

Library of Open Source Hardware: Creating a Semantic Knowledge Base and Search Engine for Open Source Hardware

Master thesis by André Lehmann

Date of submission: September 30, 2022

1. Review: Prof. Dr. Max Mühlhäuser

2. Review: Dominik Schön
Darmstadt



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Computer Science
Department
Telecooperation Lab

Abstract

The concept of open source has proven to be a great driver of innovations in terms of software and played a crucial role in the growth of digital technologies. It is believed that openly sharing designs, implementation, and documentation could have the same effect on hardware as well and could lead to rapid technological advancements [1]. However, the lack of accepted standards and tools for the development and documentation of open hardware and the unclear legal implications hinder Open Source Hardware (OSH) from unfolding its full potential. One step towards the success of OSH is to improve the discoverability of already existing hardware solutions. There are many popular online platforms for sharing hardware designs, but it is currently impossible to conveniently search for designs across multiple platforms.

The primary goal of this thesis was to create a semantic knowledge base and a web search engine for OSH products. We placed a strong emphasis on usability and user experience and projected our work accordingly. To assess the system requirements, we conducted expert interviews with members of the OSH community. The interview evaluation showed many unexpected and previously unknown user needs and wishes. We used the results to design and implement a system that not only meets the requirements but also provides a remarkably good usability and a satisfying user experience. The developed search engine collects product information from various content hosting platforms, creates an index for eligible products, and allows users to find them by their properties using an advanced query syntax. Furthermore, the service offers Semantic Web (SW) technologies to create meaningful data connections with other datasets on the web and empower future research. In order to test and validate our service, we conducted a usability testing, where we asked the previously interviewed participants to solve a set of tasks using our service and observed their interaction. The observation's outcome and the results of the subsequent questioning were evaluated and used to derive measures for further improvements.

The usability testing showed us that users could easily find product designs, filter them as desired, and configure how to display the results. The participants expressed satisfaction and deemed the service to be useful. Compared with other similar search engines, such as OHO search and the OPENNEXT LOSH demonstrator, our service provides a significantly better search functionality and a more user-friendly interface. We conclude that our contribution improves the discoverability of OSH products and proof that our service is a valuable addition to the OSH community's toolset. We are in contact with representatives of the OPENNEXT research project to discuss how our results can be exploited and improved in current and future research projects.

Contents

List of Abbreviations	IV
List of Figures	V
List of Tables	VI
1. Introduction	1
1.1. Context and Motivation	1
1.2. Problem Statement	2
2. Background	4
2.1. Open Source Hardware	4
2.2. Semantic Web	5
3. Related Work	7
3.1. Search Engines	7
3.2. Open Hardware Observatory	8
3.3. Open Know-How Specification	8
3.4. OPENNEXT - Library of Open Source Hardware	9
4. Requirements Analysis	13
4.1. Introduction	13
4.2. Requirements Interview	13
4.3. Terminology and Conventions	14
4.4. Requirements	15
4.4.1. Crawler	15
4.4.2. Product Search	18
4.4.3. RDF Resource	25
4.4.4. API	26
5. Design and Implementation	27
5.1. System Architecture	27
5.1.1. Reasoning for a New Implementation	28
5.1.2. Notable Components, Libraries and Frameworks	29
5.2. Database and Data Model	32
5.3. Crawler	33
5.4. Web Interface	40
5.4.1. Product Search	46
5.4.2. Semantic Web	54

6. Evaluation and Validation	55
6.1. Introduction to Usability Testing	55
6.2. Methodology	56
6.3. Results	57
6.4. Discussion	68
7. Conclusion and Future Work	71
7.1. Conclusion	71
7.2. Future Work	72
Appendices	VII
A. Requirements Analysis Interview Questions	VII
B. Usability Testing Tasks	XI
C. Profiles of the Interviewees	XV
D. Query Syntax Definition	XX
Bibliography	XXVIII



List of Abbreviations

API Application Programming Interface

AST Abstract Syntax Tree

EBNF Extended Backus–Naur Form

LD Linked Data

LOSH Library of Open Source Hardware

OHO Open Hardware Observatory

OKH Open Know How

OSH Open Source Hardware

OSS Open Source Software

OWL Web Ontology Language

RDF Resource Description Framework

SW Semantic Web

List of Figures

3.1. LOSH Demonstrator Network Diagram	12
5.1. Network Diagram	28
5.2. Discover Products Flowchart	35
5.3. Update Products Flowchart	35
5.4. Add Product Entry Flowchart	36
5.5. Update Product Entry Flowchart	36
5.6. Request Resource Flowchart	37
5.7. Request Product Information Flowchart	37
5.8. Homepage	40
5.9. Product Search Results Page	42
5.10. Query Syntax Cheat Sheet	43
5.11. Product Details Page	43
5.12. Licensor Details Page	44
5.13. License Details Page	44
5.14. RDF Resource Page	45
5.15. Product Search Sequence Diagram	46
5.16. Handle Search Request Flowchart	47
5.17. Query Syntax Railroad Diagram	50
5.18. Handle RDF Resource Request Flowchart	54
5.19. Handle RDF Resource Representation Request Flowchart	54
6.1. Number of issues by category	68

List of Tables

3.1. LOSH Demonstrator Components	10
3.1. LOSH Demonstrator Components	12
D.1. Query Syntax - General Expressions	XX
D.2. Query Syntax - <i>Text</i> Operator Type	XXI
D.3. Query Syntax - <i>Boolean</i> Operator Type	XXI
D.4. Query Syntax - <i>Number</i> Operator Type	XXI
D.4. Query Syntax - <i>Number</i> Operator Type	XXII
D.5. Query Syntax - <i>DateTime</i> Operator Type	XXII
D.6. Query Syntax - Basic Operators	XXIII
D.7. Query Syntax - Categorization	XXIII
D.7. Query Syntax - Categorization	XXIV
D.8. Query Syntax - Repository	XXIV
D.9. Query Syntax - License	XXIV
D.9. Query Syntax - License	XXV
D.10. Query Syntax - Licensor	XXVI
D.11. Query Syntax - Files	XXVI
D.12. Query Syntax - Standard, Publication, Maturity, etc.	XXVI
D.12. Query Syntax - Standard, Publication, Maturity, etc.	XXVII

1. Introduction

1.1. Context and Motivation

In terms of software, the concept of *open source* proved to be a driver of great innovations. Today, most modern businesses rely on *Open Source Software (OSS)* [2]. The fact that OSS can be freely inspected, copied, modified, and redistributed played a crucial role in the growth of digital technologies and thus gave rise to a billion-euro economy [3]. *OSH*, on the other hand, is far less prevalent. Today, thousands of designers and makers are already freely sharing their hardware designs, reusing and adapting existing solutions, and discussing new ideas online. Unfortunately, the trend of open-sourcing designs and documentation for hardware has not caught on with industries. The OSH movement still suffers from a lack of accepted standards and tools for developing, documenting, and sharing hardware creations. Furthermore, the unclear legal implications of exploiting open hardware discourage adoption by industries. There is a high demand for renewable energies and other sustainable technologies [4] where the concept of open source may be a key enabler for future technological advancements. As such OSH can have a substantial positive impact and may help solve today's pressing environmental and economic matters.

There are quite a few platforms for publishing and sharing hardware designs and documentation online. The problem lies in the poor connections between these platforms and their communities. It is currently not conveniently possible to share creations between platforms and to search across platforms to find desired hardware designs. To draw the comparison with OSS, there is no package manager that allows downloading and managing product designs. Thus, makers cannot easily find already existing hardware solutions and cannot effectively adopt, use and improve OSH technologies. With this in mind, the *Open Know How (OKH)* and *Library of Open Source Hardware (LOSH)* specifications were created. Their goal is to improve the discoverability and portability of knowledge on making open hardware [5], [6].

The goals of the LOSH project are twofold. Firstly, define a standard to describe products with metadata, and secondly, create a public semantic knowledge base and search engine for OSH products. The project aim is to collect product metadata and technical documentation from various platforms and make these immediately searchable and cross-referenceable [6]. The LOSH project efforts led to a web service being created that served as a demonstrator. The service collected product metadata from multiple platforms and offered a very basic web interface for searching through the catalog of collected data. The demonstrator met the basic requirements and was, to some degree, useful as a search service. Furthermore, it contained enough high-quality OSH designs to be helpful in technology research, but the overall functionality and user experience leave much to be desired [7]. Although the demonstrator showed some usefulness, it lacks the capabilities that users expect from a fully featured search engine. Furthermore, the usability of the demonstrator is rather poor, and the

user experience is not that good either. Lastly, as the system architecture and technological choices are unsuitable, it would require significant efforts to improve the system.

In this work, we created a semantic knowledge base and search engine for OSH. The goal was to create a service that is easy and satisfying to use, improves the discoverability of open hardware, and presents value to the OSH community. For the search engine to have a chance for adoption by the community, we implemented an advanced search functionality to grant access to the full set of product information with an appealing easy-to-use web interface. We conducted expert interviews and user studies with volunteers from the OSH community to assess system requirements and evaluate the usability and user experience of the final service.

1.2. Problem Statement

Creating a knowledgeable base and search engine comes with a variety of challenges. The main goal of the thesis is to provide a functional, fairly usable search service with good chances of adoption by the OSH community. As such, the thesis focuses on aspects and challenges that are fundamental to the success of the service. At the center of research and development are usability and user experience.

