UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

GABRIEL MADEIRA

A Visualization Tool for the Exploration of Knowledge Graphs

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Science

Advisor: Prof. Dr. Joao Luiz Dihl Comba
Coadvisor: Dr. Heiko Maus

Porto Alegre
August 2024

ABSTRACT

Knowledge graphs (KGs) offer a powerful structure to organize and connect information, providing structured data for computational analysis and insights for the user through exploration. However, visualizing KGs presents significant challenges due to the huge volume of nodes and relationships. This work addresses these challenges by introducing a tool designed to visualize and explore large and generic KGs. It bridges the gap between the current node-link visualization approaches and the non-expert user within large KGs. The tool incorporates functionalities to support interactive exploration, filtering, and grouping. Its effectiveness is demonstrated by implementing three real-world KGs: DBpedia and two corporate KGs. A user evaluation with ten participants was performed in the DBpedia environment. In addition to user feedback, in one of the two formal evaluation tasks, the tool proved to be more efficient. Finally, it was also possible to achieve a 79.5 score on the System Usability Scale method.

Keywords: Knowledge graph. Interactive visual exploration. User-centered design.

Uma ferramenta de visualização para a exploração de grafos de conhecimento

RESUMO

Grafos de conhecimento (KGs) oferecem uma estrutura poderosa para organizar e conectar informações, fornecendo não apenas dados estruturados para análise computacional, mas também insights para o usuário por meio de sua exploração. No entanto, a visualização de KGs apresenta desafios significativos devido ao enorme volume de nós e relacionamentos. Este trabalho aborda esses desafios introduzindo uma ferramenta projetada para visualizar e explorar KGs grandes e genéricos. Ele contribui preenchendo a lacuna entre as abordagens atuais de visualização node-link e o usuário não especialista. A ferramenta incorpora funcionalidades para dar suporte à exploração interativa, filtragem e agrupamento. Sua eficácia é demonstrada através de sua implementação em três KGs do mundo real: DBpedia e dois KGs corporativos. Uma avaliação de usuário com dez participantes foi realizada no ambiente do DBPedia. Além de feedbacks dos usuários, em uma das duas tarefas formais de avaliação, a ferramenta provou ser mais eficiente. Por fim, também foi possível atingir uma pontuação de 79,5 no método System Usability Scale.

Palavras-chave: Grafos de conhecimento. Exploração visual interativa. Design centrado no usuário.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

KG	Knowledge Graph
KGE	Knowledge Graph Explorer
UI	User Interface
LOD	Linked Open Data
IRI	Internationalized Resource Identifier
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
API	Application Programming Interface
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema

CONTENTS

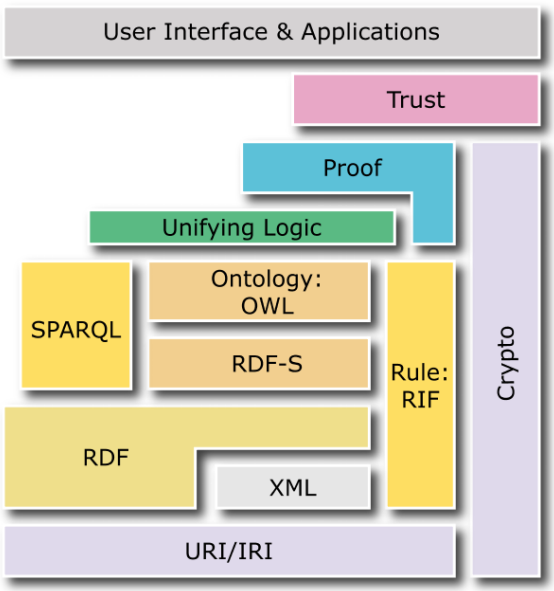
1 INTRODUCTION

A huge amount of data has been generated every day since the beginning of the internet. Yet, many of those are unstructured and unconnected in isolated databases. For users and applications to be able to iterate, analyze, and extract useful knowledge, it is essential to have it structured. There is a long history of research into knowledge representation, and with the growth of the internet, the concept of the semantic web emerged, bringing standards to structure the data (LI et al., 2023). The Semantic Web (LASSILA; HENDLER; BERNERS-LEE, 2001) (SHADBOLT; BERNERS-LEE; HALL, 2006) was proposed back in 2001, and it seeks to extend the "classical web" with web standards to structure available web data creating machine-interpretable information, making semantic links between the documents, allowing reasoning, and useful complex queries. Its architecture is composed of many layers (Figure 1.1) where semantic technologies such as Resource Description Framework (RDF) and Web Ontology Language (OWL) support the upper-level ones like logic, proof, and trust.

Another technical foundation present among the bottom layers is the Uniform Resource Identifier (URI) which is defined as "a compact sequence of characters (ASCII) that identifies an abstract or physical resource" [1]. A Uniform Resource Locator (URL) is

[1] https://datatracker.ietf.org/doc/html/rfc3986

Figure 1.1: Semantic web stack.



Source: (BRATT, 2007)

a subset of URIs that provide formalized location and access information of a resource via the internet, while URIs refer to arbitrary things.

The RDF [2] has been a W3C standard since 2004 and can describe semantic networks as triples <subject, predicate, object>. A URI, and the properties/predicates link resources to other resources and literals identify each resource. Literals are atomic values that could have a language or a data type (*e.g.,* integer, string, dates, etc.). It can only be an object, not a subject or predicate, *i.e.,* they are nodes connected with just a central resource, representing some specific information. Resources can also have a type using a predefined property "rdf:type". The Resource Description Framework Schema (RDFS) [3] is another standard that supports building ontologies on top of RDF. With properties like "rdfs:Class", "rdfs:subClassOf", "rdfs:subPropertyOf", "rdfs:domain", it is possible to define an ontology and derive new facts through deductive reasoning.

The SPARQL Query Language for RDF [4] can be used to perform queries on an RDF network. It is another standard from W3C and similar to SQL queries. The following example shows a query to return a person who knows the father of a second person. The "otherPerson" variable shows how the use of recurring variables can be used to define complex patterns.

```
SELECT ?person1 ?person2
WHERE { ?person1 :knows ?otherPerson
?otherPerson :fatherOf ?person2. }
```

The Linked Data is the result of a practical implementation of the Semantic Web where Tim Bernes-Lee introduced four principles [5]: use URIs to identify things; use HTTP URIs so that people can look up those names; provide useful information upon URIs, using standards like RDF; include links to other datasets. The freely available set of this data is called Linked Open Data (LOD). LOD Cloud [6] counts with billions of triples and millions of interlinks with a broad range of domains such as government, publications, and social web.

The context opened the way for introducing the knowledge graph (KG) data structure, allowing it to represent a semantic network and, therefore, human knowledge. A knowledge graph (KG) (Figure 1.2) is a graph data structure that can connect data sources, providing structured information where human knowledge can be extracted. Although no widely accepted formal definition for a KG was found, it can be defined (WANG et al.,

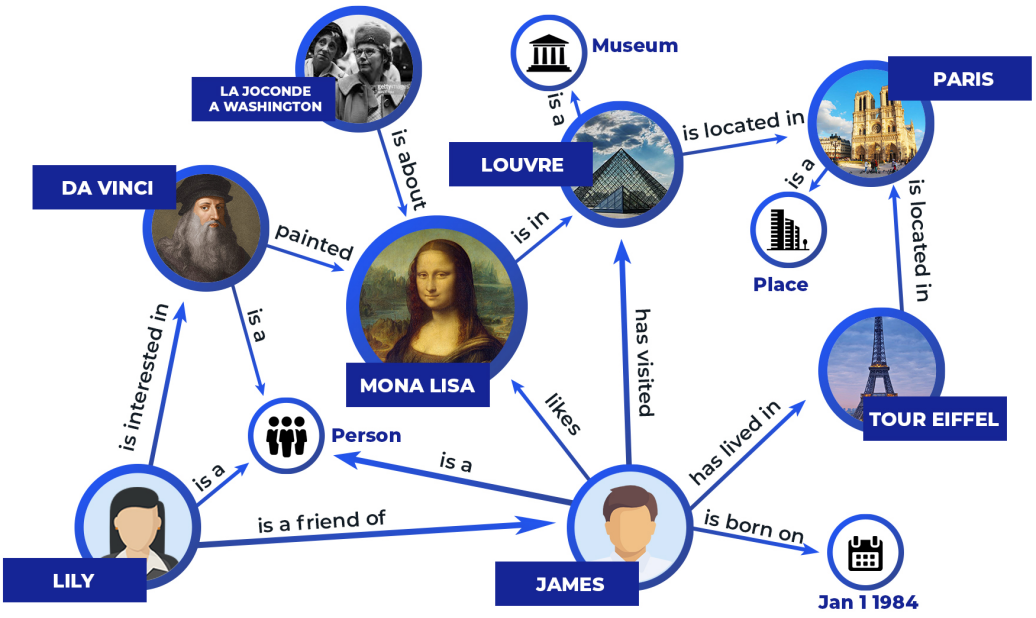
[2] https://www.w3.org/TR/rdf11-primer
[3] https://www.w3.org/TR/rdf-schema/
[4] https://www.w3.org/TR/sparql11-query
[5] https://www.w3.org/DesignIssues/LinkedData.html
[6] https://lod-cloud.net/

2017) as a multiple-relational graph composed of entities (nodes) and relations (types of edges). Each edge can be defined as a triple of <head, relation, tail>, also called a fact.

The popularity of the term knowledge graph rose when Google announced in 2012 a KG to improve its search engine. However, many large knowledge bases and ontologies have already been published, such as DBPedia, Wordnet, Yago, and Freebase. Many research topics cover KGs, such as knowledge graph representation learning, knowledge acquisition and completion, temporal knowledge graph, and knowledge aware applications (JI et al., 2021). A KG can also be described with the RDF. A KG's triple <head, relation, tail> can be translated into the RDF's triple format <subject, predicate, object>. In addition, an ontology can be created to define entity types and their semantic relationships.

Graph visualization approaches can be essential to extract KG's stored knowledge. Among many reasonable use cases, it can facilitate relationship exploration and human comprehension. Visualization designs are included among KG challenges (LI et al., 2023). One of its issues is a lack of efficacy in using node-link diagrams when users interpret and seek insights. As the number of nodes increases, it quickly becomes a "hairball". That is the reason why end users tend to prefer Wikipedia-style interfaces. However, if a node-link visualization is required, visualization tools should provide features

Figure 1.2: Knowledge graph illustration.



Source: (SETH, 2019)

to select a specific point or region of interest, filter, bundle, condense, collapse, or expand areas of the KG during exploration, and the ability to switch views while maintaining context. Additionally, node-link diagrams can be useful for transversing the KG while discovering new information. For this to work, it is necessary to balance digestibility and discoverability. Otherwise, a large knowledge graph visualization can quickly become overly complex and overwhelm the user's information processing.

However, there are challenges in the visualization of large and generic KGs. Considering the current node-link diagram solutions, there is a scalability issue where the view easily becomes overloaded with a few iteration levels (LI et al., 2023). In addition, there is limited support for organic discovery, and it is even harder to make visualizations legible to end users. A visualization KG tool with proper exploration functionalities could assist an expert and end user in exploring a generic KG.

In this work we propose a user-centric tool to explore and visualize knowledge graphs through filtering, grouping, and exploration features. It enables a non-expert user to explore and obtain knowledge from large and generic KGs visually and interactively. The solution was implemented in three environments. The first implementation was called "Wiki Knowledge Graph Explorer", using the DBPedia's KG. In the other two implementations, a "Knowledge Graph Explorer" submodule was created in already existing web platforms from two corporate KGs, CoMem and EnviaM. A user evaluation was performed with ten participants using the implementation of Wiki Knowledge Graph Explorer. In one of the two formal evaluation tasks, the tool proved to be more efficient. It was also possible to achieve a 79.5 score on the System Usability Scale (SUS) method. Finally, feedback was collected from the participants.

The text is organized as follows. Chapter 2 describes related work and the main contributions that inspired this work. The main data sources, DBPedia, CoMem, and EnviaM, are described in Chapter 3, with respective examples of their KG relationships. Chapter 4 is composed of the project development itself where first the filtering, grouping, and exploration features methodology are explained, and then the implementation architecture is described. The user evaluation methodology and its results are explained in Chapter 5. Chapter 6 concludes this manuscript by discussing the results implications and future possible improvements.

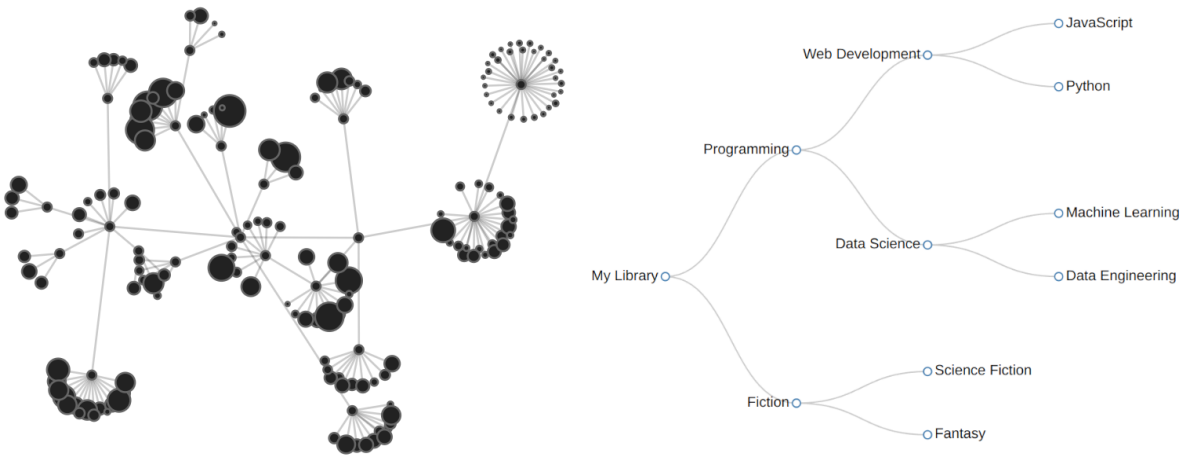
2 RELATED WORK

2.1 Graph visualization

Many datasets across diverse application areas exhibit relational characteristics (*e.g.,*, social networks, internet websites, hyperlinks, urban roadmap). Those can be represented as a graph. Graphs can provide a data structure to store and iterate very efficiently this type of information. Although using it as a backend structure is often enough, visualizing it could be useful to discover insights about its nature. Graphs can be either directed or undirected, and, among many visualization approaches, they can be visualized using node-link diagrams. Additionally, aesthetic criteria should be considered to reach more effective graph visualization (BENNETT et al., 2007) such as short and uniform edge lengths; symmetry, a uniform distribution of children; minimizing edge crossings; minimizing edges crossing nodes; and the most compact possible layout.

The Force-directed layout (Fruchterman-Reingold algorithm (FRUCHTERMAN; REINGOLD, 1991)) (Figure 2.1) can be used to visualize either directed or undirected graphs. The algorithm calculates the graph layout using physical models with forces that must be minimized. With interesting results, it is also an intuitive algorithm and easy to implement. However, it still faces challenges, as it is commonly not deterministic, and for densely connected graphs, the visualization can quickly become a “hairball”.

Figure 2.1: Force-directed layout and node-link tree layout illustrations.



Force-directed layout Node-link tree layout

Source: Author

Trees can be defined as acyclic graphs and used to represent hierarchical relations. Those can be visualized with the Node-link tree layout (Reingold-Tilford algorithm (REINGOLD; TILFORD, 1981)) (Figure 2.1). As a bottom-up iterative approach, it draws a node left and right subtrees and moves them as close as possible. The parent is positioned between the children.

