1.0 notation, which is used for indexing. It also includes details about handling different data types, such as "keyword" for unprocessed strings and "text" for preprocessed strings analyzed by Elasticsearch. Additionally, the file provides instructions on handling arrays of elements and automatically generating entry IDs if not present in the dictionary. Lastly, it outlines the default indexing behavior, which involves indexing the entire entry without analyzing XML tags.

3.1.3 Endpoints

Kosh offers a REST and a GraphQL API for each dataset indexed, and it also indicates in JSON which datasets are accessible for each Kosh instance. Each dataset comprises: (i) information about the queryable fields, such as "id", "lemma", and "sense"; (ii) available Elasticsearch query types, like "wildcard" and "prefix"; (iii) the number of entries available. This endpoint information holds computational significance, as it can be utilized by other applications, including the Kosh Suite frontend. For instance, detailed information about each dataset is accessible at https://kosh.uni-koeln.de/api for all datasets deployed under a specific Kosh instance. Similarly, individual datasets also contain this information, as seen in https://kosh.uni-koeln.de/api/de_alcedo.

3.2 Frontend

Analogous to the backend, the frontend is deployed with a Docker container. As illustrated in Section 3.1.3 Endpoints, the frontend capitalizes on the information furnished by Kosh's backend to dynamically generate user interface components for the purpose of querying and exhibiting data provided by the backend. This distinctive attribute permits Kosh to accommodate a diverse range of datasets while simultaneously affording users the flexibility to establish naming conventions, as they possess the freedom to determine field names in the JSON file delivered to the backend.

4. Reference Implementations

4.1 FreeDict

4.1.1 About FreeDict

The FreeDict project aims to serve as the preeminent repository for free bilingual dictionaries. These resources not only come at no cost but also confer the rights to examine, modify, and adapt them, provided that users extend these liberties to others. Established in 2000, FreeDict currently offers a compendium of more than 200 multilingual dictionaries, spanning approximately 45 languages, with its continuous growth thanks to the contributions of its members.

4.1.2 FreeDict Implementation

Freedict hosts its hand-written dictionaries in TEI format on GitHub, while also provides a comprehensive list of all its dictionaries in JSON format. There is a repository on GitHub that generates the necessary data for Kosh, including Kosh dotfiles and JSON files, to enable deployment[11]. The implementation of this repository facilitates continuous monitoring of updates to the database, thereby ensuring synchronization between both Kosh and the FreeDicts. Kosh APIs generated for FreeDict are available at: https://kosh.uni-koeln.de/freedict

4.2 Cologne Digital Sanskrit Dictionaries (CDSD)

4.2.1 About the CDSD

The Cologne Digital Sanskrit Dictionaries (CDSD) project began with the efforts of Thomas Malten from the University of Cologne in 1994, initially focusing on digitizing the Monier-Williams Sanskrit-English Dictionary (Monier-Williams, 1899). As of now, the project boasts a collection of 38 dictionaries. The CDSD portal relies on data hosted on GitHub[12], where a diverse team of scholars and users from around the world work together to maintain and improve the available information.

4.2.2 CDSD Implementation

The CDSD project initially utilized a unique markup language for encoding. Presently, the dictionaries on the CDSD are derived from XML files that have been transformed using various scripts, available on a GitHub repository[13]. We employ these repositories in conjunction with a third one[14] to generate the required JSON and Kosh dotfiles essential for deploying these dictionaries with Kosh. Access to the Kosh CDSD APIs is available at: https://kosh.uni-koeln.de/cdsd

[11] https://github.com/freedict/fd-kosh
[12] https://github.com/sanskrit-lexicon/csl-orig/tree/master/v02
[13] https://github.com/sanskrit-lexicon/csl-pywork
[14] https://github.com/cceh/csl-kosh

5. Exploring Search Queries

In this section, we provide a variety of search examples that can be used with Kosh, illustrating both GraphQL and REST queries. For GraphQL, the necessary parameters to be inputted by the user for effective interaction with the GraphiQL interface are explicitly provided.