A service that allows a user to effectively, efficiently and satisfyingly achieve a specific goal demonstrates good usability. In addition, if the service also behaves as expected and subjectively works well for the users, then the service also exemplifies a good user experience. Good usability and user experience are necessities for the success and adoption of the search service that is to be created as part of this thesis. Good usability and user experience alone do not guarantee success but improve the chances of target groups recognizing its value and adopting the service. Previous works on the LOSH project were rather concerned with functionality than the user's experience. An important aspect of this work will be to identify end users' needs, expectations, and how the users will use the service. The results must be taken into consideration when designing and implementing the service. Furthermore, to confirm reasonably good usability and user experience, the final product must be tested with actual users under real-world conditions.

Various platforms allow sharing of ideas and hardware designs openly. Each platform has its philosophy, scope, and data formats. In order to create a rich, overarching dataset, a common vocabulary and semantics must be established. The LOSH specification covers a large portion of use cases, but it has to be adapted to the requirements of the thesis. Once a common basis has been established, the platforms' differences must be dealt with. This involves a mapping of data fields and inferring missing information.

Documents on the public web are typically geared towards human consumption and interaction and thus are not easily readable and comprehensible by machines. *Semantic Web (SW)* aims to make these documents more easily machine-readable and available to automated processes by extending them with standardized metadata. This metadata is expressed through the *Resource Description Framework (RDF)*. RDF encodes the information as graph data, thereby opening up the possibility to link data entities together, even across the boundaries of unrelated datasets [8]. The value of such linked data becomes apparent once the web of interconnections is sufficiently dense. In such a net, previously invisible connections can become evident and reveal hidden truths. This is especially interesting in the field of OSH. Cross-linking products, users, and other entities might help to answer

complex questions and empower more research on the topic of OSH. The thesis aims to lay the groundwork for future research by providing all data also through SW technologies.

Web search engines such as Google perform a regular crawl of large portions of the public internet. The goal of the crawler is to find and select relevant information, store it in an index and make it easily searchable. Sites need to be revisited to check for changes and update the index accordingly. Every time a website is visited, the crawler must ensure not to overwhelm the content-providing service. These properties are defined as crawler policies that every kind of crawler must abide by. The crawler in this work may only have to visit and search a few selected platforms, but the search space is still quite large and is subject to growth. Despite the amount, newly created projects need to be found in a reasonable timeframe and kept up-to-date. While searching for OSH products, the number of requests and the request frequency to individual platforms need to be limited. Developing and optimizing policies for the presented use case of the thesis will require research and real-world trials.

2. Background

2.1. Open Source Hardware

Open Source Hardware (OSH) or simply *Open Hardware* is defined by the *Open Source Hardware Association (OSHWA)*¹ as physical artifacts — tools, machines, devices, or other physical things — “whose design is made publicly available so that anyone can study, modify, distribute, make, and sell the design or hardware based on that design.” [10]

There is no uniform standard for publishing hardware designs and documentation. Even so, there are some guidelines and recommendations for publishing OSH. Ideally, the hardware plans are made available in formats that allow modifications, readily-available materials and components are used, and the hardware is built on standard processes and open infrastructure. Furthermore, to remove the restrictions of proprietary components and software and ensure the best possible adaptability and compatibility, it is advised to use free and open-source design tools.

OSHWA created a guideline to help develop and evaluate licenses for OSH. It provides the OSH community with an easy way to publish OSH and establish a legal basis for reuse and distribution. Their definition entails information on how to document the hardware, what to include, how to attribute derivations, and more [10]. Other projects, such as the standardization effort by the *DIN SPEC 3105*, extend the previous OSHWA definition to address criteria for the technical documentation of open source hardware and introduce a community-based assessment scheme for independent verification [11]. All these efforts may eventually lead to OSH being more widely accepted and adopted by the industry.

Just like OSS gave rise to a billion-euro economy [3], the OSH paradigm can also have a substantial impact. Proprietary ownership is often a hindrance when it comes to innovations. Patents and lawsuits can dramatically impede technological advancement, which is especially bad regarding renewable energies and sustainable technologies. Sharing knowledge, resources, and blueprints while encouraging commerce through the opened-up exchange might be the key to solving pressing environmental and economic problems [1].

¹ The Open Source Hardware Association (OSHWA) (oshwa.org) is a non-profit organization that promotes the idea of Open Source Hardware. They are maintaining a certification program based on their Open Source Hardware Definition. Furthermore, they foster cooperation with other OSH movements and organizations by organizing conferences and community events [9].

2.2. Semantic Web

The dataset of OSH products is not only interesting in a use case such as a search engine but also for other applications as well. If the dataset could be connected to other datasets on the web to create a comprehensive, distributed knowledge base, it would be possible to find meaning and relationships beyond the scope of the data in itself. Other services and especially research can benefit from such a setup. This is where the concept of *Semantic Web (SW)* comes into play.

SW is a concept to turn the classic web of documents into a web of data. Classic document formats on the web are usually more geared towards human consumption and interaction. The idea of the SW is to make these documents more easily machine-readable and available to automated processes by extending them with standardized metadata. The definition of standards and technological developments on that topic is led by the *Wide Web Consortium (W3C)* and supported by researchers and industrial partners around the globe [12].

HTML, PDF, and other document formats found on the web are primarily meant to be read by human users. HTML offers a structure for data presentation, but it lacks data semantics and thus cannot easily be interpreted by machines. Web crawlers that scour the web to harvest information usually employ web scraping or some form of text mining. By encoding semantics and metadata, robots do not have to rely on that kind of data collection. Moreover, they can avoid the uncertainties and ambiguities in dealing with free-form texts. In the context of the SW, technologies such as the Resource Description Framework (RDF) and the Web Ontology Language (OWL) were created to formally represent metadata. RDF is used as a model for describing and interchanging graph data. OWL, on the other hand, is used to describe concepts, categorize objects, and express relationships between those objects [8].

Most of the data is stored in silos, isolated from the rest of the world. It is argued that the globally interlinking databases across the internet will unfold a huge potential and allow insights into hidden relations within the data. The first step is to make the dataset available using SW technologies such as RDF. The second step is to link related data together to create an actual "web" of data. This is called *Linked Data (LD)* or *Linked Open Data* when the data is openly accessible. Linked Data allows to explore relationships between data entities through semantic queries. Links are expressed by Uniform Resource Identifiers (URIs). Tim Berners-Lee, inventor of the World Wide Web and director of the W3C, names four rules for LD URIs [13], [14]:

- "Use URIs as names for things"
- "Use HTTP URIs so that people can look up those names"
- "When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL)"
- "Include links to other URIs. so that they can discover more things"

In practice, access to LD is usually provided in the following fashion. When a resource is requested from a service that offers LD access, the first thing happening is the URI gets dereferenced. Human readers requesting the information in HTML form will be redirected to a conventional web page. On the contrary, machines requesting an RDF representation will be redirected and prompted with the data in serialized RDF form. This allows traditional web browsers and semantic web browsers to

be used alike. An excellent example of a LD service is *DBpedia*². DBpedia holds structured content extracted from various *Wikimedia*³ projects and offers an interface for retrieving information as Linked Data. The sheer amount of available information on *Wikipedia* alone and the high degree of interconnections allow for a detailed exploration of any topic imaginable.

² <https://dbpedia.org/>

³ <https://www.wikimedia.org>

3. Related Work

3.1. Search Engines

Search engines provide an effective and efficient way of finding relevant information in large datasets. The best-known use for search engines is finding information and documents on the web. Search engines such as *Google*¹ or *DuckDuckGo*² scan large parts of the public internet and provide a publicly available search index for everyone to use. Other search engines for other purposes and domains exist. Some of these provide search functionality for specific pages and datasets, like classic libraries that provide a search for their stock of books, magazines, and other publications. Another example of a specialized search engine is the *GitHub*³ code search, which returns matching source code on the *GitHub* platform. The search engine presented in this work is tailored towards OSH from various hardware hosting platforms.

[15] classifies search engines into the following types:

- **Crawler-Based Search Engines:** Crawler-based search engines are the most common type of search engine found on the web. A crawler, also referred to as a Spider or Bot, is a program that automatically scours the web, collects information, and feeds them into an index. The crawler also keeps the search index up-to-date by revisiting websites. The search engine then uses the index to retrieve and rank relevant results. Crawler-based search engines are a perfect fit for the constantly changing and evolving web. Good examples of this type of search engine are *Google* or *DuckDuckGo*.

This type of search engine will be developed and deployed in this work.

- **Human-Powered Directories:** Human-powered directories, as the name suggests, are a type of search engine that rely on a human-created index. Manual human labor is required to perform any additions or updates to the index. These types of search engines can be found in classic libraries.
- **Hybrid Search Engines:** Hybrid search engines combine a crawler-based search engine with a human-powered directory. An index is created automatically with human intervention and quality control.
- **Meta Search Engines:** Meta search engines are aggregates that rely on results of other search engines. They combine the results of a search, rank them anew, and present them to the user. An example of a meta search engine is *StartPage*⁴.

¹ <https://www.google.com>

² <https://duckduckgo.com>

³ <https://github.com>

⁴ <https://www.startpage.com>

A search for a specific thing is defined by a search query. A search query is a phrase, a combination of keywords and operators used to retrieve specific information from a dataset. The syntax and capabilities of said search query varies from search engine to search engine. While some engines only support keyword-based text search, others have an advanced syntax and allow for complex queries. A lot of search engines provide what is called search operators. These operators can be used to refine the search results. For example, Google's `site:` operator can be used to limit the search results to a specific website.

3.2. Open Hardware Observatory

The *Open Hardware Observatory (OHO)* is the conceptional predecessor to the Library of Open Source Hardware (LOSH) and laid the groundwork for OSH-centered meta-search engines. It originated as an initiative by a member of the non-profit organization *Open Source Ecology Germany e.V.* 2019 it was taken over by the *Technische Universität Berlin*, and since 2021, the newly founded non-profit association *OHO - Open Hardware Observatory e.V.* has been taking care of the project [16].