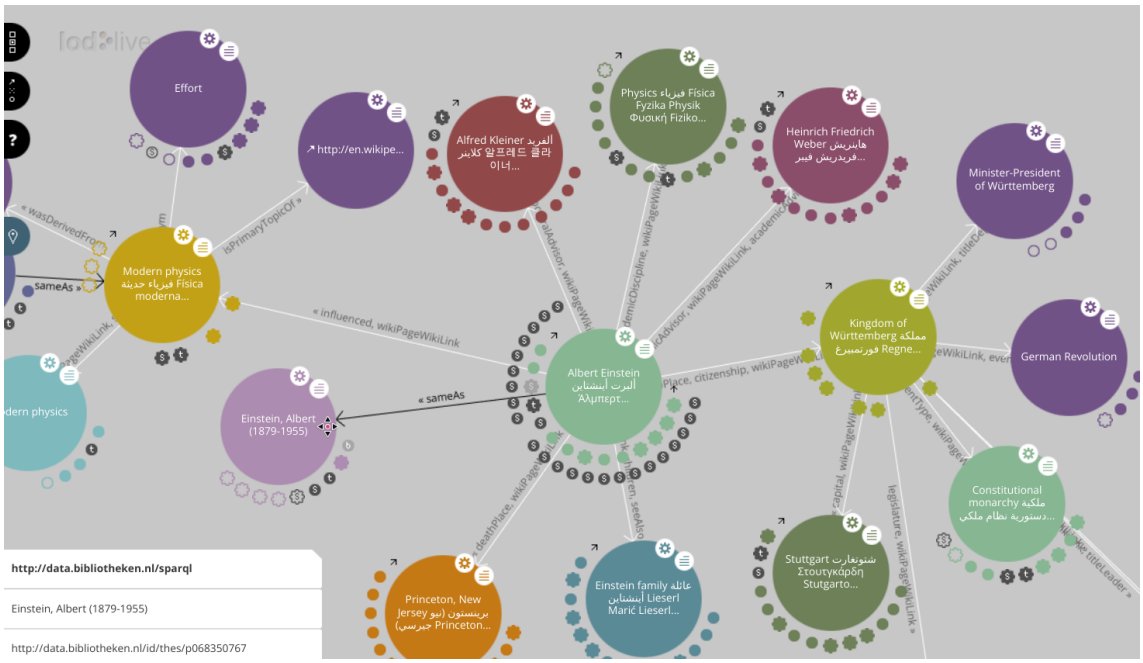
2.2 Knowledge graph visualization tools

2.2.1 Lodlive

Lodlive (CAMARDA; MAZZINI; ANTONUCCIO, 2012) (Figure 2.2) provides a visualization tool to explore Linked Open Data (LOD) datasets (*e.g.,* DBPedia, Freebase). The system aims to follow and promote LOD standards such as W3C SPARQL and RDF. It is also able to connect the resources from different configured SPARQL endpoints.

Starting from a URI node in a given dataset, the user can manually select and expand a property and entity through a ring menu. Lodlive also supports expanding entities from multiple LOD datasets endpoints at the same view (Figure 2.2).

Figure 2.2: Lodlive interface. Starting from the Albert Einstein DBPedia URI, several entities were expanded. One of them is connected with the property "sameAs" to another representation of Albert Einstein, from *data.bibliotheken.nl/sparql* endpoint.



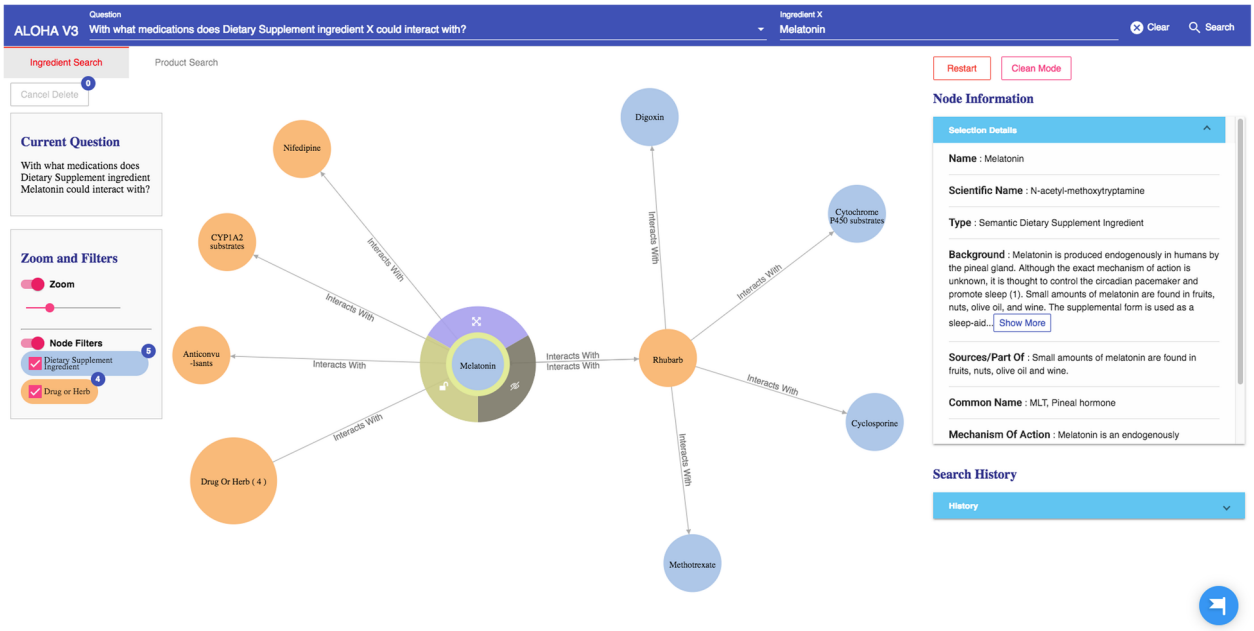
Source: Author

2.2.2 Aloha

Focusing on the context of dietary supplements (DSs), Aloha (HE et al., 2019) seeks to inform consumers of health information about DSs's safety and efficacy. An interactive graph-based visualization system was developed based on a DSs KG. During the system development, two design iterations were performed, with around 10 participants paying attention to user-centered design principles.

The system UI (Figure 2.3) comprises a DS question input field with pre-defined templates, a canvas for the graph visualization and interaction, left and right sections containing information about the current ingredient and question, and option features like zoom and filtering.

Figure 2.3: Aloha interface.



Source: (HE et al., 2019)

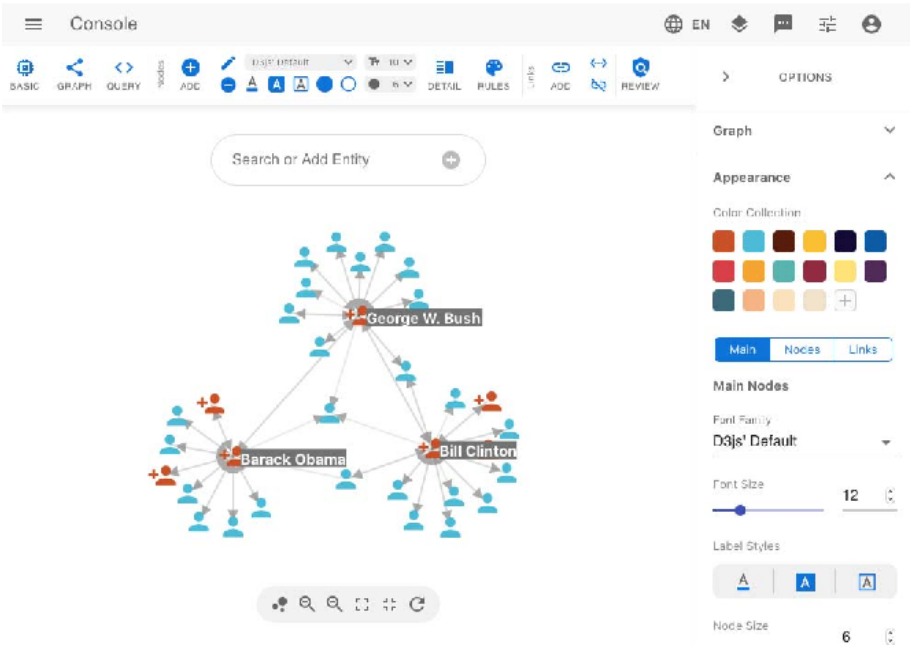
2.2.3 KGViz

KGViz (NARARATWONG; KERTKEIDKACHORN; ICHISE, 2020) (Figure 2.4) is a generic KG visualization framework that was developed based on four dimensions: modularity, intuitive UI, performance, and access control.

One of the most interesting features is the possibility of applying rules where the

user can select a target node or node type and apply specific properties like icon, color, and whether it is visible or not. In addition, it's possible to find the graph canvas in the UI and input it to search for or add an entity. On the right sidebar, there are also options to filter, sort, and even change the appearance of the nodes and links.

Figure 2.4: KGViz interface.



Source: (NARARATWONG; KERTKEIDKACHORN; ICHISE, 2020)

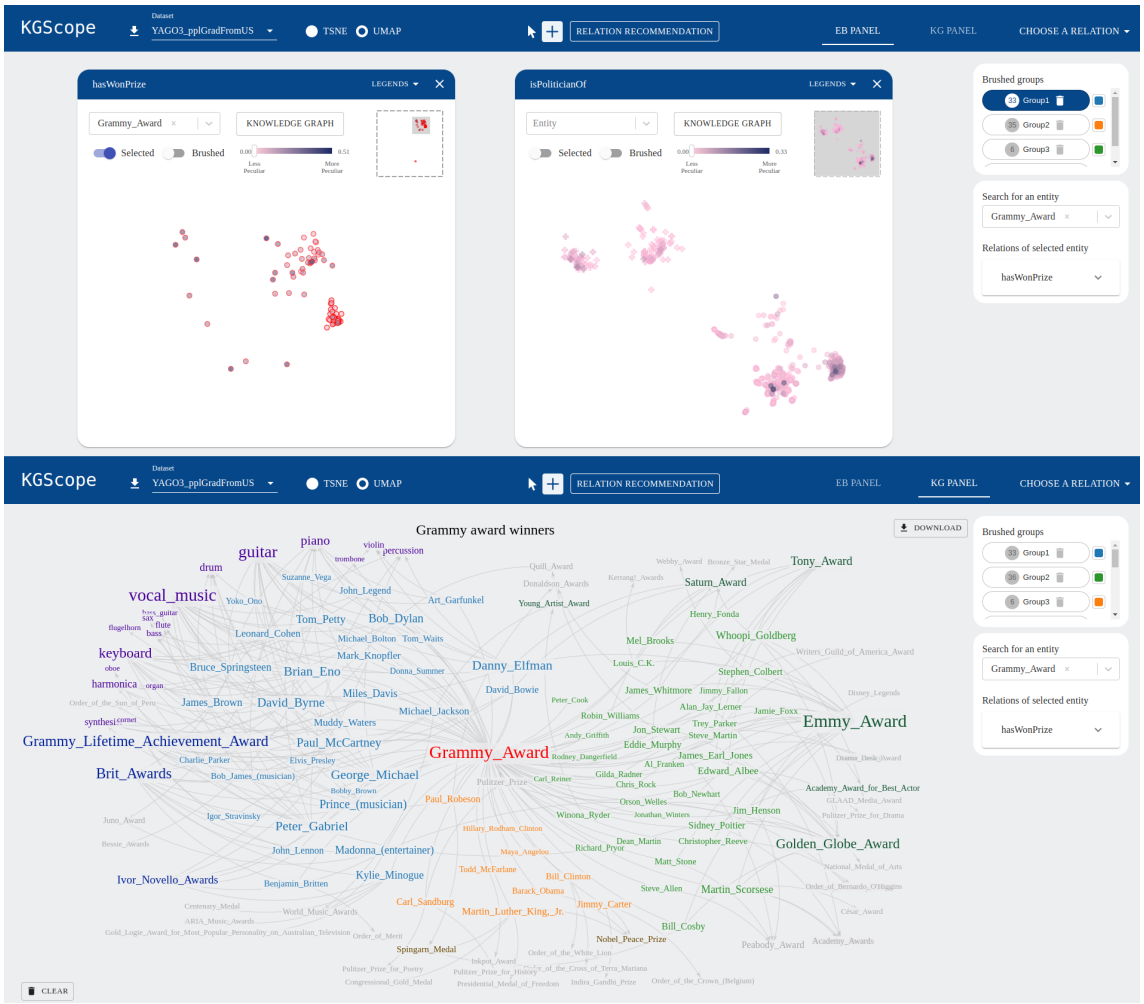
2.2.4 KGScope

KGScope (YUAN et al., 2024) attempts to fill a gap in data analysis tools for KGs, summarizing insights through interactive visual exploration with embedding-based guidance. A user study was performed to validate some usage scenarios.

The UI (Figure 2.5) comprises an embedding panel, multi-relation knowledge graph panel, brushed groups panel, related relation panel, brush tool, relation recommendation button, and schema panel button.

One of the main features is used on the embedding panel, where the user can filter the nodes based on their peculiarity. This makes it possible to identify unexpected insights once peculiarity can measure the level of unexpectedness. There are also filtering and brushing tools to limit the scope and a recommendation button to recommend relations based on embeddings.

Figure 2.5: KGScope interface.



Source: (YUAN et al., 2024)

2.2.5 Wikidata Graph Builder

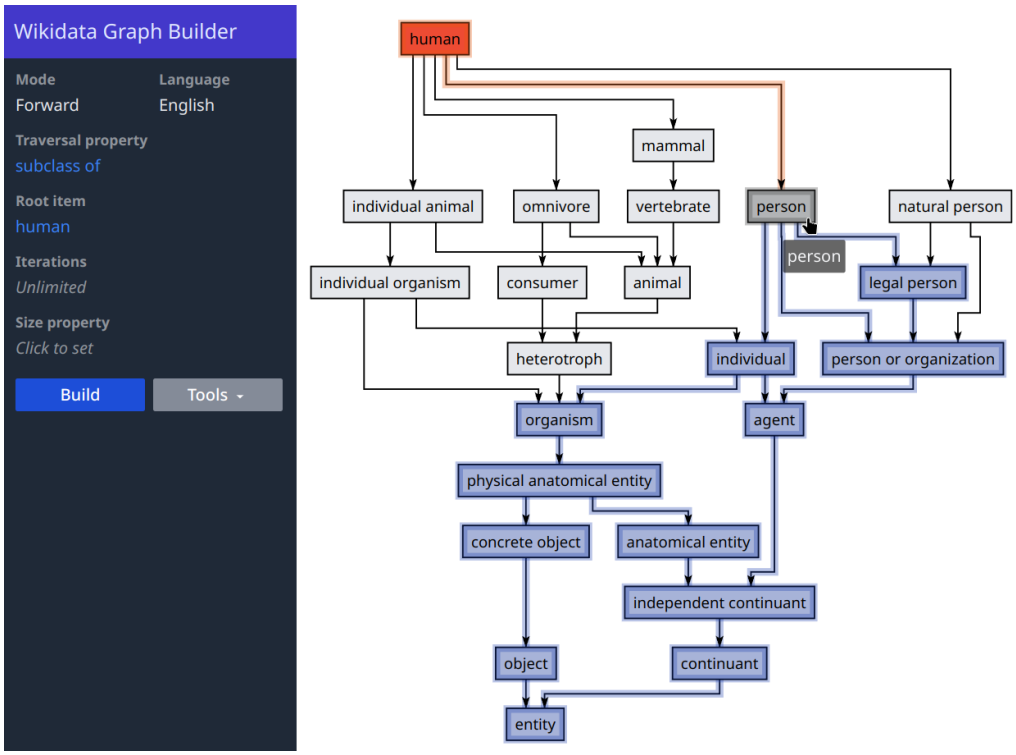
Wikidata Graph Builder [1] (Figure 2.6) uses Wikidata KG to render graphs based on a root item, traversal property, iterations, size property, language, and mode (forward, reverse, bidirectional, undirected, or SPARQL query).