1. Term: The term query finds documents that contain the exact term specified in the field specified. Examples:
   - RESTful: `https://kosh.uni-koeln.de/api/de_alcedo/restful/entries?field=lemma&query=santiago&query_type=term&size=20`
   - GraphQL: `https://kosh.uni-koeln.de/api/de_alcedo/graphql`

```
{
  entries(queryType: term,
  query: "santiago",
  field: lemma,
  size: 20) {
    lemma
    sense
  }
}
```

2. Fuzzy: The fuzzy query generates all possible matching terms that are within a certain maximum edit distance, allowing for variations in the terms. Examples:
   - RESTful: `https://kosh.uni-koeln.de/api/de_alcedo/restful/entries?field=lemma&query=ica&query_type=fuzzy&size=50`
   - GraphQL: `https://kosh.uni-koeln.de/api/de_alcedo/graphql`

```
{
  entries(queryType: fuzzy,
  query: "ica",
  field: lemma,
  size: 50) {
    lemma
    sense
  }
}
```

3. Match: The match query is a standard query that is useful for single word and phrase queries. Examples:
   - RESTful: `https://kosh.uni-koeln.de/api/ducange/restful/entries?field=xml&query=viispublicis&query_type=match&size=30`
   - GraphQL: `https://kosh.uni-koeln.de/api/ducange/graphql`

```
{
  entries(queryType: match,
  query: "viis publicis",
  field: xml,
  size: 30) {
    lemma
    xml
  }
}
```

4. Match Phrase: The match phrase query is like match, but it only returns documents where the matched words are in the order specified in the query. Examples:
   - RESTful: https://kosh.uni-koeln.de/api/de_alcedo/restful/entries?field=sense&query=reynodechile&query_type=match_phrase&size=300
   - GraphQL: https://kosh.uni-koeln.de/api/de_alcedo/graphql

```
{
  entries(queryType: match_phrase,
  query: "reyno de chile",
  field: sense,
  size: 300) {
    lemma
    sense
  }
}
```

5. Prefix: The prefix query matches documents where the value of the specified field begins with that prefix. Examples:
   - RESTful: https://kosh.uni-koeln.de/api/hoenig/restful/entries?field=lemma_ksh&query=Hau&query_type=prefix&size=20
   - GraphQL: https://kosh.uni-koeln.de/api/hoenig/graphql

```
{
  entries(queryType: prefix,
  query: "Hau",
  field: lemma_ksh,
  size: 20) {
    lemmaKsh
    translationDeu
  }
}
```

6. Wildcard: The wildcard query matches documents where the specified field matches a wildcard expression. Examples:
   - RESTful: https://kosh.uni-koeln.de/api/ducange/restful/entries?field=lemma&query=e*en&query_type=wildcard&size=50
   - GraphQL: https://kosh.uni-koeln.de/api/ducange/graphql

```
{
  entries(queryType: wildcard,
  query: "e*en"
  field: lemma,
  size: 50) {
    lemma
  }
}
```

7. Regexp: The regexp query matches documents where the specified field matches a regular expression. Examples:
   - RESTful: https://kosh.uni-koeln.de/api/tunico/restful/entries?field=lemma&query=d.*m&query_type=regexp&size=50
   - GraphQL: https://kosh.uni-koeln.de/api/tunico/graphql

```
{
  entries(queryType: regexp,
  query: "d.*m",
  field: lemma,
  size: 50) {
    lemma
    transEn
  }
}
```

6. Future Directions and Enhancements

6.1 Backend

6.1.1 Asynchronous Request Handling

Our objective is to improve Kosh by integrating asynchronous capabilities. To accomplish this, we will migrate from the existing web framework, Flask[15], to FastAPI[16]. FastAPI is an asynchronous web framework that delivers a notable performance boost. It inherently supports REST and accommodates GraphQL through the Strawberry[17] library. Moreover, we intend to introduce asynchronous queries in Elasticsearch and implement nested fields. The nested field type, a specialized version of the object data type, enables the indexing of object arrays in a manner that allows them to be queried separately from one another. These changes are anticipated to significantly enhance the system's overall performance and usability.

6.1.2 Integration with CSV Files

Drawing upon our expertise in the Zoroastrian Middle Persian Corpus and Dictionary (MPCD) project, as well as collaborations with other scholars, it has been observed that numerous researchers maintain their lexical data utilizing CSV files. While it is feasible to convert this data into XML format, this task imposes additional workload. Consequently, we endeavor to develop a methodology for directly incorporating data from spreadsheets into Kosh.