Its original purpose was to provide a mechanical and electronic hardware search engine. A web crawler was developed and deployed that searched the web for keywords related to *do-it-yourself* (DIY) and *Open Hardware*. The results were categorized in one of ~ 500 categories and made available through a web interface. The resulting dataset was curated through partly automatic and partly manual quality control. The goal was to make the thousands of DIY and OSH projects worldwide searchable and thus available for anyone to use, adapt and improve [16].

Since the Technische Universität Berlin and later the newly founded OHO association took over, the goals shifted towards supporting, improving, and publishing sustainable Open Hardware projects. OHO recognizes the immense potential that OSH and DIY can have for sustainable development. Most OSH projects do not properly publish blueprints, material lists, instructions for assembly, and other crucial information. The documentation is often lacking, which makes it very hard to replicate and adapt the projects. The OHO association sees a great demand for low-cost and proven solutions, especially in developing countries, where access to affordable products and technologies is rather limited. Therefore, OHO tries to provide a search for DIY and Open Hardware projects, create, improve and publish blueprints for selected promising projects and offer certification [17].

Due to the strong focus on DIY projects and lax policies regarding mandatory project information, the OHO index still contains a lot of projects without a license and other legal information. This poses no issue for private reuse, but it hinders adoption by small businesses and companies, where legal aspects have greater importance.

3.3. Open Know-How Specification

The *Open Know How (OKH)* specification (OKHv1) is an attempt to improve the openness of know-how for making hardware by improving discoverability, portability, and interactivity of knowledge [18]. The specification defines a metadata standard for OSH and is developed by the Open Know-How Working Group, which is part of the *Internet of Production (IoP)* alliance. The IoP alliance represents a group of people and organizations who believe in a decentralized manufacturing and shared

knowledge approach for shaping the fabrication and production of goods in the future. The idea is to build a sustainable system that is globally networked and locally executed. As a result, people shall be enabled to create products from locally sourced materials and components from global designs. As steps towards these goals, processes and open-source tools are being developed that empower makers and designers to document and share their creations easily. The OKH specification is one of those instruments [19].

The goals for the OKH project are threefold [5], [20]:

1. Discoverability: Allow documentation of OSH to be found regardless of where it resides on the internet. This goal is addressed by creating the OKH Specification, which in turn, can be utilized to develop web crawlers to discover hardware documentation.
2. Portability: Allow hardware documentation to be shared between different content platforms and transferred from one platform to another. Part of the solution to this goal is creating an accepted standard for Open Hardware metadata.
3. Interactivity: Allow know-how about making hardware to be created, updated, and published without centralized control.

As part of the efforts, a simple *Open Know-How Search*⁵ was developed. Users can enlist their hardware product by publishing metadata in the OKH standard on one of the supported community sites *Appropedia*⁶, *Field Ready*⁷ or *GOSH Community*⁸. The service collects the metadata and offers a simple web interface for searching through the products.

The OKH specification is not widely adopted yet and lacks real-world empirical results for validating its claims. Moreover, the offered search engine is in a prototype stage and has a rather limited set of features. The software is written in Node.js, which is outside of our expertise; thus, it is not used as a basis for our work.

3.4. OPENNEXT - Library of Open Source Hardware

The *Library of Open Source Hardware (LOSH) Specification* is a fork of the OKH Specification. It is an effort to incorporate current research results and make the specification applicable to *Linked Open Data*. The project was initiated in 2020 by the *OPENNEXT* project [6].

OPENNEXT is a research project of 19 industry and academic partners across Europe and has received funding from the European Union's Horizon 2020 research and innovation program. *OPENNEXT* aims to bring the principles of open-source closer to the development and distribution of products and goods. Small and medium enterprises shall be enabled to work more closely with consumers, makers, and other communities. *OSH* is deemed to be the basic building block to success. The intention is to release product designs as open source, allowing anyone to study, modify, share, and redistribute copies. The idea itself is nothing new and well-established practice in Open Source Software (OSS). Open-Sourcing designs and documentation can reduce proprietary vendor lock-in,

⁵ <https://search.openknowhow.org>

⁶ <https://www.appropedia.org>

⁷ <https://field-ready-projects.openknowhow.org>

⁸ <https://projects.openhardware.science>

planned obsolescence, and waste of resources and also empower novel business models. As stated before, the term open-source doesn't imply free-of-charge. Businesses shall still be able to earn a living with their work [1], [6].

The efforts of OPENNEXT in the LOSH project are very similar to the OKH project. One of the differences is the strong emphasis on legal aspects such as proper licensing to enable commercial exploitation of the designs. Many hardware creators freely share their designs online but do not supply a license, or they license their hardware under conditions prohibiting commercial use. This is a hindrance to the growth of OSH and one of the reasons open hardware products are not widely accepted throughout industries. As a consequence, potential users that do not have a background in OSH have a relatively high hurdle to overcome to find suitable designs that can be legally adapted, commercially produced, and sold. To address some of these issues, the following two solutions were proposed and developed in the context of the LOSH project [6], [21]:

- **LOSH Specification:** The specification is based on the OKH specification, incorporates current research findings, and ensures compatibility with LD. Like the OKH specification, the LOSH specification is also a standard to describe OSH products with metadata. It is different from OKHv1 in that it tries to merge datasets from different platforms, whereas the OKHv1 approach is one-size-fits-all.
- **Semantic Knowledge Base and Search Engine:** Powering a distributed database and offering a knowledge base and search engine for OSH. The service shall enable users to search across various hardware hosting platforms. Furthermore, the users shall also be able to access any product information as LD. To ensure a minimal quality standard, only products are considered that conform to the specification. The authors must supply mandatory information such as the author's name, license, and other legal information. The service is not intended to replace existing hardware hosting platforms but as a tool to bridge the gaps between them.

In the course of the LOSH project, a demonstrator⁹ of the semantic knowledge base and search engine was created. The demonstrator is based on *Wikibase*. Wikibase is developed by Wikimedia Deutschland and comprises a set of MediaWiki extensions and other third-party tools. It is used to build knowledge bases for the semantic web [22]. Like the previously mentioned OHO Search Engine, LOSH also consists of a database, an interactive web interface, and a crawler component. The demonstrator provides only a minimal set of features, which is just enough for a proof of concept. Table 3.1 contains a description of all components. The interaction of the components is depicted in Figure 3.1.

Component	Description
Krawler	<i>Krawler</i> ¹⁰ is the crawler component, which searches through platforms such as Wikifactory and GitHub, downloads product metadata, parses and sanitizes it, and converts it into an RDF format for uploading it into the Wikibase database. The Krawler application is developed as part of the LOSH project and is written in Python.

Table 3.1.: LOSH Demonstrator Components

⁹ <https://losh.opennext.eu>

Component	Description
LOSH Backend	<p>The <i>LOSH Backend</i>¹¹ provides search capabilities and data access to the <i>LOSH Frontend</i>. When a search is issued, the LOSH Backend first performs a full-text search using the ElasticSearch database to find matching products. In the second step, the full product details are requested from the Wikibase database for each result found in the first step. The entire result set is then exposed over a GraphQL API and displayed via the LOSH Frontend.</p> <p>The LOSH Backend application is developed as part of the LOSH project and is written in Node.js.</p>
LOSH Frontend	<p>The <i>LOSH Frontend</i>¹¹ provides the web interface for users to search for hardware products. It uses the LOSH Backend application to perform the search and receive results.</p> <p>The LOSH Frontend application is developed as part of the LOSH project and is written in Node.js.</p>
Wikibase	<p>Wikibase¹² is a set of MediaWiki extensions for storing and managing semi-structured data. It is used to build knowledge bases for the semantic web. Like MediaWiki itself, the extensions are also developed by Wikimedia. Most notably, two extensions are of relevance in the context of the LOSH project. First, the <i>Wikibase Repository</i> which is used as a structured data repository and allows editing and storing of data. Second, the <i>Wikibase Client</i> which is used to display data from the <i>Wikibase Repository</i> via Lua modules or parser functions. The data is stored in the same relational database as the MediaWiki content but is managed solely by the Wikibase extensions.</p>
WDQS Backend	<p>The Wikidata Query Service (WDQS) Backend¹³ is triplestore database based on Blazegraph and developed by Wikimedia. It receives a copy of the data stored in the Wikibase dataset and offers a SPARQL endpoint for querying the data.</p>
WDQS Frontend	<p>The frontend for Wikidata Query Service¹³ provides a web-based UI for the WDQS Backend. It allows users to run SPARQL queries against the dataset.</p>
MariaDB	<p>MariaDB¹⁴ is a community-developed fork of the MySQL relational database management system (RDBMS). The database is primarily used as a backing store for Wikibase/MediaWiki. All the dynamic contents of MediaWiki are stored within the MariaDB database together with the semi-structured data of Wikibase.</p>
ElasticSearch	<p>ElasticSearch¹⁵ is a document-based database used for efficient full-text text search. In the project context, the database is used for full-text search on the various text-based product properties like product name or description. For it to work, the LOSH dataset is copied from Wikibase into the ElasticSearch database, and in turn an efficient search index is created. The downside is, of course, the need for data duplication and higher resource demands.</p>

Redis

Redis¹⁶ is an in-memory key-value database mainly used for caching and message brokering. In the LOSH demonstrator, it is used to cache rendered HTML pages of MediaWiki and thus improve access times. It is not used for the LOSH Frontend and therefore doesn't contribute much to the project.