2.2.6 KG Explorer

KG Explorer (EHRHART; LISENA; TRONCY, 2021) is a web-based knowledge graph search engine that can be customized to different information domains. The UI

[1] https://angryloki.github.io/wikidata-graph-builder

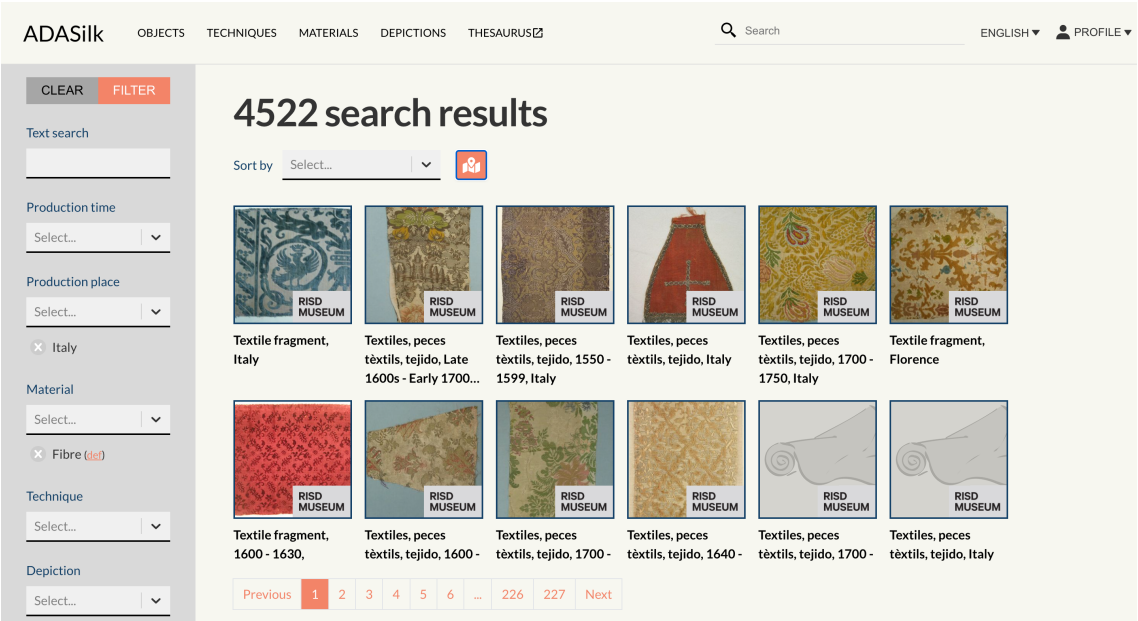
Figure 2.6: Wikidata Graph Builder interface.



Source: Author

components (Figure 2.7) and the queries can both be configured. Some of its features include full-text search, facet-based advanced search, and favorite item lists.

Figure 2.7: KG Explorer interface in one of its in-use instances, ADASilk.



Source: (EHRHART; LISENA; TRONCY, 2021)

3 DATA SOURCES

This work used three main knowledge graph data sources to test the KG explorer tool: DBPedia, CoMem and EnviaM.

3.1 DBPedia

DBPedia (LEHMANN et al., 2015) is a project started in 2007 to extract structured content from Wikipedia, allowing users to create useful queries to access knowledge. Its RDFs triples also link to several external data sources, allowing this information to be used together. It is also one of the central nodes in the Linked Open Data (LOD), once several sources are pointing to it. DBPedia provides several dataset variants and online tools, such as their SPARQL endpoint [1].

The DBPedia extraction framework mainly comprises parsing, ontology mappings, and labeling. Their 2022 snapshot release [2], which is a small subset of the whole DBPedia extraction (around 14% of total), counts more than 850 million triples, with an ontology composed of more than 55 thousand properties (relationship types), 1377 being from DBPedia ontology. It is constantly being updated with new releases.

DBPedia's ontology [3] is formed with 768 classes and is described by 3000 different properties counting with more than 4 million instances. Different from some other KGs, DBPedia's ontology instances can have more than one class (node type).

DBPedia triple example 1:

```
< http://dbpedia.org/resource/Albert_Einstein,
http://dbpedia.org/ontology/wikiPageWikiLink,
http://dbpedia.org/resource/Theory_of_relativity >
```

The Theory of relativity Wikipedia's article was mentioned in Albert Einstein's article, it was mapped to the KG pointing the Theory of relativity DBPedia's node to Albert Einstein's node with the "wikiPageWikiLink" property. This specific property is one of the most common of DBpedia ontology.

DBPedia triple example 2:

```
< http://dbpedia.org/resource/Albert_Einstein,
```

[1] https://dbpedia.org/sparql
[2] https://www.dbpedia.org/blog/dbpedia-snapshot-2022-09-release
[3] https://www.dbpedia.org/resources/ontology/

```
http://dbpedia.org/ontology/doctoralAdvisor,
http://dbpedia.org/resource/Alfred_Kleiner >
```

This triple represents the available information in Albert Einstein's Wikipedia article info box (present on the right side of articles), about its doctoral advisor Alfred Kleiner.

DBPedia triple example 3:

```
< http://dbpedia.org/resource/Albert_Einstein,
http://dbpedia.org/property/birthDate,
1879-03-14 (xsd:date) >
```

As a literal example, this triple stores Albert Einstein's birthdate information. The object, which is a date in the format "xsd:date", can't be a subject in other triples.

3.2 CoMem

CoMem [4] connects several information sources through a dynamic KG aiming to create an ecosystem to assist personal and team management, exploring the concept of a semantic desktop. This process is made by integrating daily work systems such as email, calendar, web browser, and file explorer. The main CoMem users are its creators, the members of the German Research Center for Artificial Intelligence (DFKI) department of Smart Data & Knowledge Services.

Example of CoMem environment usage: The user accesses a conference website, and the sidebar software performs entity recognition and creates nodes in the KG for the conference connecting with its respective entities. Now the user creates an event on the calendar to remember its presentation at the conference. The system can now suggest the location and possible attendees based on the information collected on the conference website.

The KG has more than 100k resources, more than 1.5 million triplets between resources, and more than 300k literals. In CoMem ontology, unlike DBPedia, there is a small and limited number of types, and each entity can have only one type. It counts a total of 65 classes and 254 properties. Many of the properties have a corresponding inverse. It also has the RDF Schema ontology as dependent, so properties like "rdf:type" are used. Examples of original node types (classes) are event, person, location, media/document, project, etc.

[4] https://comem.ai

CoMem triple example 1:

```
< pimo:1342797660296:20 (Topic X),
pimo:thing#hasSubTopic,
pimo:1311091247326:5 (Topic Y) >
```

Entity recognition is present in CoMem software. The system can automatically recognize when a topic is present on a website or in an email body. The "hasSubTopic" property links two of those mapped topics.

CoMem triple example 2:

```
< (Person X) pimo:1681822035293:3,
pimo:thing#isMemberOf,
pimo:1444048778655:22 (Organization Y) >
```

The "isMemberOf" property can be used to link people to organizations, projects, and groups.

CoMem triple example 3:

```
< (Person X) pimo:1681822035293:3,
pimo:thing#receives,
pimo:1561617483167:12 (Email Y) >
```

As CoMem can be integrated with email clients, the received emails can be linked to the person with the "receives" property.

CoMem triple example 4:

```
< (Event X) pimo:8588b367-7e0b-4b30-b1f5-d7300278753a,
pimo:thing#hasAttendee,
pimo:1681822035293:3 (Person Y) >
```

Integration with a calendar client is also possible. Events can be mapped and linked to their attendees with the "hasAttendee" property.

3.3 EnviaM

EnviaM [5] is a German energy provider company that has many separate internal information silos. A KG similar to CoMem was built to make finding connections between the sources easier and then provide better decision-making knowledge. EnviaM's employees are the primary users. The KG has more than 13 million resources, more than 163 million triplets between resources, and more than 29 million literals. Its ontology has the CoMem ontology as a dependence and counts a total of 98 classes and 397 properties.

EnviaM triple example 1:

[5] https://www.enviam-gruppe.de

```
< (Standort X) pimo:ilv:standort:1315632,
pimo:ilv#hatBestandsakte,
pimo:ilv:bestandsakte:Y (Bestandsakte Y) >
```

Two class examples introduced in EnviaM ontology are company location (Standort) and inventory file (Bestandsakte), which can be connected with the property “hatBestandsakte”.

EnviaM triple example 2:

```
< (Standort X) pimo:ilv:standort:1315632,
pimo:ilv#liegtInNetzregion,
pimo:ilv#topic:sachsen-anhalt (Sachsen-Anhalt) >
```

A company location (Standort) can be linked to a network region (Netzregion) using the “liegtInNetzregion” property.

4 KNOWLEDGE GRAPH EXPLORER

This work proposes creating an end-user-centric tool for exploring and visualizing large KGs through filtering, grouping, and exploration features. Once many possible nodes and links are displayed, guiding a non-data expert user through this discovering process is an important task. The methodology consists of starting from a root node selected by the user to expand its outgoing nodes filtering and/or grouping according to user intentions. An outgoing node is a neighbor node with an incoming edge from the expanded node. The action of expanding nodes can be repeated on demand, limited only by a sufficiently high number of nodes, due to visual pollution and performance reasons.

Choosing the right filters can be challenging for the end user. For this reason, auto-filtering methods are important because they can smooth the learning curve and prevent the user from giving up. Thus, choosing between specific or automatic filters creates a balance between flexible settings for expert users and automatic settings for new users.

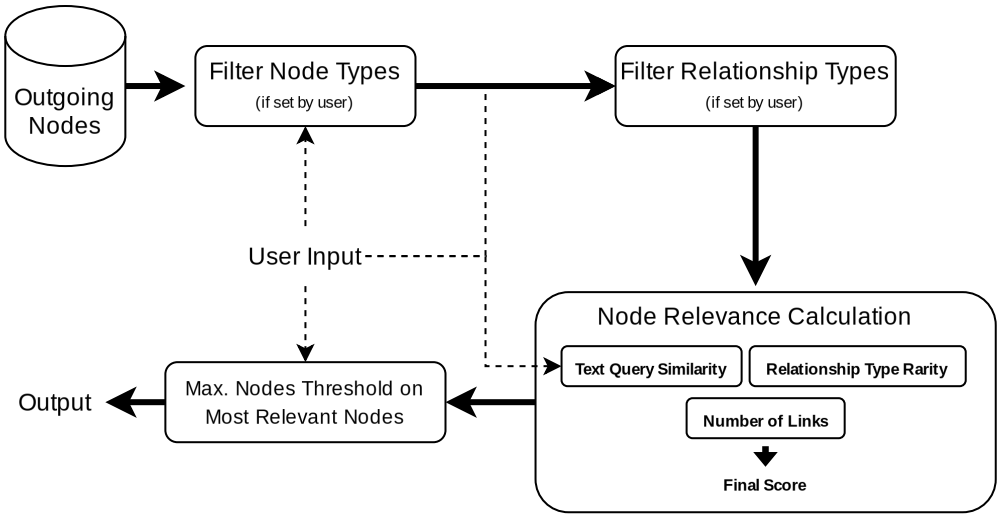
4.1 Filtering

There are many ways to filter the outgoing nodes on an expansion action. In this work, we use the following:

- **Node type:** Select a set of node types to consider or not consider;
- **Relationship type:** Select a set of relationship types to consider or not consider;
- **Number of links:** Outgoing nodes with more links connected to the expanded node can have more relevance;
- **Relationship rarity:** Outgoing nodes with rare relationship types can have more relevance;
- **Text query:** Given a user text query input, calculate the similarity between the input and the available labels (*e.g.,*, node, node type, relationship type);
- **Memory buoyancy threshold:** Filter items with low memory buoyancy (NIEDEREE et al., 2015) value, which represents items that the user rarely accesses.

The user controls the node type filter, relationship type filter, memory buoyancy threshold, and text query input. The number of links and relationship rarity are used as heuristics to calculate a final node relevance score. In Figure 4.1, the filtering flow is illustrated. First, the outgoing nodes that do not fit in the node type and/or relationship

Figure 4.1: Filtering flow. From outgoing nodes set filtered by node and relationship types defined by the user, then node relevance calculation and finally returning the N most relevant nodes according to the threshold also defined by the user.



Source: Author

filters are removed from the set of result outgoing nodes. From the remaining nodes, a node relevance score is calculated based on the relationship rarity, number of links, and text query similarity if the user has set.

4.1.1 Text query similarity

The text query similarity calculation is split into two methods with the same weight in the final Text Query Score (TQS) (Equation 4.3). A Substring Search ($SS1$) of the text query is performed on the concatenated string of the node, node type, and relationship type labels. In addition, another Substring Search ($SS2$) is performed with the concatenated labels with non-alphabetic / non-number characters removed. If any match is found, the return is 1; otherwise, it is 0 (Equation 4.1).

The second method was based on WordNet (MILLER, 1995), which is a database of English verbs, adjectives, and adverbs, totaling more than 150 thousand words. They are connected based on semantic connections like synonyms, hyponyms, and meronyms. Due to those relations between words, this work calculates the similarity between the text query input and the node content to improve results.

To apply the WordNet similarity, both the text query and the concatenated labels have their non-alphabetic / non-number characters replaced by space characters, which

will be used to split to create the words. A pairwise comparison is made between the text query words and the concatenated label words. The average (Equation 4.2) of the similarities above a 0.5 threshold ($SA05$) is returned. After some tests, the 0.5 threshold proved to be useful in enabling only relevant similarities to be considered when calculating the average.

There are several algorithms (PEDERSEN et al., 2004) to measure WordNet's word similarity. This project chose semantic similarity based on corpus statistics and lexical taxonomy (JIANG; CONRATH, 1997). As the output can be greater than one, it was limited to 1 ($max(output, 1)$).

$$SS = max(SS1, SS2) \tag{4.1}$$

$$WNS = average(SA05) \tag{4.2}$$

$$TQS = (WNS + SS)/2 \tag{4.3}$$

4.1.2 Node relevance

The Number of Links Score (Equation 4.4) is calculated by counting the number of links between the expanded node and the outgoing node (CNL) divided by the maximum number of links between the expanded node and any of its outgoing nodes ($MAXNL$). The Relationship Rarity Score (Equation 4.5) is given by one subtracted by the number of nodes with a specific relationship type ($CNRT$) divided by the number of nodes of any relationship type with the maximum count ($MAXNRT$). If a text query is given by the user, the Node Relevance With Text Query (Equation 4.7) is returned as the final node relevance score. Otherwise, the Node Relevance Without Text Query (Equation 4.6) is returned. After some tests, the proportion of the weighted mean of 80% and 20% in the equations proved to work satisfactorily. Once NLS demonstrated through tests to be more important than the RRS in Equation 4.6 and the text query similarity (TQS) more important than the normal node relevance in Equation 4.7, both received 80% as weight.

$$NLS = CNL/MAXNL \tag{4.4}$$

$$RRS = 1 - (CNRT/MAXNRT) \tag{4.5}$$

$$NRWOTQ = NLS * 0.8 + RRS * 0.2 \tag{4.6}$$

$$NRWTQ = TQS * 0.8 + NRWOTQ * 0.2 \tag{4.7}$$

After the final node relevance score for each outgoing node is calculated, the nodes are then sorted based on their relevance. The user can set a maximum number of nodes to be returned during the expansion, and then the top-scored nodes are returned.

4.1.3 Memory buoyancy threshold

Memory buoyancy (NIEDEREE et al., 2015) represents the relevance of an item for a user over time. The value increases as the item is accessed/modified and decreases as the user does not interact with it. Recalculation routines are constantly triggered. This metric was available and used to filter non-relevant items for users in one of this project's implementation environments.

4.2 Grouping

After the node filtering, a grouping step is performed if the user sets it. The two ways to group are by relationship type and node type. The nodes can also be maintained ungrouped. The grouping is applied to the filtered outgoing nodes of a specific node expansion, *i.e.,*, only the newly expanded nodes in the canvas will be grouped. If there is a new group with the same name as an existing one, they are merged.