6.1.3 Integration with Large Language Models

Recent advancements in the field of large language models (LLMs) present a promising landscape for APIs handling natural language processing data in the upcoming future. The

[15] https://flask.palletsprojects.com/en/2.2.x
[16] https://fastapi.tiangolo.com
[17] https://strawberry.rocks/docs/integrations/fastapi

progress in this domain is unparalleled, given the rapidity and depth of the transformations witnessed in recent months. Toolformer (Schick et al., 2023), a model designed to determine which APIs to invoke, when to initiate them, which arguments to transmit, and how to optimally integrate the outcomes into subsequent token predictions, operates in a self-supervised manner, necessitating only a few demonstrations for each API. The researchers trained a GPT-J model akin to GPT-3, albeit with significantly fewer parameters—6B compared to 175B—and achieved comparable results to GPT-3 across various benchmarks. Furthermore, OpenAI is incorporating external APIs into Chat-GPT, adhering to the methodology delineated in Toolformer, and accessing external APIs through plugins. Although plugin access remains in limited beta at the time of writing, the initial draft outlining the creation of a Chat-GPT plugin has been released. The prevailing specification employs Open-API, or Swagger, which is also utilized by Kosh for their REST APIs. While this specification may evolve and encompass GraphQL in the future, it is likely to remain the standard employed by Chat-GPT and other LLMs. This development is highly propitious for the evolution of Kosh and the integration of knowledge from underrepresented languages into mainstream LLMs.

7. References

Amundsen, M. (2020). *Design and build great web APIs: robust, reliable, and resilient*. The pragmatic programmers. Raleigh, North Carolina: The Pragmatic Bookshelf.

Cologne Digital Sanskrit Dictionaries (2019). Version 2.4.79. Cologne University. Accessed on April 13, 2023. https://www.sanskrit-lexicon.uni-koeln.de.

Fielding, R.T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Ph.D. thesis, University of California, Irvine. URL https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf.

GraphQL (2021). GraphQL Specification. https://spec.graphql.org. Accessed: April 18, 2023.

Grassmann, H.G. (1873). *Worterbuch zum Rig-veda*. Wiesbaden: O. Harrassowitz. OCLC: 184798352.

Kiss, B., Kölligan, D., Mondaca, F., Neuefeind, C., Reinöhl, U. & Sahle, P. (2019). It Takes a Village: Co-developing VedaWeb, a Digital Research Platform for Old Indo-Aryan Texts. In S. Krauwer & D. Fišer (eds.) *TwinTalks at DHN 2019 – Understanding Collaboration in Digital Humanities*, volume 2365 of *CEUR Workshop Proceedings*. URL http://ceur-ws.org/Vol-2365/05-TwinTalks-DHN2019_paper_5.pdf.

Medjaoui, M., Wilde, E., Mitra, R. & Amundsen, M. (2021). *Continuous API Management*. O'Reilly Media, Inc., 2 edition. ISBN: 9781098103521.

Mondaca, F., Esser, M., Neuefeind, C. & Eide, Ø. (2022). MPCD: An API-based Research Environment. URL https://doi.org/10.5281/zenodo.7839927.

Mondaca, F., Rau, F., Neuefeind, C., Kiss, B., Kölligan, D., Reinöhl, U. & Sahle, P. (2019a). C-SALT APIs – Connecting and Exposing Heterogeneous Language Resources. URL https://doi.org/10.5281/zenodo.3265782.

Mondaca, F., Schildkamp, P. & Rau, F. (2019b). Introducing Kosh, a Framework for Creating and Maintaining APIs for Lexical Data. In *Electronic Lexicography in the 21st Century. Proceedings of the eLex 2019 Conference, Sintra, Portugal*. Brno: Lexical Computing CZ, pp. 907–21. URL https://elex.link/elex2019/wp-content/uploads/2019/09/eLex_2019_51.pdf.

Monier-Williams, M. (1899). *A Sanskrit-English dictionary: Etymologically and philologically arranged with special reference to Cognate indo-european languages.* Oxford: The Clarendon Press.

Měchura, M. (2017). Introducing Lexonomy: an open-source dictionary writing and publishing system. In *Electronic Lexicography in the 21st Century. Proceedings of the eLex 2017 Conference, Leiden, Netherlands.* Leiden: Lexical Computing CZ, p. 18. URL https://elex.link/elex2017/wp-content/uploads/2017/09/paper41.pdf.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N. & Scialom, T. (2023). Toolformer: Language Models Can Teach Themselves to Use Tools. URL http://arxiv.org/abs/2302.04761. ArXiv:2302.04761 [cs].

Tasovac, T., Romary, L., Banski, P., Bowers, J., de Does, J., Depuydt, K., Erjavec, T., Geyken, A., Herold, A., Hildenbrandt, V., Khemakhem, M., Lehečka, B., Petrović, S., Salgado, A. & Witt, A. (2018). TEI Lex-0: A baseline encoding for lexicographic data. https://dariah-eric.github.io/lexicalresources/pages/TEILex0/TEILex0.html.

TEI Consortium (2023). TEI P5: Guidelines for Electronic Text Encoding and Interchange. http://www.tei-c.org/Guidelines/P5/. Accessed on 18.04.2023.