Table 3.1.: LOSH Demonstrator Components

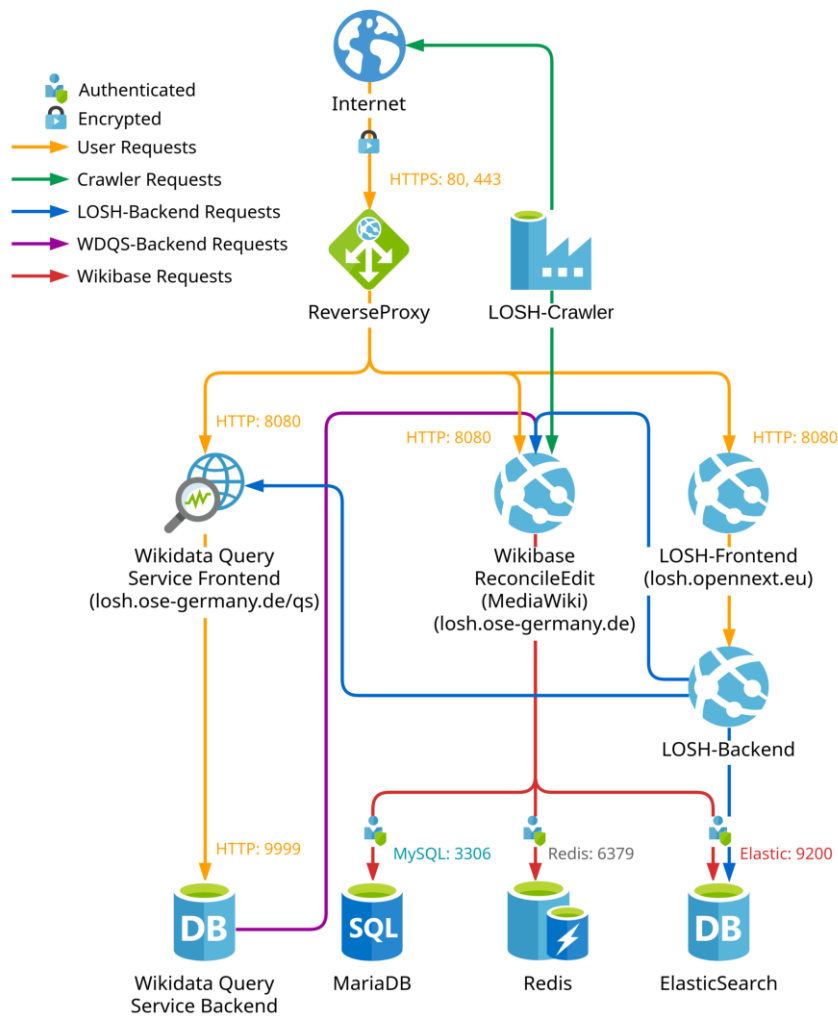


Figure 3.1.: LOSH Demonstrator Network Diagram

¹⁰ <https://github.com/OPEN-NEXT/LOSH-Krawler>
¹¹ <https://github.com/wmde/LOSH-Frontend>
¹² <https://wikiba.se>
¹³ https://www.mediawiki.org/wiki/Wikidata_Query_Service
¹⁴ <https://mariadb.com>
¹⁵ <https://www.elastic.co/elasticsearch/>
¹⁶ <https://redis.io>

4. Requirements Analysis

4.1. Introduction

Before software can be designed and successfully implemented, a requirements analysis needs to be performed. The requirements analysis is used to capture the stakeholder's wishes and expectations of the software to be built or modified. The process involves analyzing, documenting, and validating the system requirements. The result of these efforts is the *System Requirements Specification* (SRS). A fully featured SRS should contain a detailed description of the purpose of the system, its functionality, how it should look and feel, how it should perform, and any other system parameters that deem important. The goal is to provide a document that can be used as ground truth for the entirety of the development process.

For this thesis, a simplified, lightweight SRS has been chosen instead of a fully dressed one. Each requirement is described in a brief paragraph stating the problem and main success scenario.

The requirements analysis in this thesis was performed using two methods. First, a feral study of existing and similar software solutions was carried out, and second, interviews with OSH community members were conducted. Studying existing search engines and hardware hosting platforms revealed valuable insights into general usability, user experience, and best practices regarding web crawling. The interviews as well yielded practical and sometimes unexpected requirements and wishes that have been taken into account.

4.2. Requirements Interview

Interviews with OSH community members were conducted to capture the requirements and wishes of potential future users of the LOSH service. The goal was to directly involve the community and end users in the development process to get a general idea of how the final service would be used.

The interviews were conducted online in a face-to-face meeting using a semi-structured style. The interviewees were first tasked to perform search queries using the existing OPENNEXT LOSH demonstrator. Each participant was asked to "think out loud" while performing the tasks, revealing their experiences, struggles, and expectations. After each challenge, qualitative questions were asked to spawn a conversation and get deeper insights into specific aspects of the future service. Once the challenges concluded, and the participants had a relatively good understanding of what the project entails, they were asked to state their opinions regarding the planned features and usage behavior. The following list contains an excerpt of the questions asked. The full set of questions used as a guide to the interview can be found in Section C the appendices.

-
- *What OSH Platforms have you heard of and used before?*
 - *What would you use the new LOSH service for, and what do you think other users would use it for?*
 - *What do you expect of an overview page of a product?*
 - *What importance does versioning play and should the service index all available versions and make them searchable?*

The interviews were evaluated by defining keywords, categorizing the results, and phrasing short requirement descriptions. The results are presented in Section 4.4. There, each requirement is grouped into components and topics. For requirements that are derived from the interviews, the context, i.e. the stated problem and expectations, is preserved as part of the description.

4.3. Terminology and Conventions

This section describes the standards, typographical conventions, and terminology used to document the requirements.

Categories with requirements articulated by one or more interviewees are split into two sections. The requirements stemming from an interview are placed first. Those also contain additional context from the interviews themselves and have references (initials) to the participants who stated them, e.g., **JP**.

Requirements are captured as either mandatory or optional. The following terms are used in that regard:

- **SHALL:** Requirements marked with *SHALL* are mandatory and must be implemented. They make or break the functionality of the service.
- **SHOULD:** Requirements marked with *SHOULD* are optional. They are either nice-to-haves or simply not feasible to implement in the given time frame.

The following terms are used to describe a fact:

- **Product:** *Product* denotes an tangible OSH product.
- **Full-Text Search:** *Full-Text Search* refers to searching for words in a comprehensive text data source. A match is found if the source text contains any or all the words from the query. The text is preprocessed to unify different word forms and tenses to be more effective. The preprocessing includes the following:
 - split the text into chunks (tokens)
 - convert tokens into lowercase
 - perform Unicode normalization
 - reduce words to their root form (stemming)
 - remove insignificant stop words
- **Service:** *Service* denotes the web service to be developed as part of this thesis.

-
- **Robot:** *Robot* denotes a machine (automated program) that queries the service on a user's behalf.
 - **User:** *User* refers to a human that uses a web browser to access the service.
 - **Client:** A *Client* is either a human user or a robot that uses HTTP to request resources from the service.
 - **Platform:** A *Platform* represents a place for users to document and discuss their product creations (e.g. Wikifactory). The platforms are to be indexed and made searchable using the service.
 - **Web Interface:** The *Web Interface* provides users access to the resources stored in the database. The interface takes the user input and returns an HTML document containing the requested information.

4.4. Requirements

4.4.1. Crawler

The *Crawler* is the component that scours the content hosting platforms for OSH products.

Indexing

- R1** Index Update: New products are created daily, and new versions of existing products are released frequently. Therefore, the crawler *SHALL* automatically and regularly update the index. Previously indexed product releases *SHALL* be checked for changes as well, since some properties may have changed. **NW** gave the example of revoking an attestation according to DIN SPEC 3105-2, which needs to be reflected in the search index.
- R2** Index Deletion: There is no uniform consensus regarding handling deleted products. If a product creator decides to delete their creation, then the service needs to handle this case the next time it tries to update the product information. **FR** suggested deleting products from the index which are not mentioned/referenced in other products and keeping the rest. **MH**, on the other hand, said to delete missing products from the index because the service is not intended as an archive. **JP**, **OS**, **PJ** and **TW** favor keeping deleted entries in the index, mark them as *missing* and hide them in the search per default. If desired, the user could include the missing entries by adding the appropriate search operator. The service *SHOULD* keep deleted entries for a period of time and mark them as absent. After the grace period, the entry *SHOULD* be deleted from the index.
- R3** Versioning: Almost all interviewees see a high potential in including version information for each product. A user could monitor changes over time, which may indicate the activeness of a project, get feedback on where and what versions are used most often, and see if a component is still compatible in a particular version. Also, **NW** brought up an issue regarding attestation, where versioning plays a crucial role. It is not unlikely that only a few versions of a product got actually attested. Knowledge about versions in this context is, therefore, an absolute requirement. Thus, the crawler *SHALL* include all versions when indexing the product platforms.