4.3 Exploration

Exploration features were designed to assist the user in navigating through the graph to achieve its intended objectives. They are the following:

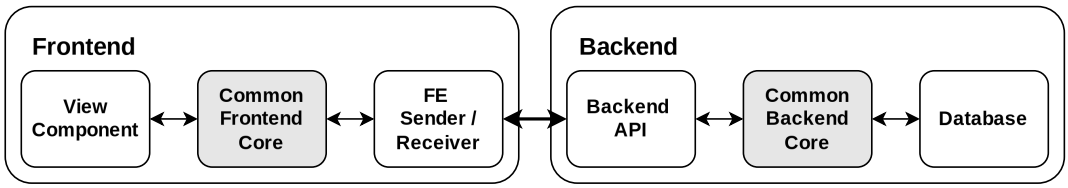
- **Node expansion:** Expand the outgoing nodes of a selected node with specific filtering/grouping features.
- **Move nodes:** Power to rearrange nodes within the graph canvas.

- **Zoom in/out:** Option to increase or decrease the graph canvas view level.
- **Hierarchical node expansion:** Given a relationship type, a root node, and the depth level, then expand it hierarchically.
- **Remove element from view:** Power to remove nodes and groups from the canvas.
- **Expansion plot order:** Helpful plot order of the outgoing nodes after an expansion.
- **Literals list:** Display the relationship type and the values of each literal of a selected node.
- **Relationship information:** Power to verify which relationship types (ontology relation) represent each visual link in the graph on canvas.

4.4 Implementation architecture

This section covers the implementation details. Figure 4.2 provides the architecture scheme, separated by front and back end, and the environment's characteristics.

Figure 4.2: Implementation architecture scheme.



Source: Author

The frontend mainly consists of three modules: the View Component, the Common Frontend Core, and the Frontend (FE) Sender/Receiver. The View Component is responsible for the user interface features, like the graph canvas position, context menus, selection boxes, and general inputs. The Common Frontend Core module stores the main frontend functions and also consumes the Graphly library methods. The FE Sender/Receiver provides methods to access the Backend API.

The backend comprises three parts: the Backend API, the Common Backend Core, and the Database. The backend API specifies the API endpoints called by the core methods. The Common Backend Core module defines the data structures and core methods for processing the KG data retrieved from the database.

The frontend and backend common cores are shared between the environments (Table 4.1). The other modules are specialized according to each environment's needs.

Java is used as the backend programming language. Also, both projects share the same frontend framework and libraries. The Vue.js [1] JavaScript framework and Vuetify [2] component library build the major part of the front end in both environments.

Table 4.1: Environments characteristics.

	Wiki KG Explorer	CoMem / EnviaM
View Component	New frontend project	CoView module
Frontend Core	**Shared**	
Backend API	Spring Boot	CoMem Env. API
Backend Core	**Shared**	
Database	SPARQL DBPedia API	MySQL
Ontology	DBPedia	CoMem / EnviaM

Source: Author

The Graphly [3] library simplifies plotting, rendering, and user interaction with graphs while providing many customization methods. It is built on top of D3.js [4], a popular and powerful JavaScript library for rendering many types of data visualization. Graphly has a force-directed layout engine. In addition, there are available features to lock nodes in specific positions. Fixing a node in a specific x and y coordinate or at a specific angle and distance from another node is possible. If the node is not fixed and is connected to other nodes, it is possible to set a strength value to the edges, modifying the force behavior.

The current plot order implementation arranges the expanded outgoing nodes around the expanded node. The outgoing node's type defines the angle, *i.e.,* nodes of the same type are plotted nearby at the same angle. The distance in an expansion is defined by the total number of nodes in the Wiki KG Explorer environment, by memory buoyancy in the CoMem environment, and by the last modified date in the EnviaM environment. When a user moves a node, it is fixed exactly in the x and y coordinates in which it was positioned. In addition, if during an expansion, an outgoing node is already in the canvas, it loses the fixed position, and the force-directed engine arranges its position.

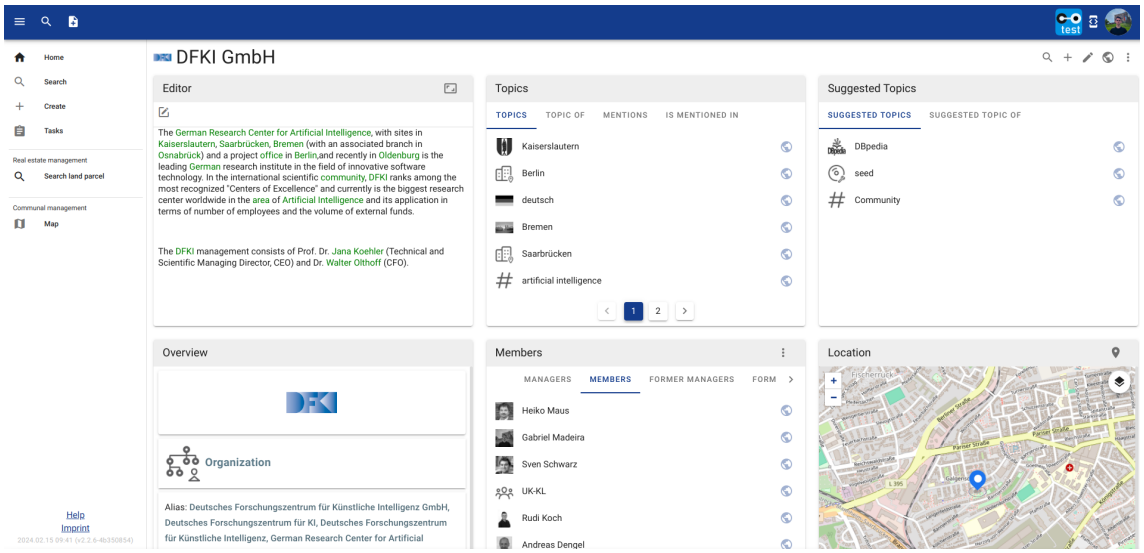
For the hierarchical expansion feature, in CoMem and EnviaM environments, the D3.js method "tree" was used to build a tree node-link diagram, which implements the Reingold-Tilford "tidy" algorithm. The positions generated by the method were converted to Graphly fixed x and y positions. If a node appears more than once in the canvas during the hierarchical expansion, it is moved to the position of the last hierarchical iteration.

[1] https://vuejs.org
[2] https://vuetifyjs.com
[3] https://docs.graphly.dev
[4] https://d3js.org

4.4.1 CoMem and EnviaM environments

For both CoMem and EnviaM environments, the Knowledge Graph Explorer frontend was **implemented as a module inside CoView**. CoView is a user interface that manages and feeds corporate KGs through dashboards and search tools. Figures 4.3 and 4.4 are examples of CoView's dashboards built based on the CoMem and the EnviaM KGs.

Figure 4.3: Example of a CoView dashboard with information about the CoMem KG's node "DFKI".

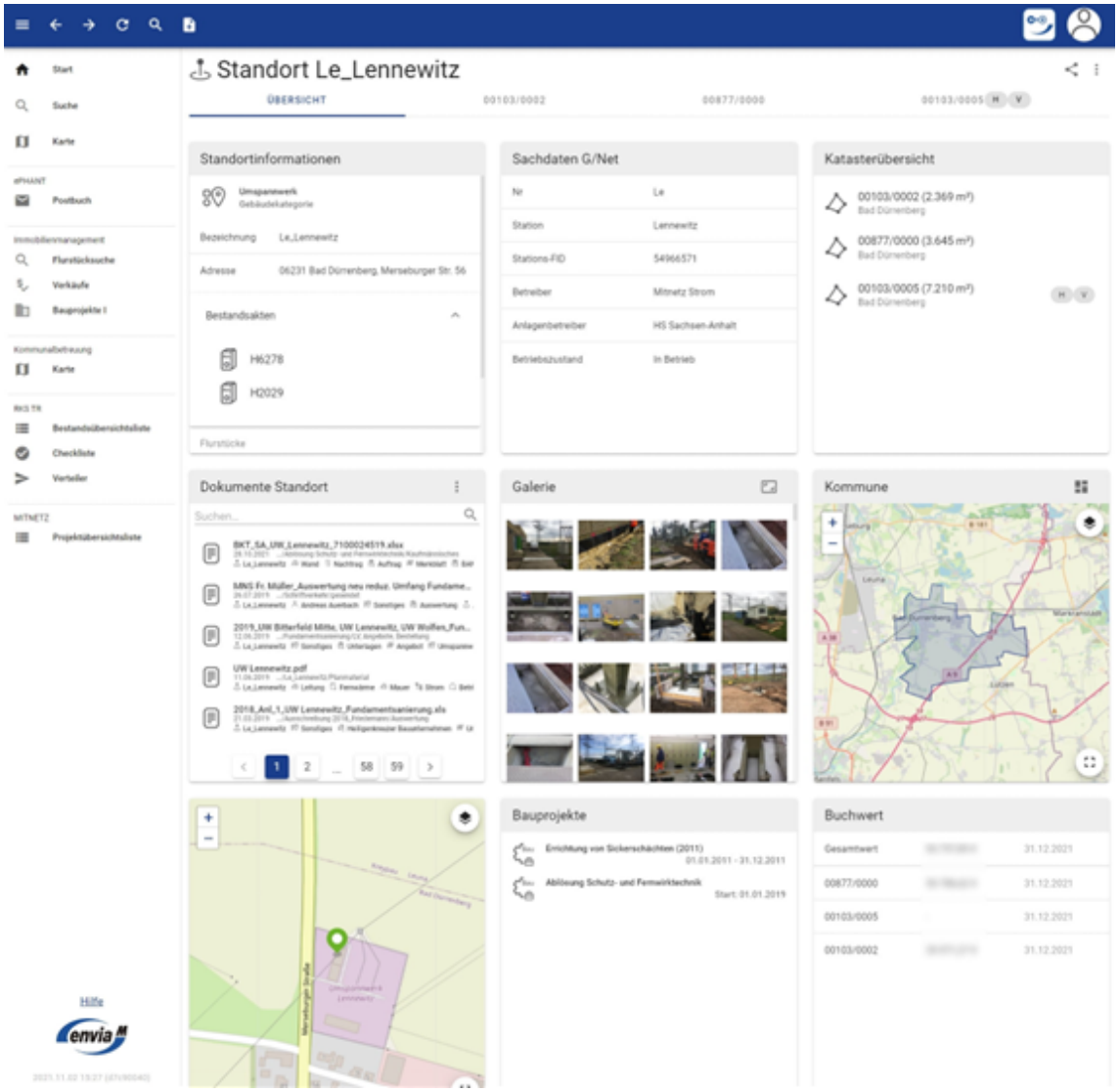


Source: Author

CoView was developed by the Smart Data & Knowledge Services DFKI department, and it is used internally and connected with the CoMem KG. CoView was also implemented by DFKI in the EnviaM infrastructure and is used by company employers to navigate through EnviaM KG.

The Knowledge Graph Explorer (KGE) interface implemented as a module in CoView is presented in Figure 4.5. The first node expansion is performed automatically on the chosen root node (Figure 4.5(a)) with the default expansion settings (Figure 4.5(b)) (node/relationship type filters, and hierarchical settings are hidden in the "Advanced" settings). Using the context menu in any node on the graph (Figure 4.5(c)), it is possible to perform normal expansions (Figure 4.5(d)) and hierarchical expansions (Figure 4.5(e)). In addition, the context menu allows for the performance of some integrated CoView actions (*e.g.*,, if it is a Person node type, email this person). Normal expansions use the same expansion settings as the root was first expanded. Hierarchical expansions use specific settings per expansion, and they are composed of a hierarchical property and maximum

Figure 4.4: Example of a CoView dashboard with information about the EnviaM KG's node "Standort Le_Lennewitz".



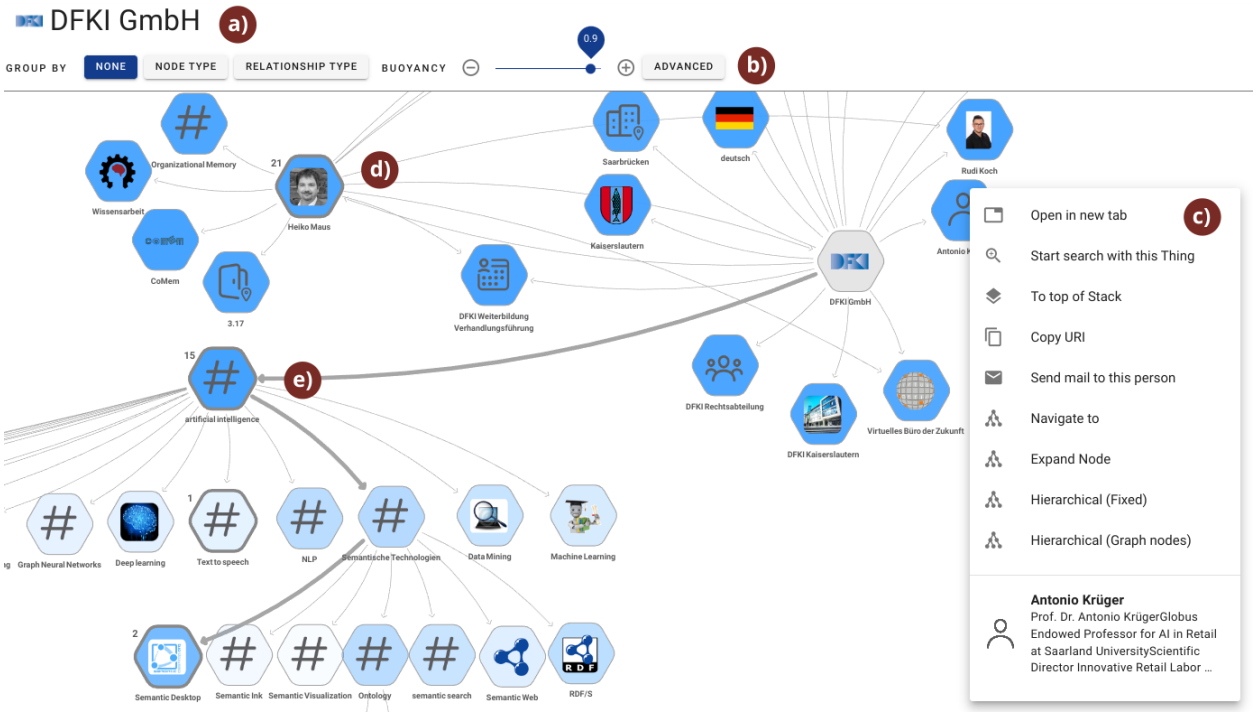
Source: Author

hierarchical depth.

The group by feature can be used to show the root's outgoing nodes grouped by relationship or node type. In low zoom levels, only three group elements are visible (Figure 4.6). Once the zoom is increased (Figure 4.7) it is possible to navigate to all group elements through a scroll bar. The group by feature is limited to one root node level and only with the threshold as the available setting.

Many connected nodes would be plotted without any filter, possibly overloading the graph canvas. Figure 4.8 exemplifies what could happen with the graph view setting the threshold to zero to bring all available outgoing nodes.

Figure 4.5: Knowledge Graph Explorer interface implemented as a module in CoView. Key components are displayed: a) Root node label, b) Expansion settings, c) Context menu, d) Node expanded with normal expansion, e) Node expanded with hierarchical expansion (expanded in a tree layout). After the root node "DFKI" was expanded, the person node "Heiko Maus" expanded, allowing it to get its connected nodes, some already in the graph. The topic node "artificial intelligence" was expanded using the hierarchical expansion with the transversal property "hasSubTopic".



Source: Author

Hierarchical expansions can be useful in scenarios with nodes with specific transversal properties. Some examples of the hierarchical expansion feature in the EnviaM environment are folder organization (Figure 4.9) and company people hierarchy (Figure 4.10).