-
- R4** Automated Crawling: The crawler *SHALL* be able to find and index products automatically. This means it *SHALL* run as a continuous job, go through the content of each platform, extract metadata of each found product and add it to the database.
- R5** The crawler *SHALL* be able to discover and collect designs that provide an OKH or LOSH manifest. This needs to be done in a timely manner to detect newly published products in a reasonable time frame.
- R6** Multiple Platform Support: Each product platform has unique traits, scope, and data formats. To support searching multiple platforms, a common data representation *SHALL* be defined. Subsequently, a mapping of the platform data model to the model of the service *SHALL* be created. If possible, any missing data fields *SHALL* be inferred from the available set of information. The goal is to support all major hardware hosting platforms. For the sake of this thesis, at least the following platforms *SHOULD* be supported:
- Wikifactory
 - OSHWA Certification List
 - GitHub
- R7** Selection Policy: Some platforms may contain large amounts of content, from which only a subset may be hardware related. A selection policy for each platform *SHALL* be defined and implemented to pre-select possibly interesting content to crawl. The selection policy is required to guarantee a reasonable crawling performance.
- R8** Re-Visit Policy: Already indexed products need to be revisited to check if any updates are available. A re-visit policy *SHALL* be defined and implemented that strikes a good balance between up-to-dateness and performance.
- R9** Politeness Policy: The crawler needs to ensure not to overwhelm any platform with thousands of data-intensive queries in a short time. Best practice dictates that the speed and number of requests per time *SHALL* be limited, and the crawling activity be throttled.
- R9-1** Respect robots.txt: The robots.txt can be used by website owners to define what pages web crawlers are allowed to access. The LOSH crawler *SHALL* respect the instructions given in the robots.txt.
- R10** Parallelization Policy: To scale the service and include support for more platforms, more than one crawler instance needs to be executed in parallel. Multiple platforms *SHALL* be crawled in parallel, but each platform shall only be accessed by one crawler instance at a time. Ergo, a policy *SHOULD* be defined and implemented to coordinate multiple crawler instances.
- R11** Extendability: The architecture of the crawler *SHALL* be easily extendible so that support for other platforms can be added in the future.
- R12** Low Ressource Usage: CPU, RAM, Network, and other system resources *SHALL* not be wasted. The footprint of the application *SHALL* be as small as possible so that it can also be run on low-spec hardware. Consequently, we *SHOULD* refrain from using a mix of different programming languages and technologies, which often wastes resources. Sticking to a small number of technologies and dependencies also makes it easier to port to a newer system configuration and overall easier to deploy the application.

Security

- R13** Sanitize Inputs: The crawler inherently works on untrusted user-provided input data. There are several ways of exploiting the service and causing harm by injecting code or other arbitrary data that is out of spec. Therefore, the crawler *SHALL* implement appropriate measures to defend against such threats to protect the public database and other parts of the infrastructure.
- R14** DOS: To ensure stable service operation and uptime, the service *SHALL* be protected against some sort of Denial of Service (DOS). For example, DOS can be the result of triggering particular resource and time-consuming server tasks. For protection against these kinds of threats, appropriate countermeasures *SHALL* be implemented. The measures should include enforcing restrictions (request size, request execution time, etc.), using caching extensively, and handling carefully.
- R15** Non-Related Media: The service *SHALL* detect and prevent misuse in form of storing non-related media and arbitrary data in the index.

Privacy

- R16** Control Over Data: The users own the data they provide with their products. Therefore users *SHOULD* be able to remove their information from the search index database by either updating the corresponding products or other means.
- R17** Malicious Content: Not all users have good intentions. Some users might try to misuse the service to spread malicious information and content. Such content *SHOULD* be identified and removed from the index.
- R18** Unintentional Data Exposure: A user might expose privacy sensitive information such as phone number or location into their documentations. These could be misused by scammers and *SHOULD* be proactively redacted before being stored in the index.

Administration

- R19** Configuration: The application *SHALL* be easily configurable using the YAML file format. It *SHALL* feature configuration validation and the generation of a default configuration. Making the configuration easily readable and testable reduces the chances of a faulty configuration in production and, thus, a more robust service operation.
- R20** Deployment: The service components *SHALL* be easy to deploy, regardless what hardware platform or operating system is used. For portability and deployment in a clustered environment, the service components *SHALL* also be provided in a containerized format (e.g. Docker).
- R21** Logging: A stable and performant operation needs to be ensured. To do so, the service *SHALL* provide configurable logging functionality. Proper logging can help to identify issues and provide insights into the service state.
- R22** Monitoring And Performance Metrics: The service *SHOULD* offer a monitoring endpoint and a comprehensive set of performance metrics. This is to measure the system state, health, and potential bottlenecks. Some metrics that should be considered:

-
- Runtime
 - CPU Usage
 - RAM Usage
 - Network Usage
 - download rate/amount
 - upload rate/amount
 - Crawled Products
 - Number of indexed products per second
 - Number of total indexed products
 - Number of added products
 - Number of deleted products
 - Number of updated products
 - Number of valid products
 - Number of erroneous products
 - Errors

4.4.2. Product Search

The *Product Search* refers to the web interface that the user uses to search for desired products.

Search Query

A *Search Query* is a phrase, a combination of keywords and operators used to retrieve specific information from a data set. The user formulates a query to find something of interest, and the search engine returns matching results. There is no defined standard when it comes to Search Queries. Each search engine, such as *Google* or *DuckDuckGo*, has a query syntax with a varying set of features. Full-text search is the most common and most well-known feature. Most search engines also implement search operators to refine the results. For example, Google's *site:* operator allows searching for information on a specific site.

A simple yet powerful query syntax that allows complex queries to be formed is desired. However, due to technical limitations, the complexity must be limited to a reasonable degree to guarantee a performant operation.

R23 The interviewees desire a full-text search comparable to the search found in Google and other search engines. For this reason, a full-text search *SHALL* be offered for text-based product properties and assumed to be full-text searchable (e.g. product name or description). A full-text search without explicitly specifying the property to search for *SHALL* use the product's name, description, category, and tags to find results.

R24 Performance is a factor that cannot be ignored. **FR** stated that search results should be available shortly after the request was issued. A user usually performs more than one query. They start with simple queries to get a general idea about the search space and then add filters to trim down the results. Long wait times result in a poor user experience. Most users likely would stop using the service, if they had to wait for a painfully long time. Consequently, results for search queries *SHALL* return in a reasonable time frame. The mean time for search queries *SHALL* be lower than 1 second.

R25 To ensure the performance goal, the complexity of the search query *SHALL* be limited. The following limits are to be enforced:

R25-1 maximum of 500 characters in the query string (excluding whitespaces)

R25-2 maximum of 15 terms in a logical combination (AND/OR/NOT)

R25-3 maximum of 30 words for full-text search

R25-4 maximum of 10 wildcards

Search Operators / Search Filters

Search Operators are special symbols and commands that extend the regular full-text search and add other capabilities. Many websites and search engines offer search operators with varying extents of capabilities. Usually some form of logical (and/or/not), relational (= / != / < / >), navigational (search in specific page) and other types of queries are supported. For example, a simple Google search query car can be extended with a navigational query site:wikipedia.org to find cars specifically on Wikipedia.org. Search Operators are a powerful tool to create complex queries and make search results way more precise.

Search Filters are used to refine the search by removing unwanted items from the results list. Because they can be implemented as operators, they are not treated as a separate concept. In addition to specifying filters via the search query, they can also be made available through select boxes and other input elements.

The following is a list of web services that offer advanced search capabilities. These stood the test of time and can be used as a reference when defining and implementing the operators:

- GitHub: <https://docs.github.com/en/search-github>
- Google: <https://support.google.com/websearch/answer/2466433>
- DuckDuckGo: <https://help.duckduckgo.com/duckduckgo-help-pages/results/syntax>
- Slack: <https://slack.com/intl/en-gb/help/articles/202528808-Search-in-Slack>
- Zendesk: <https://support.zendesk.com/hc/en-us/articles/4408886879258>

R26 All interviewees expressed their need for filtering results beyond the ability to search products by full-text. The consensus is the more filter options, the better. While professional developers might be more interested in the development standards and legal aspects, DIY developers are more likely to search for products by appeal and complexity. As examples of good filter implementations, the following sites were mentioned:

-
- LapStore (**FR**)
 - HardwareSchotte (**FR**)
 - Geizhals (**JP**)
 - Ebay (**JP**)

Therefore the service *SHALL* offer the ability to filter products by various properties. The following properties are desired to be searchable/filterable:

- context/category/tag (**FR, JP, TW**)
- difficulty level for DIY (**JP**)
- specific license (**JP, NW, TW**)
- is a dedicated website available (**FR**)
- where/how is it used (**FR**)
- how often is it used (**FR**)
- active/inactive/archived/deprecated/missing (**JP, OS, TW**)
- documentation language (**JP**)
- description (**JP**)
- documentation and technology readiness levels (ODRL/OTRL) (**JP, MH, NW**)
- certification/attestation/standard (**MH, NW, TW**)
- version (**MH, PJ**)
- published/created date (**PJ**)
- number of forks (**OS**)
- content platform (**PJ, TW**)
- is the product fully open-source or does it contain proprietary components / uses only open formats (**PJ, TW**)
- compatible with Git (**TW**)
- organization (**TW**)
- professional/educational/DIY (**TW**)
- prototype/mass produced (**TW**)
- what operating system the included software is usable on (e.g. Linux) (**TW**)

R27 The interviewees welcome the option to select and combine more than one filter at a time. There were use cases described that require logical combinations of filters. **OS** mentioned the option to use operators and a search syntax similar to the Google search syntax. So, in addition to the required search operators, the service *SHALL* also include filtering using boolean logic (and/or/not). (**JP, MH, NW, OS**).

-
- R28** When using discrete inputs for each filter, such as select boxes, for example, **MH** suggests displaying the number of remaining elements after filtering right beside the input elements. This could help the user decide which filters to employ to reduce the number of results and find the desired products. Showing aggregated results for each filter is nice-to-have and *SHOULD* therefore be implemented if possible.
- R29** There are use cases that require the same search query to be rerun at a later point in time. For example, one could formulate a query and evaluate the search results over an extended period of time to perform a trend analysis. **JP** gave the example of storing search query URLs in bookmarks and revisiting them again later on. A different issue was brought up by **NW**, which can be resolved by encoding the search query in the URL as well. He liked the possibility of reloading the search page and keeping the previously issued search. Accordingly, the service *SHALL* encode the search query as *query parameters* in the URL, so they can be easily bookmarked, copied, shared, and rerun at a later point in time.
- R30** **JP** and **NW** pointed out that the license filter in the demonstrator is not very descriptive, and it is unclear what the difference between a "Permissive" and a "Weak Copyleft" license is. They proposed to give hints about the meaning and usage of each filter. The service therefore *SHALL* explain and optionally give a usage example for each filter. For more in-depth details, there *SHOULD* also be a reference to the syntax documentation.
- R31** **FR** expressed the difficulty of finding the desired products because of search term ambiguity. Because of unknown user intention and unknown context, it is not almost clear how to interpret a search query. Identifying ambiguities and adequately handling them is a difficult thing to do. **FR** proposed to include synonyms in the search, so, for example, when searching for *engine*, the search would also consider the term *motor*. Such a synonym search *SHOULD* be regarded as an optional search option.
- R32** The search syntax *SHALL* be similar to the Lucene syntax. GitHub and Google use a Lucene-derived syntax, which many users are already familiar with. These users should have no problems adjusting to using the service search syntax.
- R33** To avoid irritating the user, syntax errors *SHALL* be handled gracefully. When such an error occurs, the search query *SHALL* be parsed as best as possible, and appropriate results *SHALL* be displayed. This way, the search outcome might not reflect what the user desired, but it won't discourage the user as much.
- R34** Because Search Operators can offer a profound insight into a dataset, they can be "abused" to extract hidden sensitive information stored within the dataset. This technique of hacking is called *dorking*. Dorking makes extensive use of operators to search for particular combination terms. As a result, one may find documents that are not meant for public consumption or valuable information about target systems that can be used to uncover security flaws and attack vectors. With that in mind, some efforts *SHOULD* be made to remove sensitive information before they are stored in the database.
- R35** The number of operators and filters is potentially large. Displaying all available options by default would result in a cluttered user interface and discourage new users. Therefore, the number of displayed elements in the web interface *SHALL* be limited by default. The user *SHALL* still be able to access all available operators and filters if desired.

R35-1 The basic search interface *SHALL* consist only of the search query input. It *SHALL* support the full search syntax and functionality, but *SHALL* hide the complexity, which would overwhelm novice users.

R35-2 The advanced interface *SHALL* enable the user to construct a search query without the need to consult the documentation first. A possible solution could be the approach that GitHub employs on its advanced search page. In addition to the search input field, an input field for every important operator is offered. Each time the user inputs a value into one of these fields, the associated operator and inputted value are added as a term to the search query.

Search Order

Search Order defines the display order of the search results. Depending on the information needs, the user can specify to sort the results by one or more properties in ascending or descending order.

R36 The interviewees unanimously wished for an option to control the display order of the results. Therefore the service *SHALL* offer such a feature. The following product properties should be considered for ordering:

- Name
- Language
- State (Activeness)
- Popularity/Star Count (**OS**)
- Fork Count
- Version
- Created At
- Last Updated At
- License Name/ID
- License Type
- Licensor Name
- Host
- Relevance (Best Match)

R37 Search results *SHOULD* be sortable by relevance so that the "best" matching results will be prioritized and ordered appropriately. This requires the service to rate and prioritize results.

Pagination

Pagination is the process of splitting content into discrete pages. It limits the content on one page and prevents the server and client from overloading. For example, the results for a search request might be delivered with ten results per page instead of returning a potentially massive amount of data.

R38 Instead of cycling through pages, it is more convenient to display as many results on one page as **JP** stated. A single page that continues to show more results as one scrolls down is favorable. If it proves to be infeasible, the user *SHALL* at least be able to set the number of search results per page. Per default, ten results *SHALL* be shown per page, but the user *SHALL* be able to increase the number to 20, 50, or 100 results per page. This functionality is also requested by **NW**.

Presentation of Results

There is no single way of presenting the product information. Different display styles serve different purposes. For example, while data tables are well suited to display heavy data with many properties in a compact form, a card-style presentation with large preview images might be more visually pleasing and better suited for showcase purposes.

R39 As mentioned before, there are several ways to present product information. The interviewees were presented with different options and asked for their preferences in that regard. **JP** prefers a card-style overview to showcase the products. **MH** also suggests using a list/card style as a form of a gallery as default but also have the option to display the results in table form. A table view would make it easier to compare and sort results as desired. **TW** praised the platform *Knowable* for its differentiation between workspace and showcase. The workspace presents the maker with all information while the showcase offers an overview for general users. **TW** suggested adapting the concept and using cards for showcasing products, while other views might provide deeper insights. Hence, the service *SHALL* offer different presentation styles from which the user can choose. Supported *SHALL* be a table-style view and a card-style view. The latter one *SHALL* be used as default.

R40 Today, many users use their smartphones to browse the web. Even though the service is focused more on professional use where the primary input device is most likely a PC, there *SHALL* also optimized mobile view for smartphones and similar handheld devices (**TW**).

R41 **FR** and **JP** criticized the display of the product versions in the demonstrator. It doesn't add meaningful value to the search results and leads to confusion. In general, each product has a large set of information properties. Not all of these properties carry the same information value. They are also not inherently comparable between different products. Displaying all product information simultaneously would clutter the user interface without adding additional value to the user. Therefore, it is vital to provide a meaningful selection of properties to be displayed. Per default, only properties *SHALL* be selected that are most relevant and/or are comparable between different products. The following properties are considered for inclusion:

- Product Name (**OS**)
- Product Image (**JP**, **FR**, **OS**)
- Description (**OS**)

-
- Documentation Language
 - Author
 - State/Activeness (**OS**)
 - Number of Forks (**OS**)
 - Number of Stars (**OS**)
 - Version in Combination with Creation Date (**TW**)
 - Category (**TW**)

R42 Depending on the use case **FR** is interested in a specific subset of properties to be displayed. He desires an option to customize the default selection of properties on a by-search basis. Therefore the user *SHOULD* be able to make their selection of properties to be displayed in the search results overview.

R43 **JP**, **FR**, **OS** and **PJ** mentioned the importance of preview images. Almost all platforms showcase preview images of the products so that the user gets a general idea by just glancing over them. Because of the value that preview images add, the service *SHALL* also present preview images. The original product images must be therefore indexed alongside the general product information. **PJ** also suggested generating preview images from 3D volume models if the creator provided no avatar.

R44 **FR** proposed to include a catalog of synonyms. Product categories, names, and other properties could then be extended with one or more synonyms. The user would then be able to select synonyms to discover similar but differently named products quickly. The service *SHOULD* integrate a set of synonyms for different languages and give the user a choice to search by synonym to discover more products. There are some synonym datasets freely available. For example WordNet and OpenThesaurus offer lexical databases for download.

R45 **FR** and **JP** mentioned the use of tooltips. Even though the interface is supposed to be simple and intuitive, each input element *SHOULD* feature a helpful tooltip describing its use and optionally additional information.

R46 The value of the dataset not only stems from the number of products, components, and other elements in that dataset but also the relationships between these elements. In the opinion of **FR** and **PJ**, those relationships *SHOULD* be made visible and explorable through visualization such as interconnected bubbles, a visualization style most prominently used by the Neo4j database. An online example of that feature can be seen at connectedpapers.com.

R47 Products, where 3D models are available, can be visualized with a web-based 3D editor. **FR** proposed to add highlighting to proprietary components, so it is immediately evident which parts are open-source and which are not.

R48 An interesting opportunity was described by **FR** to add a rough location map of where OSH products are created. This can potentially help identify manufacturers and accelerate the product creation process. In that regard, the service *SHOULD* collect rough location data where possible and offer a simple map for the user to explore.

-
- R49** Displaying the search results in the overview can only include a small selection of collected information for each product. A dedicated details page *SHALL* be offered so **MH** and **OS**. This details page would contain all information about one product and give the user an in-depth view. The page *SHALL* list all indexed product properties, display all containing components and *SHALL* also link to the related semantic page. The page *SHALL* also feature a version history, from which the user can select a specific product version to display. As an inspirational example, **OS** suggested the work of OHO and their overview pages. **PJ**
- R50** In the context of version history, **JP** mentioned a diff mechanism, where two versions of a product could be compared side by side. This feature would allow the user to see what changed between versions and may disclose introduced incompatibilities between two versions of a product. Such a feature does have potential, and *SHOULD* therefore be implemented.
- R51** The properties mentioned in the user search query *SHOULD* be included in the results overview because they can be considered information of interest to the user.

Export

The search results will be displayed in a human-readable form via the service's web interface. The user would need to use web scrapping to get the results in a structured, machine-readable format. This method is unreliable due to future changes to the web interface. Furthermore, the amount of information one would receive for each result this way would be pretty limited. This issue can be resolved by providing an export functionality. An export function would allow the product search results to be downloaded in a machine-readable form. The server creates a document containing each result's information and presents it to the client to download.

- R52** For research purposes **MH** sees great importance in the ability to export search results. He states that comparing and processing results locally is often quite helpful. **FR** also sees some benefit in evaluation and automation but generally considers the export less important. Nonetheless, the user *SHALL* be able to export the results of the current page in CSV format.
- R53** The number of results is potentially quite large, and exporting all of them will be relatively computationally expensive. To protect the service against overloading, the number of exportable results *SHALL* be limited to 300. This limit should be more than enough for the needs of most users.

4.4.3. RDF Resource

To create a semantic web service, the service *SHALL* provide a resource lookup mechanism. Each product, part, and other resource will be given a unique URI as an identifier. When a client requests a resource, the URI gets dereferenced, and the user will receive a human-readable HTML document, while a machine will receive a RDF representation.

- R54** All resources *SHALL* be referenceable by an unambiguous URI. The URI *SHOULD* be concise and preferably easily human-readable (e.g. no long chain of parameters).