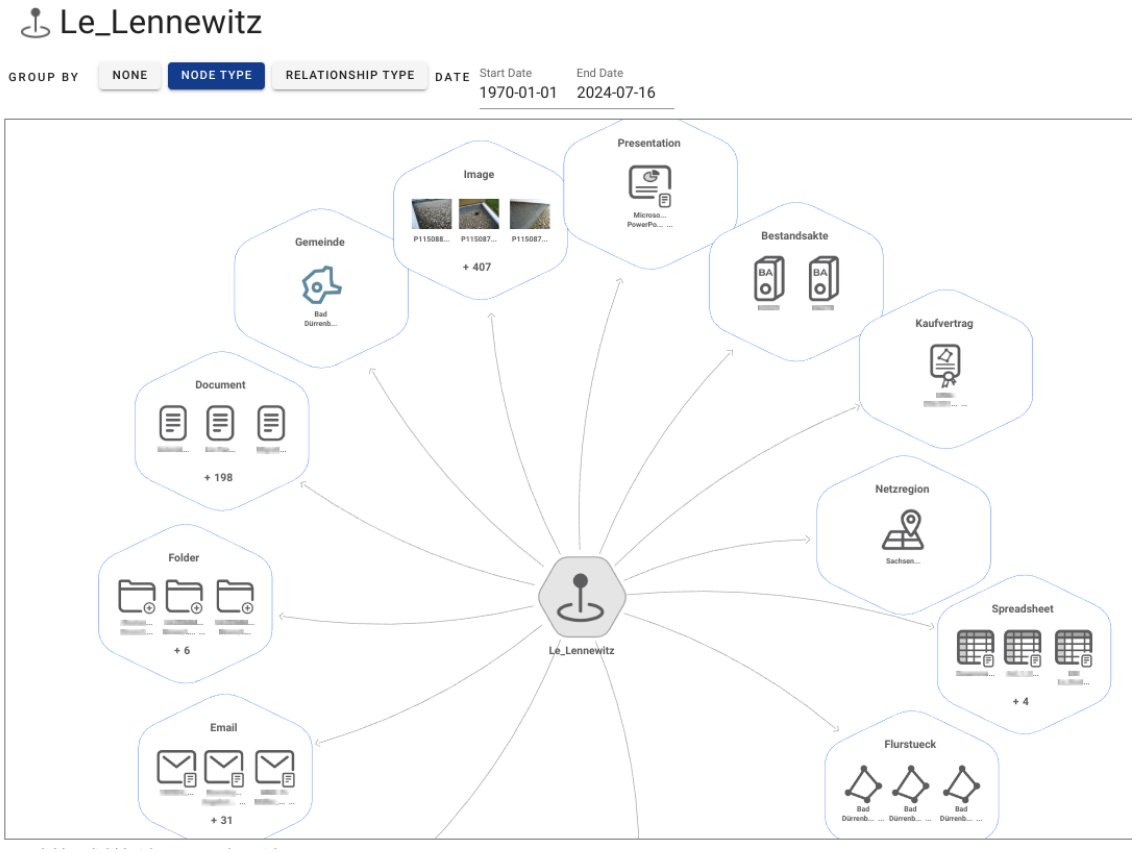
Both CoMem and EnviaM KG databases are instances of MySQL servers, so SQL queries were developed to access their triplets.

4.4.2 Wiki Knowledge Graph Explorer environment

Unlike the last two environments, the Wiki Knowledge Graph Explorer [5] frontend project was created specifically for the tool described in this work. The key UI elements are the graph canvas, selected node information, expansion settings, and central node title.

[5] explorer.gabrielmadeira.com

Figure 4.6: Low zoom level of the “Le_Lennewitz”’s EnviaM node grouped by node type. At this zoom level, only a maximum of three elements are displayed.



Source: Author

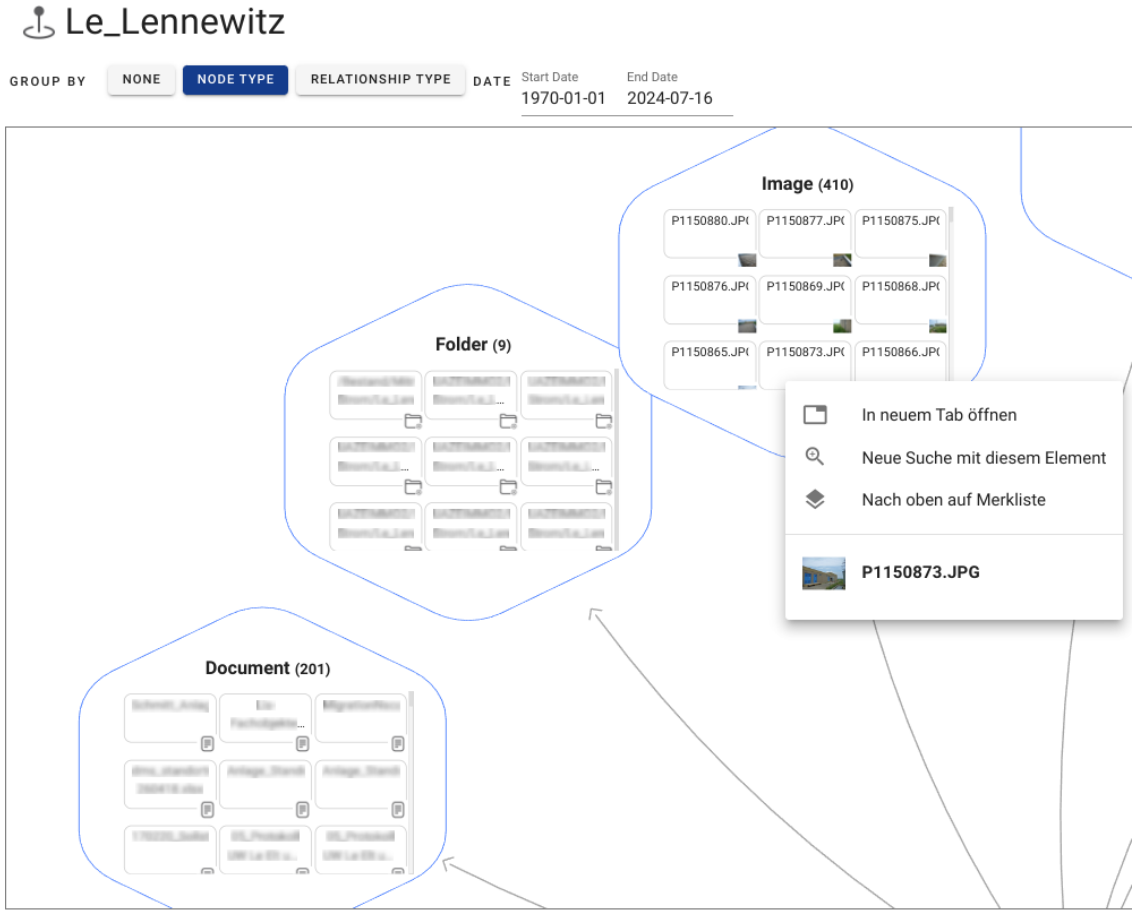
The backend API was developed with the Java library Spring Boot [6]. To retrieve DBPedia KGs data, the DBPedia SPARQL API [7] was used. Although the system is currently dependent on DBPedia API project managers, it would not be difficult to create a local instance of the DBPedia SPARQL database to continue running the application.

The SPARQL query below retrieves the outgoing nodes from a specific node, in this example, from Albert Einstein. For each row, the query returns the outgoing node URI, its label and thumbnail, the property URI, a string list separated by “;” of the node type URIs, and another list with its labels. Many of those are set as optional inside “WHERE” conditions. Some special DBPedia SPARQL API endpoint functions are used. “isIRI” returns true if a URI is a resource or false if it is literal. “langMatches” is used to retrieve the labels in the English language. The literals are retrieved using a different query.

```
SELECT DISTINCT ?outgoingNode ?prop1
(GROUP_CONCAT(DISTINCT ?outgoingNodeTypeUri; SEPARATOR=";") AS ?outgoingNodeTypesUri)
(GROUP_CONCAT(DISTINCT ?outgoingNodeTypeLabel; SEPARATOR=";") AS ?outgoingNodeTypesLabel)
```

[6] https://spring.io/projects/spring-boot
[7] https://dbpedia.org/sparql

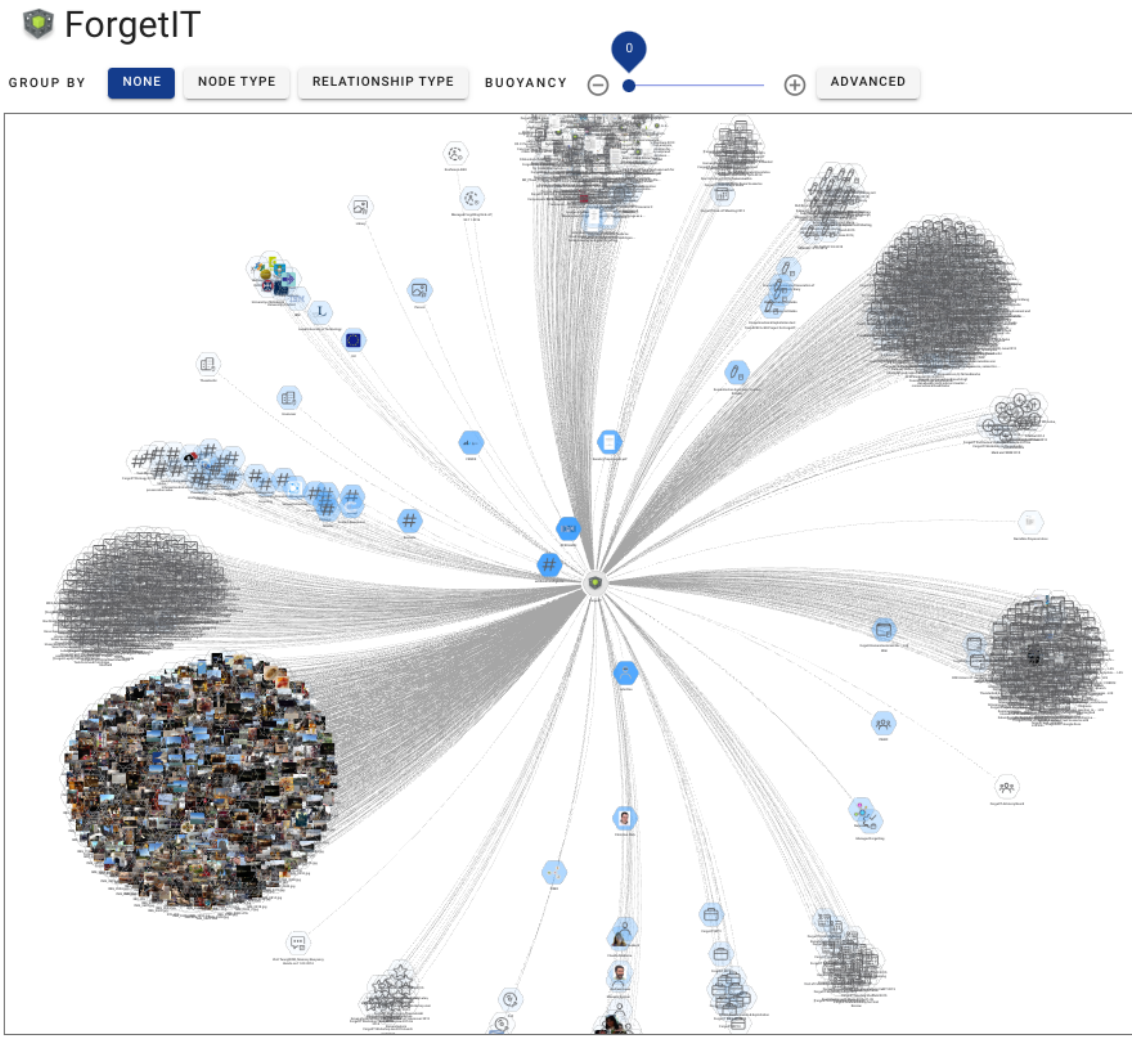
Figure 4.7: High zoom level of the “Le_Lennewitz”’s EnviaM node grouped by node type. At this zoom level, navigating through all group elements is possible by scrolling the list.



Source: Author

```
?outgoingNodeLabel ?outgoingNodeThumbnail
WHERE {
  ?mainNode ?prop1 ?outgoingNode.
  ?outgoingNode rdfs:label ?outgoingNodeLabel.
  FILTER(
    ?mainNode = <http://dbpedia.org/resource/Albert_Einstein> &&
    isIRI(?outgoingNode) && langMatches(lang(?outgoingNodeLabel), "EN")
  ).
  optional{ ?outgoingNode <http://dbpedia.org/ontology/thumbnail> ?outgoingNodeThumbnail. }
  optional{
    ?outgoingNode rdf:type ?outgoingNodeTypeUri.
    FILTER(strstarts(str(?outgoingNodeTypeUri), str(dbo:))).
    optional{
      ?outgoingNodeTypeUri rdfs:label ?outgoingNodeTypeLabel.
      FILTER(langMatches(lang(?outgoingNodeTypeLabel), "EN")).
    }
```

Figure 4.8: "ForgetIT"'s CoMem project node and all its outgoing nodes. Without any filter, the canvas may become overloaded with many outgoing nodes.



Source: Author

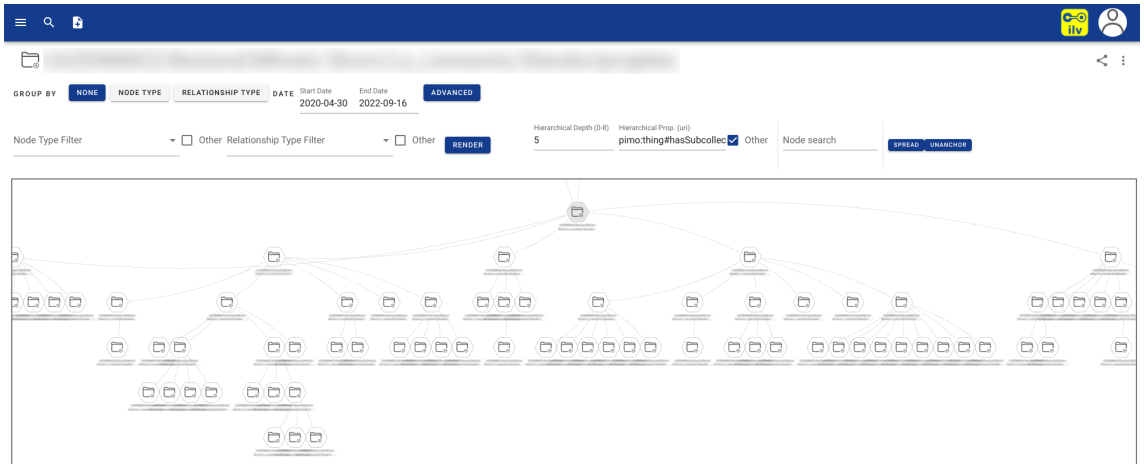
```
  }
} GROUP BY ?outgoingNode ?prop1 ?outgoingNodeLabel ?outgoingNodeThumbnail
```

In addition, a search page (Figure 4.11) was created to enable users to search for DBpedia nodes easily. The system responsible for receiving users' free text queries and returning a list of node labels and URIs is DBPedia Lookup API [8]. Again, although it's dependent on the system's current state, it would not be difficult to run a local instance of DBPedia Lookup once it is open source.

The Wiki Knowledge Graph Explorer's interface is presented in Figure 4.12. The root node is expanded at the first moment. After the root node, the selected node (initialized with the root node, being able to change as the user clicks on other nodes) can be

[8] https://www.dbpedia.org/resources/lookup

Figure 4.9: A hierarchical expansion starting from a root EnviaM's folder node, using the transversal property "hasSubcollection".



Source: Author

expanded with specific settings. The literal list provides additional information about the selected node.

Several possible use cases use the filtering features provided by the Wiki Knowledge Graph Explorer tool. Figure 4.13 demonstrates a use case where the relationship type filter can be used to expand specific relationships between people. Furthermore, Figure 4.14 shows how expanding with specific settings could be beneficial when switching between filter types. Finally, a use case for the text query feature can be found in Figure 4.15.