-
- R55** A web client *SHALL* be able to dereference a resource URI and negotiate the desired document type to receive. The web client sets the HTTP headers Accept and/or Accept-Language to define the desired format. The server then answers with a 303 redirect and a Location header where to find the document. The web client must proceed by creating a new request to the document location. The server then again answers with the document and the appropriate HTTP headers Content-Type, Content-Language and Content-Location.
- R56** For each resource, there *SHALL* be a human-readable representation in the form of a web page at `domain.tld/page/<resource>`. To get the location for this representation a web client must set `Accept: text/html`. This will be the default when requesting the resource via a web browser.
- R56-1** The web page *SHALL* visually reveal the locations for corresponding RDF documents. This way, a user can easily download an RDF representation for that resource.
- R56-2** The web page *SHALL* also contain in its header `<link>` elements that point to the corresponding RDF documents, e.g.: `<link rel="alternate" type="text/turtle" href="xxx" title="Structured Descriptor Document (Turtle format)"/>`
- R57** For each resource there *SHALL* be a Turtle RDF representation at `domain.tld/data/ttl/<resource>`. To get the representation location, a web client must set `Accept: text/turtle` when requesting the resource by URI.
- R58** For each resource there *SHALL* be a N-Triples representation at `domain.tld/data/ntriples/<resource>`. To get the representation location, a web client must set `Accept: application/n-triples` when requesting the resource by URI.

4.4.4. API

An *Application Programming Interface (API)* is a technique for two or more systems to communicate with each other via defined protocols. A API endpoint allows programmers to connect to the service and use its functionality in their applications.

- R59** The service *SHOULD* offer a GraphQL endpoint at `domain.tld/graphql`. Users could then use the endpoint to query resources easily and receive results in structured, machine-readable *JSON*.
- R59-1** The GraphQL endpoint *SHOULD* provide access to the service search functionality.
- R59-2** The GraphQL *SHOULD* give "direct" access to the underlying dataset.
- R60** The service *SHOULD* offer a *SPARQL* endpoint to provide an RDF native method to perform complex queries. The endpoint should be available at `domain.tld/sparql`
- R61** Long-running, resource-intensive queries can harm the service's operation and availability. API request *SHALL* therefore be rate limited.

5. Design and Implementation

This section contains high-level details about some of the design and implementation choices and challenges. In addition, the underlying problems are described, and the reasoning behind the design and implementation is explained.

5.1. System Architecture

The entire application is written in the Go¹ programming language. To mention some of the benefits of Go: It is a type-safe language and has built-in memory management while still being more performant and resource efficient than Java, it has a robust standard library and rich ecosystem of community packages, it compiles blazing fast and has many more aspects that make it attractive for modern days applications.

The application in this work consists of three components, two of which are developed within this work and one external software module. A network diagram of the components is depicted in Figure 5.1.

- **Crawler:** The crawler is a bot program that automatically collects metadata and technical documentation from selected platforms. It is responsible for finding product data, parsing it, extracting relevant information, checking conformance with the OKH-LOSH specification, and adding eligible products to the search index. More details about the crawler and its implementation can be found in Section 5.3.
- **Web Interface:** The web interface allows users to interact with the service. It offers a convenient way to search and filter products, parts, and other entities based on their properties and relationships. Furthermore, it provides a semantic structure to access information based on URIs using well-established SW technologies. In Section 5.4 we offer some insights into the functionality behind the product search and SW data access.
- **Database:** The database is used to store and manage product information. This includes metadata, technical information, and references to where to find the product's design source. In addition, the database plays a crucial role in the product search, since most of the search logic is directly executed there. In this work we decided to use Dgraph, a graph database instead of a more conventional relational database such as MariaDB² or PostgreSQL³. In Section 5.2 we go into details about why and how we used the database and what data schema we came up with.

¹ <https://go.dev>

² <https://mariadb.com>

³ <https://www.postgresql.org>

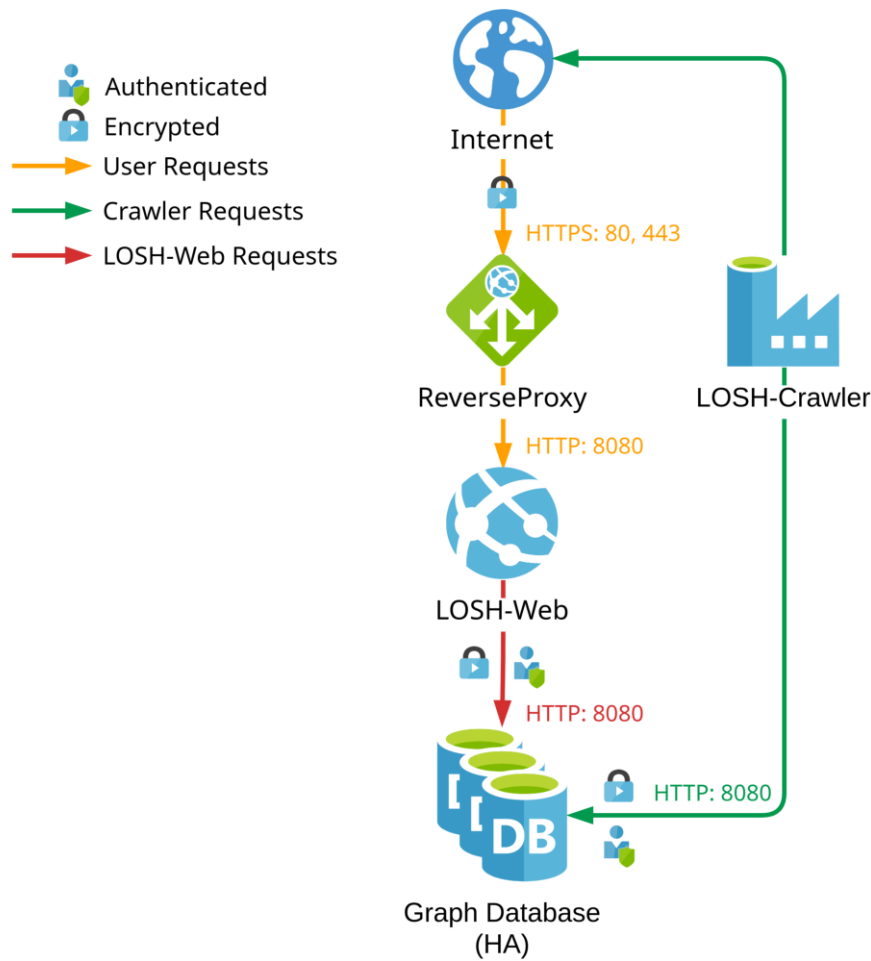


Figure 5.1.: Network Diagram

The following sections cover some interesting aspects, choices, and challenges we encountered while implementing the service.

5.1.1. Reasoning for a New Implementation

A demonstrator has been developed as part of the OPENNEXT LOSH project. The technological choices and utilized components proved inadequate for the given use case. And because the used components are an integral part of the whole system, they cannot easily be removed and replaced. Therefore, we decided to create a new implementation for the whole service instead of reusing and improving the existing service.

We compiled a list of reasons that speak against the demonstrator as a basis for future developments:

- Wikibase is designed to be used within the MediaWiki wiki and offers only a bare minimum external API for developers. The full extent of features can thus only be used from within MediaWiki. The demonstrator is designed as a dedicated service that uses Wikibase for storage but does not build on MediaWiki. In order to deal with the API limitations, a plugin called

*WikibaseReconcileEdit*⁴ has been developed. It offers additional endpoints for Wikibase data access and modifications but is still very limited in functionality. Missing API features lead to the exploitation of the other databases in the architecture to provide at least a minimum set of service functionality. For example, Wikibase does not offer full-text search by itself but relies on the full-text search capabilities of ElasticSearch. The Wikibase API doesn't expose this functionality; thus, the LOSH Backend application has to query the ElasticSearch database directly.

- The demonstrator relies on three database systems to work. All product data must be duplicated and synced between these three datasets, thereby burning computing and memory resources. The reasoning for using these three technologies is simple. Wikibase doesn't offer full-text search or a SPARQL endpoint and must rely on additional services to provide these features.
- The entire architecture is too complex to set up, and there was insufficient documentation on that matter. As a result, the developers of the crawler and LOSH Frontend applications connected directly to the production environment to develop and debug the software. Ideally, the developers can easily set up a local or at least dedicated development environment for development purposes without the need to ask for access to the production environment.
- A range of different programming languages, frameworks, and environments are used in the project. Node.js, Python, PHP, and Java, just to name a few. This makes it really hard for any single developer to comprehend and contribute to the project.

Building a new service from the ground up allowed us to tailor a system specific to the use case of the thesis. In general, it offered us the following advantages:

- Using a programming language of our choice
- Design a simple and efficient system architecture
- Reuse of functionality and libraries across components for an internally consistent system
- Allow us to focus more on the aspect of usability and user experience

5.1.2. Notable Components, Libraries and Frameworks

Dgraph

*Dgraph*⁵ is a distributed graph database and is developed by Dgraph Labs. It offers a wide set of features, notably [23]:

- Native GraphQL: Dgraph offers native support for GraphQL. It is not restricted to data access. Dgraph also allows to define the database data schema directly as a GraphQL schema and automatically generates a CRUD (Create, Receive, Update, Delete) API for accessing and managing the data.

⁴ <https://github.com/wmde/WikibaseReconcileEdit>

⁵ <https://dgraph.io>

-
- **Powerful Query Language:** Dgraphs supports GraphQL, but its functionality is somewhat limited. Thus, Dgraph offers a custom query language called DQL, which allows harnessing the complete feature set of Dgraph. The query language is based on GraphQL and extends its features to meet the requirements of the database.
 - **Full-Text Search:** Full-Text search with stemming and more are built into the database and do not require an additional third-party service. Full-text search is available for all text-based data. All that is needed is a full-text index on the desired data fields.
 - **Custom Logic with Lambdas:** Custom logic allows to extend the database with data fields, resolvers, mutations, and webhooks. The custom logic can be written in JavaScript or Go and deployed on a serverless platform.
 - **Horizontal Scalable:** Dgraph employs sharding in order to shard data across multiple servers. It automatically synchronizes changes with all database instances within the cluster.
 - **Transactional Queries:** Dgraph offers transactional queries with ACID (Atomicity, Consistency, Isolation, Durability) properties. It guarantees high levels of data integrity and availability.