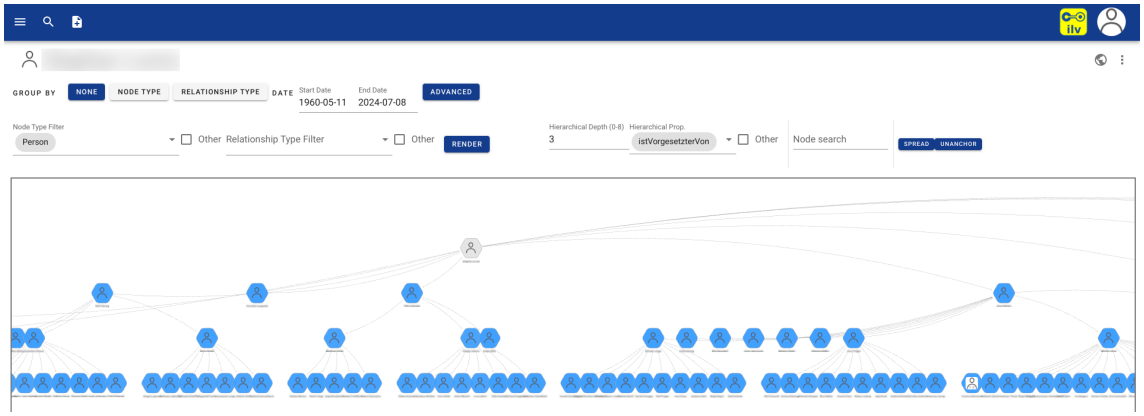
4.4.3 Common core modules

Both frontend and backend core modules were designed to easily share methods between the environments. Although not all features were implemented in all environments, it would not be difficult due to this structure. It contains data structures to which the original environment's KG format is converted. Then, methods to perform operations on top of the graph data structures. In addition, the WS4J (WordNet Similarity for Java) [9] library was used to calculate the WordNet similarity between the text query and node labels.

Due to time constraints and environmental characteristics, the features implemented in each environment vary. Table 4.2 describes which features were implemented

[9] https://code.google.com/archive/p/ws4j

Figure 4.10: A hierarchical expansion starting from a root EnviaM's person node, using the transversal property "istVorgesetzterVon"/"isSuperiorOf".



Source: Author

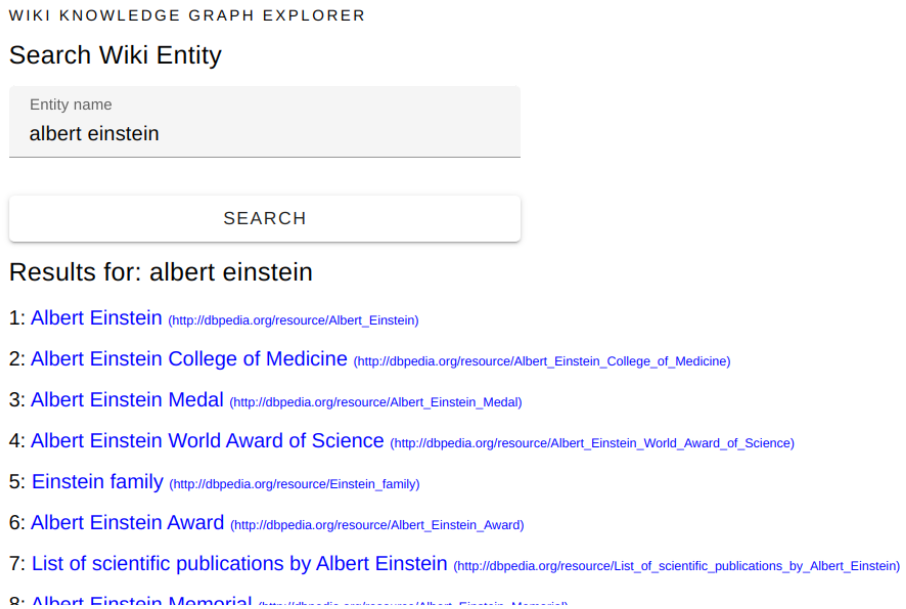
in each environment.

Table 4.2: Implemented features for each environment.

	Wiki KG Expl.	CoMem and EnviaM
Basic: Node expansion, move nodes, zoom	Yes	Yes
Relationship Info	Mouse over dialog	Along link path
Node relevance methods	Yes	No
Remove element from view	Yes	No
Groups	Included on expansion settings	Limited to central node's first level, no expansion features available
Specific settings per node expansion	Yes	No
Hierarchical expansion	No	Yes
Threshold	Max. # nodes threshold	Memory buoyancy (CoMem) and date range (EnviaM)
Text query	Yes	No
Literals list	Yes (in selected node section)	Yes (inside node template)
Node type filter options	Dynamic for each node	Pre-defined set
Relationship type filter options	Dynamic for each node	Pre-defined set

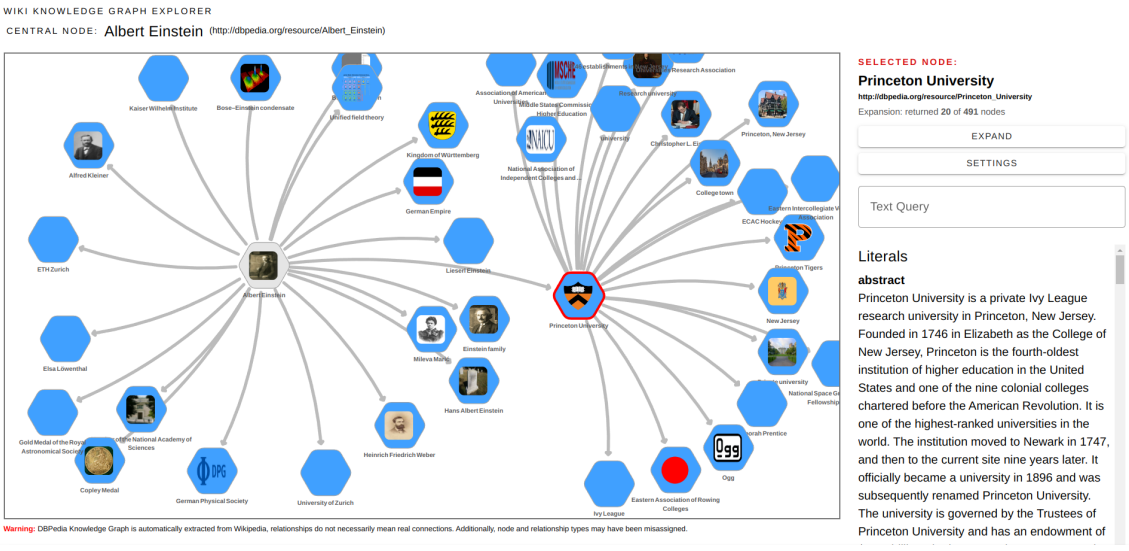
Source: Author

Figure 4.11: Wiki Knowledge Graph Explorer's search page. The text query "Albert Einstein" was given as input, and a list with the most similar DBPedia entities was returned from DBPedia Lookup API.



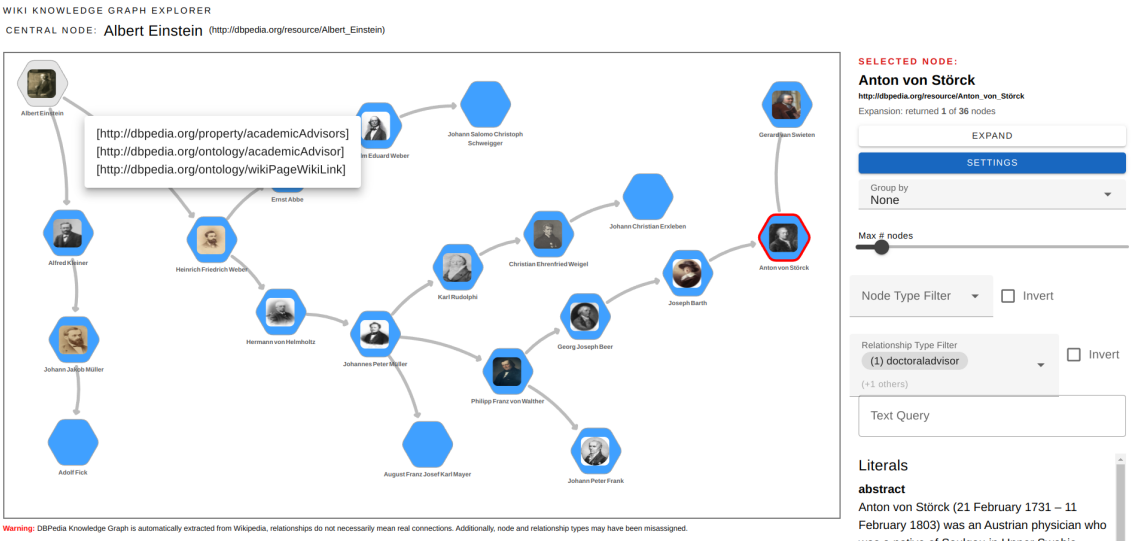
Source: Author

Figure 4.12: The Albert Einstein DBPedia's node is first expanded using default settings, where the top 20 (default value) more related nodes are retrieved. The result is called the "overview" of a node. The same expansion is performed on Albert Einstein's related institution node. The final results are an overview of both Albert Einstein and the institution that connects with it, Princeton University.



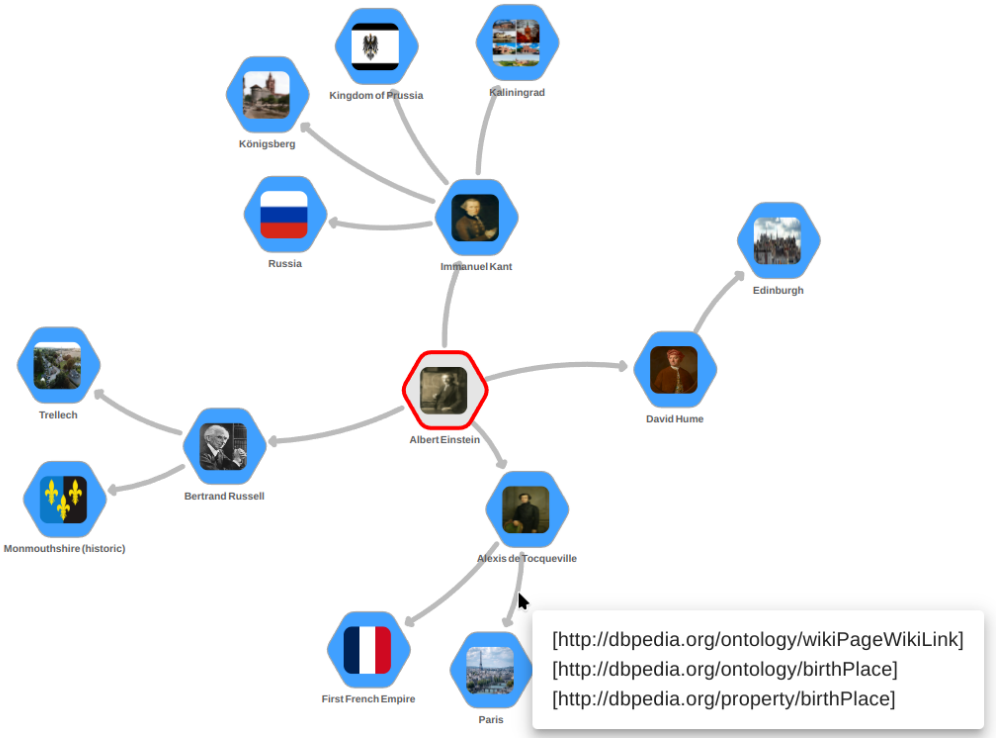
Source: Author

Figure 4.13: The Albert Einstein DBPedia's node is expanded with academic advisors' relationship types. Then the expansion is repeated using the same type of relationship. The result is an academic genealogy tree.



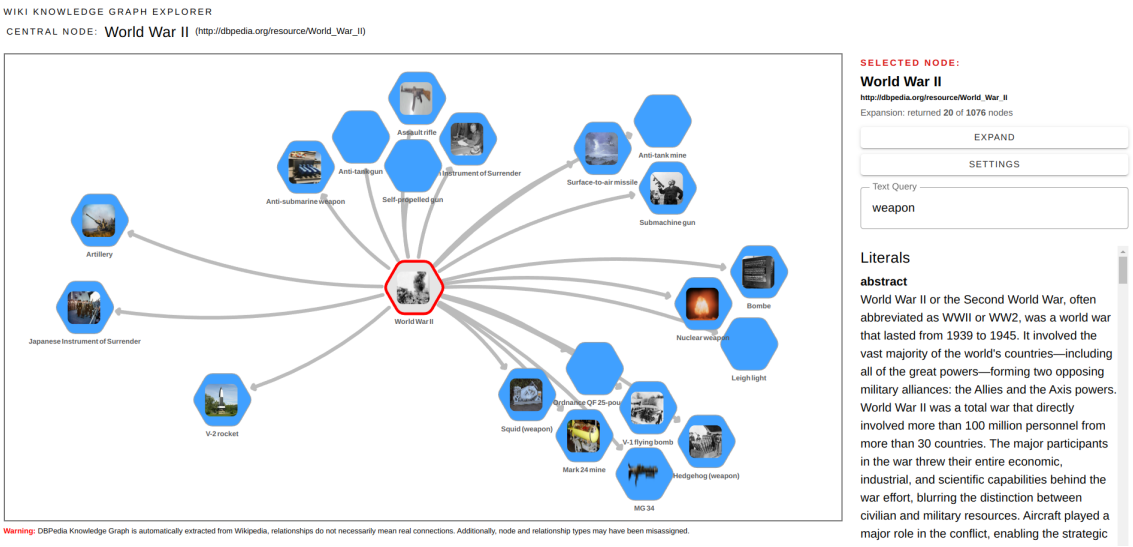
Source: Author

Figure 4.14: The Albert Einstein DBPedia's node is first expanded with philosophers node type filter then, for each philosopher, it was expanded with birthplace relationship type filter. The result is a graph showing the philosophers connected with Albert Einstein and their birthplaces.



Source: Author

Figure 4.15: The World War II DBPedia's node is expanded with the text query "weapon". The result shows, from a total of 1076 outgoing nodes, the top 20 (default threshold) more related nodes to the text query.



Source: Author

5 USER EVALUATION

The User Evaluation was performed in the Wiki Knowledge Graph Explorer (Wiki KG Explorer, KG Explorer, KGE) environment, which uses the DBpedia KG. It was conducted remotely, using a form with five sections: User Profile, Briefing, Usability Evaluation, Formal Evaluation (Tasks), and System Usability Scale. The Briefing and Usability Evaluation sections were guided. In addition, the user was asked to share their screen during most of the experiment, except for the moments when answering the questionnaires. The Formal Evaluation section was recorded for posterior analysis and time measurement. The total expected duration of the experiment is approximately 40 minutes.

Four questions were asked in the User Profile section:

- Do you have an IT (Information Technology) background? (Yes/No)
- How often do you use Wikipedia? (Scale answer: (Never) 1-10 (Every day))
- Do you know the graph concept (from graph theory)? (Yes/No/Partially)
- Do you know the concept of a Knowledge Graph? (Yes/No/Partially)

Then, a guided briefing was made about the KGs, the Wikipedia-DBPedia context, how the application interface works, and its purpose.

5.1 Usability evaluation

The main purpose of this section was to guide users through the first steps of using the system. The user was asked to search and explore the specific DBpedia node of Albert Einstein[1] using the system features. The following sequence of guided actions was performed:

- Overview first expansion outgoing nodes, checking relationship types;
- Expand an outgoing node without any filter;
- Expand it again with the group by node type option;
- Expand it again with the group by relationship type option;
- Test removing a node from a group and ungroup options;
- Expand the Albert Einstein node with just philosophers related to its node (node type filter test);

[1] http://dbpedia.org/resource/Albert_Einstein

- Expand the Albert Einstein node with the academic advisor and birthplace relationships (relationship type filter test);
- Use the system freely for some minutes;

After the user's first interaction with the system, the following questionnaire was applied with a scale answer range from 1 to 10:

- Q1 - How good is the node overview?
- Q2 - How useful is the node type filter?
- Q3 - How useful is the relationship type filter?
- Q4 - How useful are the groups for the exploration?

5.2 Formal evaluation

The formal evaluation aims to compare the execution performance on Wikipedia and in the Knowledge Graph Explorer through two distinct tasks. The user was given some general conditions beforehand:

- It is not allowed to use external search tools (*e.g.,* Wikipedia and Google search engines). Use only hyperlinks to navigate through pages;
- It is not allowed to copy and paste. Although the answer does not need a perfect match of characters, as long as it is possible to understand;
- Finish in the shortest possible time. Time limit of 12 minutes.

5.2.1 Task I

Task I aims to check which approach would be more efficient for searching specific types of information connected to a root concept.

Task description: Search for four events related to Socrates and one city related to each of those events.

Wikipedia Instructions:

1. Access Socrates' Wikipedia page[2]
2. Find hyperlinks to events * cited in Socrates' Wikipedia page; by accessing the

[2]https://en.wikipedia.org/wiki/Socrates

page for these events, find a city related to each of them.

3. Fill in what you found in this form with the following answer format: “event1: city, ... event4: city”.

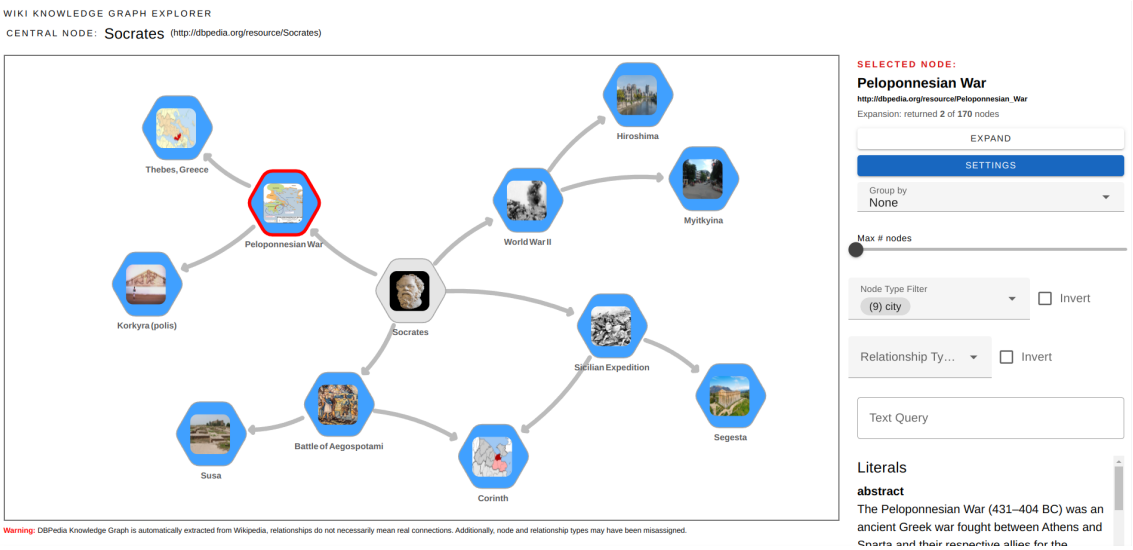
* The user was informed they could ask whether a page hyperlink was considered an event or a city.

KG Explorer Instructions:

1. Access Socrates’ Wiki KG Explorer page[3]
2. Select the event node types connected to Socrates’ node, then expand a city connected to each event.
3. Fill in what you found in this form with the following answer format: “event1: city, ... event4: city”.

Figure 5.1 exemplifies the expected user result on Wiki KG Explorer.

Figure 5.1: Expected user Task I result on Wiki KG Explorer.



Source: Author

5.2.2 Task II

Task II determines the most effective approach for obtaining specific information about concepts connected by some sequential relationship.

[3]http://explorer.gabrielmadeira.com/graphvis/?http://dbpedia.org/resource/Socrates

Task description: Search for recent US presidents, their political party, successor, predecessor, birthplace, and spouse.

Wikipedia Instructions:

1. Access Biden's Wikipedia page[4]
2. Search for party, birthplace, and spouse information in Biden's Wikipedia page, then navigate to the other presidents through predecessor/successor links to continue collecting information.
3. Fill in what you found in this form with the following answer format: "president1: party, birthplace spouse; ... president4: party, birthplace, spouse;"

KG Explorer Instructions:

1. Access Biden's Wiki KG Explorer page[5]
2. Select end expand nodes with the following relationship types: party, birthplace, spouse, and predecessor, first for Biden's node, then continue expanding predecessor presidents. (Presidents sequence: Joe Biden - Donald Trump - Barack Obama - George W. Bush)
3. Fill in what you found in this form with the following answer format: "president1: party, birthplace spouse; ... president4: party, birthplace, spouse;"

Figure 5.2 exemplifies the expected user result on Wiki KG Explorer.

5.2.3 User tasks distribution

Table 5.1 displays the task distribution among the users aiming to reach independent samples, *i.e.,* each user performs a specific task with only one technique.

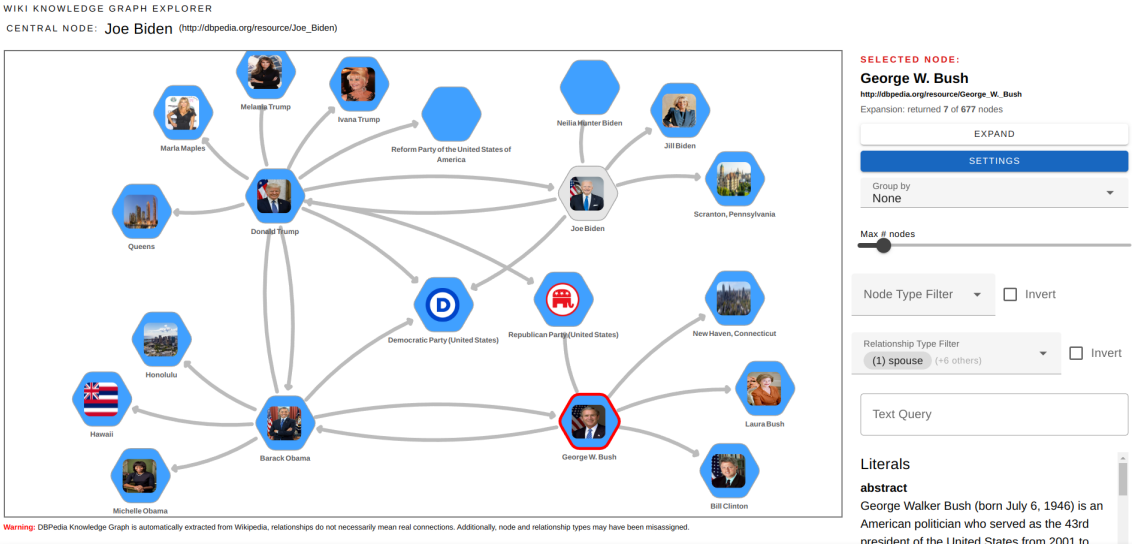
Table 5.1: Distribution of the tasks among users.

	U1	U2	U3	U4	U5	U6	U7	U8	U9	U10
Task Group	A	B	A	B	A	B	A	B	A	B
Wikipedia	T1	T2	T1	T2	T1	T2	T1	T2	T1	T2
KG Explorer	T2	T1	T2	T1	T2	T1	T2	T1	T2	T1

Source: Author

[4]https://en.wikipedia.org/wiki/Joe_Biden
[5]http://explorer.gabrielmadeira.com/graphvis/?http://dbpedia.org/resource/Joe_Biden

Figure 5.2: Expected user Task II result on Wiki KG Explorer.



Source: Author

5.3 System Usability Scale

In the last section of the form, the System Usability Scale (SUS) (BROOKE et al., 1996) questionnaire was applied. SUS provides a set of standardized questions to score the subjective usability experience of a system. The set of questions can help evaluate criteria such as satisfaction, effectiveness, and efficiency. It expects positive answers in the odd question numbers and negative ones in the event. SUS's score ranges from 0 to 100, where above 70 is rated acceptable, neutral between 50 and 70, and not acceptable below 50. It is composed of the following questions with a scale answer range from 1 to 5:

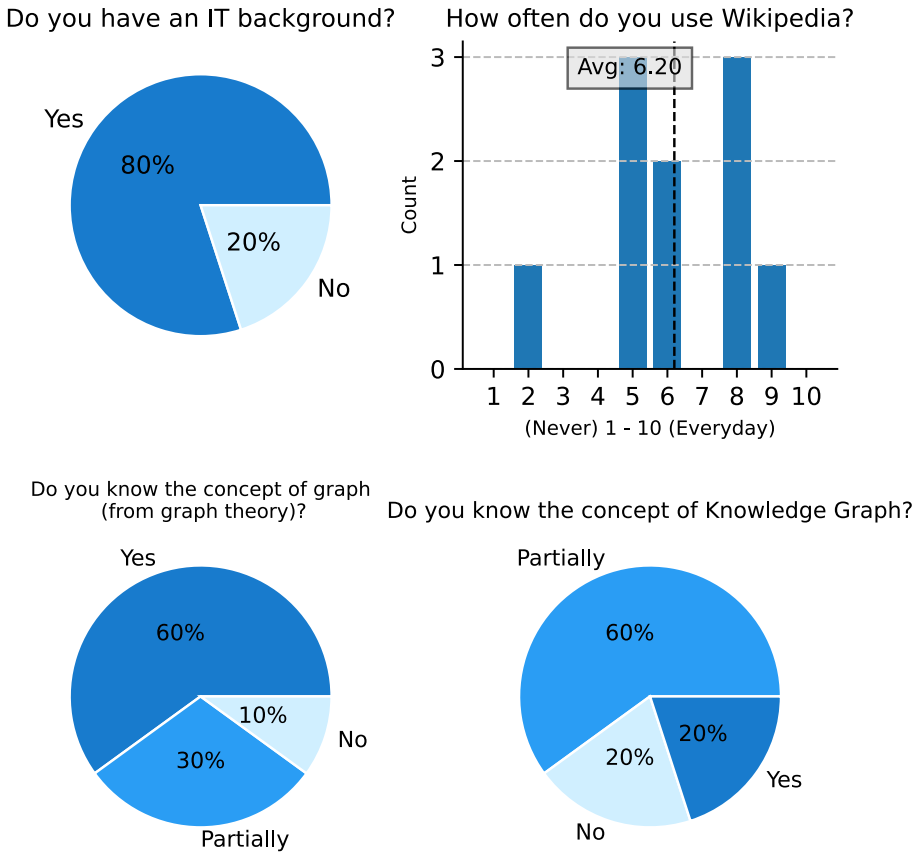
- Q1 - I think that I would like to use this system frequently.
- Q2 - I found the system unnecessarily complex.
- Q3 - I thought the system was easy to use.
- Q4 - I think that I would need the support of a technical person to be able to use this system.
- Q5 - I found the various functions in this system were well integrated.
- Q6 - I thought there was too much inconsistency in this system.
- Q7 - I would imagine that most people would learn to use this system very quickly.
- Q8 - I found the system very cumbersome to use.

- Q9 - I felt very confident using the system.
- Q10 - I needed to learn a lot of things before I could get going with this system.

5.4 Results

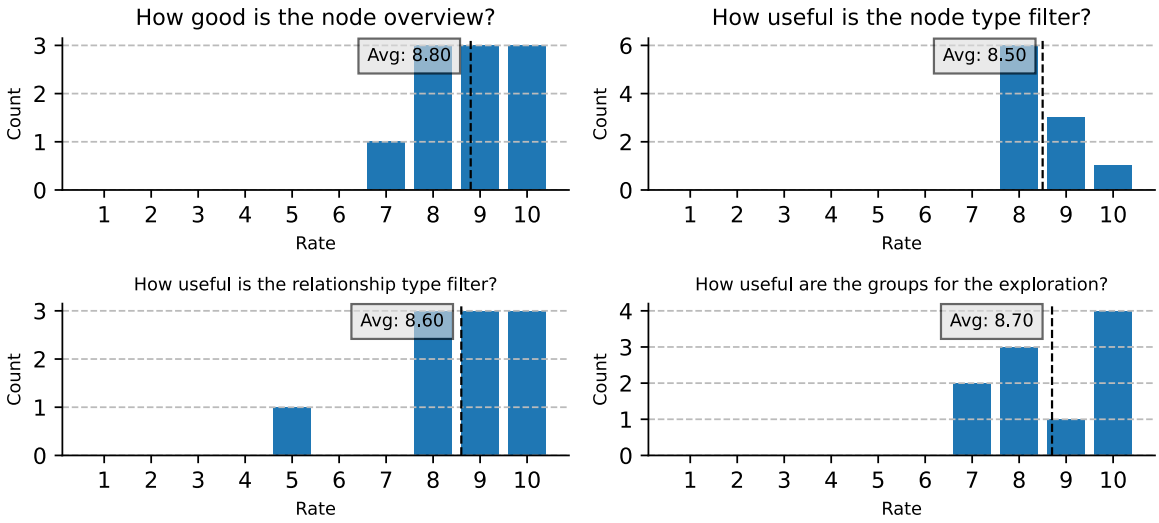
The user profile (Figure 5.3) consists briefly of 80% with an IT background, a normally distributed use of Wikipedia, 60% understanding of the concept of a graph, and finally, 60% partially familiar with the Knowledge Graph concept. The average user rate for the four usability evaluation questions was between 8 and 9 (Figure 5.4). The node overview had the highest average score, and the node type filter feature had the lowest.

Figure 5.3: Pie charts and histogram illustrating answer distribution for each of the four user profile questions.



Source: Author

Figure 5.4: Histogram of the scores given by the user for the four usability evaluation questions.



Source: Author

Regarding the formal evaluation, overall, Task I proved to be more efficiently performed on Wiki KG Explorer (Table 5.2). However, Task II performed better on Wikipedia (Table 5.3).

Table 5.2: **Task I** (Socrates, events, and cities) results (* indicates that the time limit has been reached). With a lower average time and higher average accuracy, it is noticeable that the task performed on **KGE** was **more efficient**.

		Wikipedia		KGE	
User	Task Group	Time(s)	Accuracy(0-1)	Time(s)	Accuracy(0-1)
1	A	387	0.50	-	-
2	B	-	-	393	1.00
3	A	720*	0.75	-	-
4	B	-	-	240	1.00
5	A	720*	0.75	-	-
6	B	-	-	400	1.00
7	A	720*	0.50	-	-
8	B	-	-	212	1.00
9	A	720*	0.75	-	-
10	B	-	-	245	1.00
Average		653.40	0.65	**298.00**	**1.00**

Source: Author

A statistical hypothesis test was conducted to determine if there is a significant difference between the two methods for the two tasks. The time and accuracy metrics were combined into a score value dividing time by accuracy, where the lower values

Table 5.3: **Task II** (US presidents and their political party, birthplace, and spouse) results. With a lower average time and higher average accuracy, it is noticeable that the task performed on **Wikipedia** was **more efficient**.

		Wikipedia		KGE	
User	Task Group	Time(s)	Accuracy(0-1)	Time(s)	Accuracy(0-1)
1	A	-	-	590	1.00
2	B	460	1.00	-	-
3	A	-	-	624	0.92
4	B	267	1.00	-	-
5	A	-	-	600	1.00
6	B	430	1.00	-	-
7	A	-	-	518	1.00
8	B	415	1.00	-	-
9	A	-	-	651	1.00
10	B	286	1.00	-	-
Average		**371.60**	**1.00**	596.60	0.98

Source: Author

mean a better score (Figure 5.5). Three statistical tests were performed using Task I (T1) and Task II (T2) results (Table 5.4). The independent-samples t-test was performed for all three tests, as there are two sample means for each test, and all samples are independent with a Shapiro P-value greater than 0.05. Rejecting the null hypothesis (H0) in Test I, it is possible to conclude that T1 is more efficiently performed on KGE than on Wikipedia. Rejecting H0 on Test II shows that T2 performs significantly better on Wikipedia. Finally, not rejecting H0 on Test III, reveals that there is no significant difference using Wikipedia or KGE (considering the T1 and T2 scenarios combined).

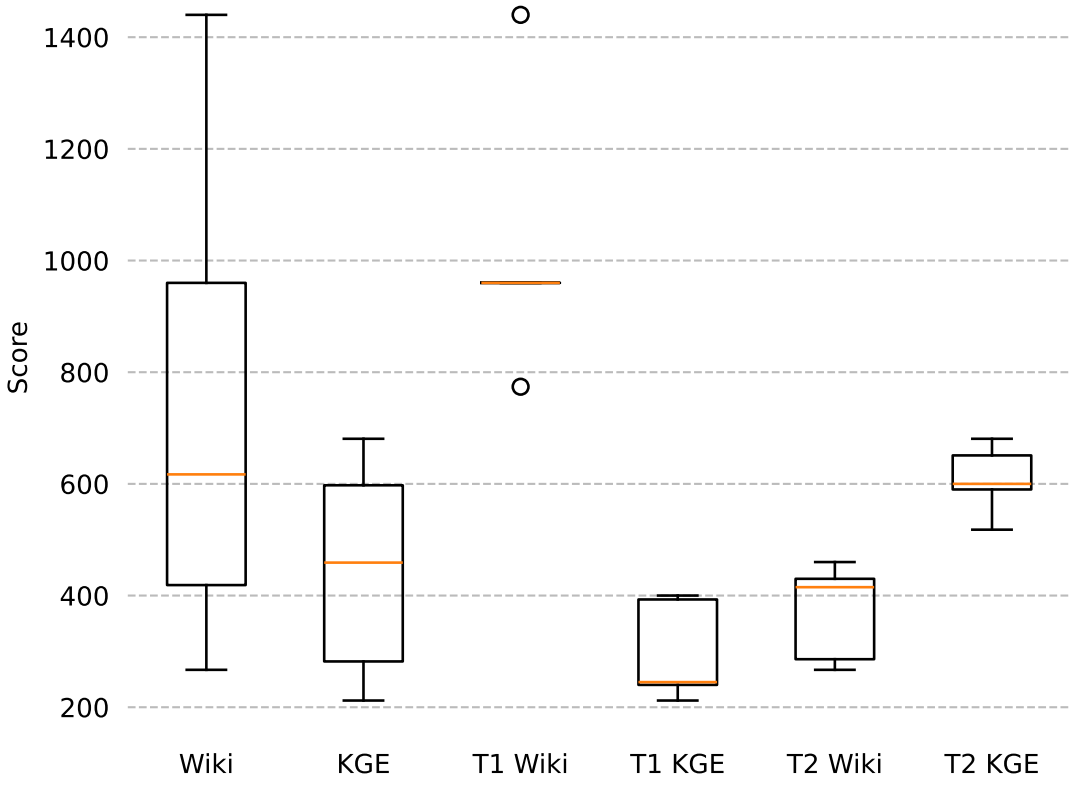
Table 5.4: Statistical tests using Task I (T1) and Task II (T2) results. T1/2 Wiki/KGE are the sets of scores for the individual tasks/methods. Test I proved that KGE performed better in T1. Test II proved that using Wikipedia is more efficient in T2. Finally, Test III suggests that there is no statistical significance in the difference in efficiency between the two approaches when combining the tasks.

	Test I	Test II	Test III
u	T1 Wiki	T2 Wiki	Wiki
v	T1 KGE	T2 KGE	KGE
H0	$u = v$	$u = v$	$u = v$
H1	$u > v$	$u < v$	$u > v$
p-value	0.00029456	0.00123200	0.08731825
Conclusion	Reject H0	Reject H0	Do not reject H0

Source: Author

The final System Usability Scale score indicates that the KGE usability is acceptable (Figure 5.6). Additionally, by analyzing the answer distribution on SUS questions, it is possible to understand which questions got the best and worst results (Figure 5.7).

Figure 5.5: Formal evaluation score. It is calculated with the time divided by accuracy. The lower the result, the better. The first and second box plots are the combined Task I and Task II metrics sets. The four next box plots indicate the score for each task using Wikipedia and KGE.



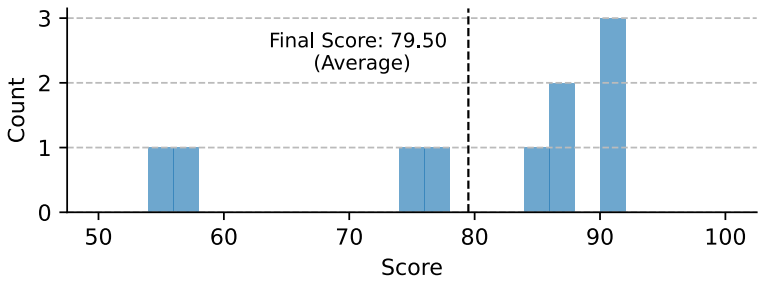
Source: Author

5.4.1 User improvement feedbacks

Besides the positive feedback about the tool's usability, the users gave many improvements feedback. They were mapped as follows:

- Improve UI arrangement. *e.g.,* reduce the distance between type filters select boxes and expand the node button.
- Format relationship type URIs, removing URI useless path.
- Relationship types labels along the node links instead of using a dialog.
- Short tutorial mechanism, introducing main features, how to use, and the key concepts.
- Consider node types ontology tree; once in the current type filter list implementation, they are treated as different types.
- Improve automatically plotting arrangement. *e.g.,* auto move expanded nodes to an

Figure 5.6: Histogram containing each user's System Usability Scale scores and the final score (average). A 79.5 final score is considered acceptable as it is above 70.

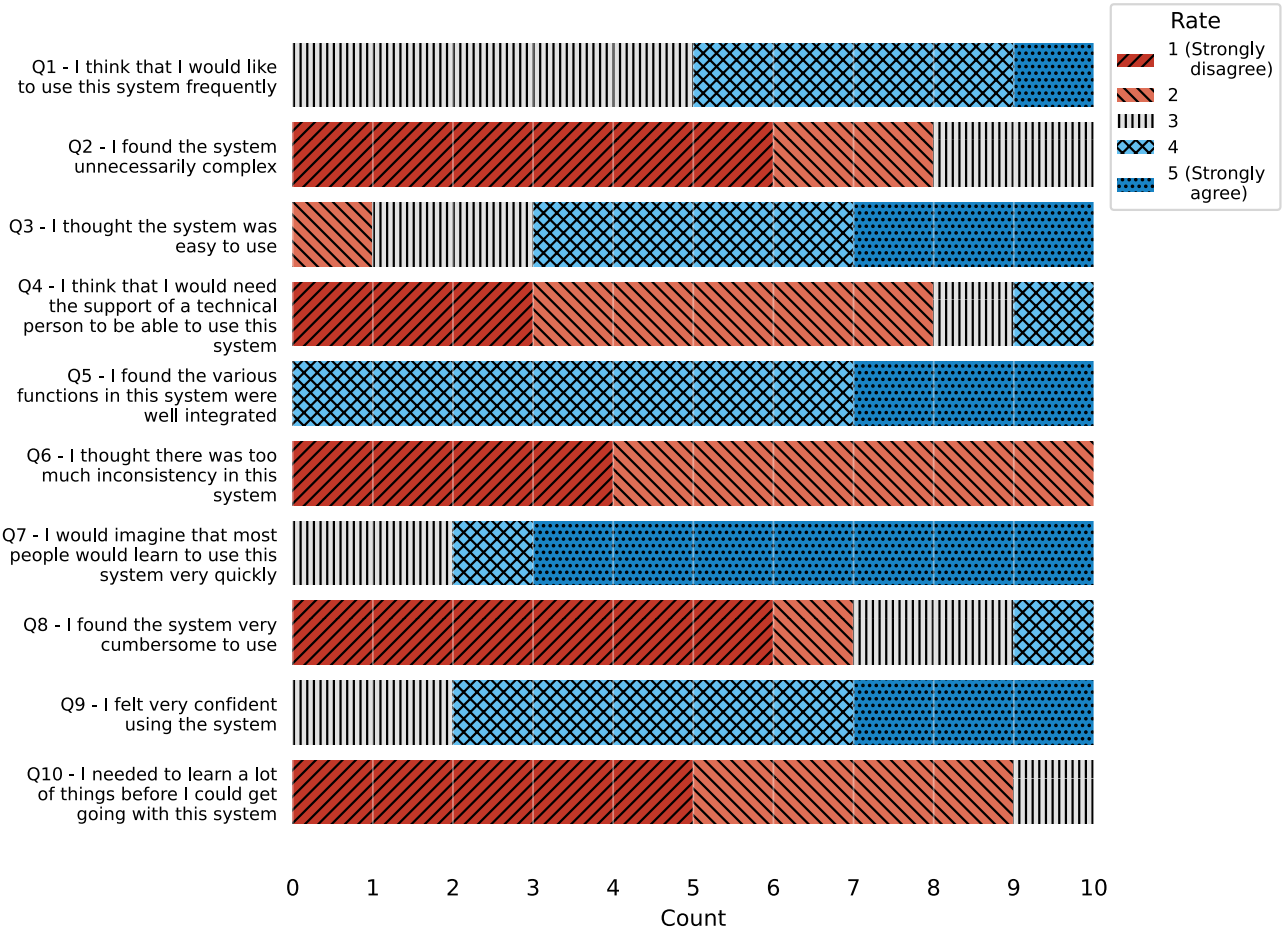


Source: Author

empty area of the canvas.

- Auxiliary buttons: zoom in/out; reset.
- Keep filters after changing the selected node.
- Use colors and highlight mechanisms to improve the view.
- Multi-selection to move the nodes.
- Improve high and low-level zoom font sizes.
- Shortcuts to expand nodes. *e.g.,* double click.
- Improve grouping approach. *e.g.,* the feature can group all the nodes in the canvas instead of just the outgoing nodes.
- Keep the context of already expanded nodes.

Figure 5.7: Stacked bar chart displaying each question's System Usability Scale answer distribution. There are expected to be positive rates for the odd questions and negative rates for the even ones. Considering the rate average, the questions ordered from the best to worst results are: Q7, Q2, Q6, Q10, Q5, Q8, Q9, Q4, Q3, Q1



Source: Author

6 CONCLUSION

Knowledge graphs offer a powerful structure to organize and extract information from data. Node-link diagrams are commonly used to visualize KGs. However, this type of visualization can face scalability issues, mainly with large and generic KGs. A tool with features to enhance exploration and filtering is essential to handle information overload. This allows the user to iteratively and organically explore while focusing on the specific information it seeks.

This work proposes a tool that enables end users to access a generic and large knowledge graph through filtering, exploration, and grouping features. The features allow the user to filter by node and relationship types, set thresholds to get the most relevant nodes, and iteratively repeat this process toward its goals. This process can be done without affecting the capability of the user information process. Implementing the tool in three environments demonstrates it has modular components and could be implemented in other contexts without much effort. Also, one of the three implementations, the Wiki Knowledge Graph Explorer[1], will remain available to the community, with its source code[2] as well.

Through a formal evaluation with the right user task distribution, aiming for independent samples, and applying statistical tests, Task I proved to perform better with the tool developed in this work. This shows how useful for some specific use cases it can be. On the other hand, Task II still proved to be better performed with common methods. In addition, acceptable usability when using the system was reflected by the 79.5 SUS score.

The application could be further improved in many aspects. A better and more intelligent plot approach and an auto-arrange mechanism could be implemented to follow graph visualization aesthetic criteria once it still faces overlapping problems with nodes and edges, demanding the users to constantly move the nodes for better visualization. The semantic zoom feature could be better implemented by bringing in-node details as the user zooms in. It would be important to save the user context at some level, once in the current implementation it loses progress as soon it leaves the page. Additionally, keeping track of the current user actions, custom relevance for filtering nodes could be applied, creating a recommender system aligned with most user content preferences. The text query could be improved to support stop word variations and even misspelled words, and it could even be combined with an automatic threshold set. This work focused on the English

[1] http://explorer.gabrielmadeira.com
[2] https://github.com/gabrielmadeira/wiki-kg-explorer

language, but it could be extended to other languages. Finally, several UI improvements were mapped and pointed out by the users' feedback. Additional user evaluation rounds with a larger number of users would increase the usability and efficacy of the application even more. These should also include non-evaluated features, such as the text query.

REFERENCES

BENNETT, C. et al. The aesthetics of graph visualization. In: **CAe**. [S.l.: s.n.], 2007. p. 57–64.

BRATT, S. **Semantic Web, and Other Technologies to Watch**. 2007. <https://www.w3.org/2007/Talks/0130-sb-W3CTechSemWeb/#(1)>. Accessed: 2024-08-12.

BROOKE, J. et al. Sus-a quick and dirty usability scale. **Usability evaluation in industry**, London, England, v. 189, n. 194, p. 4–7, 1996.

CAMARDA, D. V.; MAZZINI, S.; ANTONUCCIO, A. Lodlive, exploring the web of data. In: **Proceedings of the 8th International Conference on Semantic Systems**. [S.l.: s.n.], 2012. p. 197–200.

EHRHART, T.; LISENA, P.; TRONCY, R. Kg explorer: a customisable exploration tool for knowledge graphs. In: **VOILA 2021, International Workshop on the Visualization and Interaction for Ontologies and Linked Data**. [S.l.: s.n.], 2021.

FRUCHTERMAN, T. M.; REINGOLD, E. M. Graph drawing by force-directed placement. **Software: Practice and experience**, Wiley Online Library, v. 21, n. 11, p. 1129–1164, 1991.

HE, X. et al. Aloha: developing an interactive graph-based visualization for dietary supplement knowledge graph through user-centered design. **BMC medical informatics and decision making**, Springer, v. 19, p. 1–18, 2019.

JI, S. et al. A survey on knowledge graphs: Representation, acquisition, and applications. **IEEE transactions on neural networks and learning systems**, IEEE, v. 33, n. 2, p. 494–514, 2021.

JIANG, J. J.; CONRATH, D. W. Semantic similarity based on corpus statistics and lexical taxonomy. **arXiv preprint cmp-lg/9709008**, 1997.

LASSILA, O.; HENDLER, J.; BERNERS-LEE, T. The semantic web. **Scientific American**, v. 284, n. 5, p. 34–43, 2001.

LEHMANN, J. et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. In: . [S.l.]: IOS Press, 2015. v. 6, n. 2, p. 167–195.

LI, H. et al. Knowledge graphs in practice: Characterizing their users, challenges, and visualization opportunities. **IEEE Transactions on Visualization and Computer Graphics**, PP, p. 1–11, 12 2023.

MILLER, G. A. Wordnet: a lexical database for english. **Communications of the ACM**, ACM New York, NY, USA, v. 38, n. 11, p. 39–41, 1995.

NARARATWONG, R.; KERTKEIDKACHORN, N.; ICHISE, R. Knowledge graph visualization: Challenges, framework, and implementation. In: IEEE. **2020 IEEE Third International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)**. [S.l.], 2020. p. 174–178.

NIEDEREE, C. et al. Forgetful digital memory: Towards brain-inspired long-term data and information management. **ACM Sigmod Record**, ACM New York, NY, USA, v. 44, n. 2, p. 41–46, 2015.

PEDERSEN, T. et al. Wordnet:: Similarity-measuring the relatedness of concepts. In: **AAAI**. [S.l.: s.n.], 2004. v. 4, p. 25–29.

REINGOLD, E.; TILFORD, J. Tidier drawings of trees. **IEEE Transactions on Software Engineering**, SE-7, n. 2, p. 223–228, 1981.

SETH, Y. **Introduction to Question Answering over Knowledge Graphs**. 2019. <https://yashuseth.wordpress.com/2019/10/08/introduction-question-answering-knowledge-graphs-kgqa/>. Accessed: 2024-08-12.

SHADBOLT, N.; BERNERS-LEE, T.; HALL, W. The semantic web revisited. **IEEE Intelligent Systems**, v. 21, n. 3, p. 96–101, 2006.

WANG, Q. et al. Knowledge graph embedding: A survey of approaches and applications. **IEEE Transactions on Knowledge and Data Engineering**, v. 29, n. 12, p. 2724–2743, 2017.

YUAN, C.-W. H. et al. Kgscope: Interactive visual exploration of knowledge graphs with embedding-based guidance. **IEEE Transactions on Visualization and Computer Graphics**, IEEE, 2024.