Dgraph is offered as open-source under a community license. Furthermore, a paid enterprise version is available that offers additional features not included in the open-source version, such as Full and Incremental Backups, Data At Rest Encryption, and more.

The software is entirely written in Go, which fits nicely into the service's overall Go architecture and is likely easy to adapt when the need occurs.

Fiber

*Fiber*⁶ is a Go web framework, that is inspired by the popular *Express.js* for Node.js. It is built on top of *Fasthttp*, one of the fastest⁷ HTTP engines for the Go programming language. As such, Fiber combines the ease of use of the *Express.js* framework with the performance of Go for building high-performance web applications and web APIs.

Fiber provides robust routing, templating, error-handling and more. In addition, the request-response cycle can be extended through middleware with additional functionality such as request logging or compression.

Fiber is developed as a community effort and released as open-source under the MIT license.

Liquid

*Liquid*⁸ is a templating engine created by Shopify. It is intended as a customer-facing template language to allow easy and flexible theming for web applications. It is used by various services, including Shopify itself, Salesforce, Zendesk, and many more. The language has a similar grammar as *Django*⁹ or *Jinja*¹⁰, and is thus easy for developers with a Python background to learn.

⁶ <https://gofiber.io>

⁷ <https://github.com/savsgio/atreugo#benchmark>

⁸ <https://shopify.github.io/liquid/>

⁹ <https://www.djangoproject.com>

¹⁰ <https://jinja.palletsprojects.com/en/latest/>

The template engine is written in Ruby and used in the popular *Jekyll*¹¹ static site generator. Jekyll combines Liquid templating with Markdown and HTML/CSS to generate easily deployable static websites hosted with a plain web server without needing additional software and dependencies.

As a personal effort of a developer to reimplement Jekyll in Go, the Liquid templating engine was ported to Go¹². The port of Liquid and some plugins of the Go Jekyll implementation are used to implement the service that is part of this thesis.

Both the original Liquid template engine and the Go implementation are licensed under the terms of the MIT license.

Tabler

*Tabler*¹³ is a UI web kit. It is build on *Bootstrap*¹⁴, a free and open-source HTML/CSS framework for building responsive, mobile-ready, web front-ends. Tabler offers a coherent, modern style and a wide variety of ready-to-use UI components and templates. It is suitable for modern websites, especially interactive dashboards with various visual elements for presentation and interaction. To name a few components and features that Tabler offers:

- Input Forms (buttons, text input boxes, etc.)
- Modals
- Tables
- Drop-Down Menus
- Galleries
- Icons

The templates and demo page contents are provided as a Jekyll environment. The templates are written in the Liquid language and use Jekyll and some custom filters and tags. To reuse the Tabler-provided templates and avoid larger modifications, the Go Liquid implementation and some Go Jekyll components are used in the service implementation.

Tabler is licensed and distributed under the MIT license.

Participle

*Participle*¹⁵ is a Go library for generating parsers.

A parser is a syntactic analyzer. It is used to analyze a string of symbols in the form of a computer or natural language and turn it into an Abstract Syntax Tree (AST). The input string must conform to a formal grammar to be parsable and return meaningful results. This grammar can be expressed in different forms, e.g., the Extended Backus–Naur Form (EBNF). Now, a parser generator takes a

¹¹ <https://jekyllrb.com>

¹² <https://github.com/osteele/gojekyll>

¹³ <https://tabler.io>

¹⁴ <https://getbootstrap.com>

¹⁵ <https://github.com/alecthomas/participle>

grammar expression and creates a parser in the target programming language that can decipher an input language based on that grammar. It is also possible to encode the logic to parse a language manually. These types of parsers are typically referred to as *handwritten parsers*. Handwritten parsers usually perform better and are more flexible when handling invalid inputs, but they are more labor intensive and harder to maintain.

Participle belongs to the category of parser generators. Rather than taking the grammar as a separate file, the Participle approach combines the grammar with the data model of the AST. This is accomplished by the Go native *struct tags* annotations, which are commonly used for data encoding and other forms of data processing. Mapping an input to a field works by annotating it with the fitting grammar. The grammar itself is defined in a Participle specific syntax.

Participle is open-source and released under the terms of the MIT license.

5.2. Database and Data Model

The database is used for the storage of product information. All the product metadata and technical documentation that is collected by the crawler, along with service-related data, is stored in a structured format in the database. Besides data storage and simple data access, the database also plays a crucial role in the product search. A user-submitted search query gets translated to a database query and is executed by the database. More details on the search functionality can be found in Section 5.4.1.

Dgraph was chosen as a database for this work. Dgraph is a graph database, meaning the data is stored as a knowledge graph using the notion of *Nodes* and *Edges*. Data is attached to Nodes, which are interconnected via edges to express relations between the data nodes. Relational databases such as MariaDB or PostgreSQL rely on data tables and a fixed schema to store the data. Graph databases do not require a predefined schema and are thus very flexible and can be changed more easily throughout the application's lifetime. We prefer to choose a graph database over a relational database because of the following reasons:

- A Graph database expresses relations between data entities by connecting nodes via edges. This allows the database to resolve and access related data by simply following edges in the graph. Graph databases are a good fit for datasets whose value lies primarily in relations between the data. Relational databases rely on *Joins* to express relations. These are quite costly, which is why relational databases, despite their name, are less optimal for workloads with many relationships.
- The product information that we collect from various platforms varies quite a lot in quality and quantity. Many products are provided with only a minimal set of metadata. Thus, our dataset contains a lot of sparse data. Graph databases perform better than relational databases when it comes to sparse data such as ours.
- Graph databases do not require a schema to store data and are more flexible regarding data model changes. Since our data model changed multiple times and is still evolving, graph databases proved to be a good fit.
- Most of the relational databases only provide fundamental text search functionality. For a full-text search that is comparable to Google's text search, an additional service, like another document-based database, must be used. Dgraph natively supports full-text search, making it preferable in our use case.

-
- RDF as a SW technology is based on a graph structure. It can be implemented on top of relational databases, but directly using a graph database is advantageous.

Even though no data schema is required to get started with Dgraph, going without one is not recommended. For our work, we created a data schema that is derived from the LOSH specification. The specification is intended for documentation purposes and thus cannot be used without modifications. Therefore, we designed a data model suitable for this work's use case that shares similarities with the LOSH specification. As Dgraph allows us to define the schema as a regular GraphQL schema, we created and documented the data model using GraphQL. From there, we implemented the model as a recursive, graph-like data structure that is used by the crawler and web interface components.

A short excerpt of the data model is presented in Listing 1.

```
1 interface CrawlerMeta {
2   discoveredAt: DateTime! @search
3   lastIndexedAt: DateTime! @search
4   dataSource: Repository!
5 }
6 type Product implements Node & CrawlerMeta {
7   name: String! @search(by: [hash, fulltext, regexp])
8   releases: [Component!]! @hasInverse(field: product)
9   # ...
10 }
11 type Component implements Node & CrawlerMeta {
12   name: String! @search(by: [hash, fulltext, regexp])
13   description: String! @search(by: [fulltext, regexp])
14   version: String! @search
15   repository: Repository!
16   license: License
17   licensor: UserOrGroup!
18   # ...
19 }
```

Listing 1: Data Model Excerpt

5.3. Crawler

A *Web Crawler*, also known as Spider or Bot, is a program that automatically scours the internet to find and extract information of interest. It does so by following links, downloading documents, extracting relevant information, and organizing discovered information in a structured manner for later retrieval and usage. Crawlers are mostly used by search engines like Google (called Googlebot). Search engine crawlers crawl large portions of the public internet, create an index that functions as a catalog of information for efficient lookup, and make it possible for users to search and find things on the web they desire. There are more use cases. To name a few: Aggregators for news or business intelligence

sites or harvesters of supposedly private and confidential data for spamming/phishing purposes [24], [25].

For the purpose of this thesis, the crawler is used to automatically collect metadata and technical documentation from selected platforms. It is responsible for parsing the data, extracting relevant information about hardware products, checking conformance with the LOSH specification, and storing it in the database. Unlike regular web crawlers, the crawler for this project only has to navigate and index selected platforms and not the entire web. Furthermore, instead of crawling HTML documents and extracting information via text mining, platform-provided APIs, if available, are used to get product metadata in a structured format.

The crawler in this work primarily performs two operations: discovering products and updating them. The discovering operation refers to a process of finding relevant products that provide at least a minimum set of mandatory information, and adding the product to the index. The update operation checks already known and indexed products for changes and updates the indexed information accordingly. As security and availability of the whole service are taken seriously, any user-generated collected data is sanitized by removing HTML and other elements before it is stored in the index. The high-level process steps of the discovery and update operations are depicted in Figure 5.2 and Figure 5.3. Figure 5.4, 5.5, 5.6 and 5.7 are concerned with lower-level details that shall not be discussed here.

A few challenges need to be overcome when designing and implementing a crawler. Some of these challenges are related to the unstructured nature of most web documents and the need for text mining. Other challenges regard the selection and update process of information. However, there are four challenges that essentially every kind of crawler needs to overcome in order to guarantee an up-to-date data index and stable operation. Those are the challenges that are addressed in this work. The following sections cover the implemented solutions.

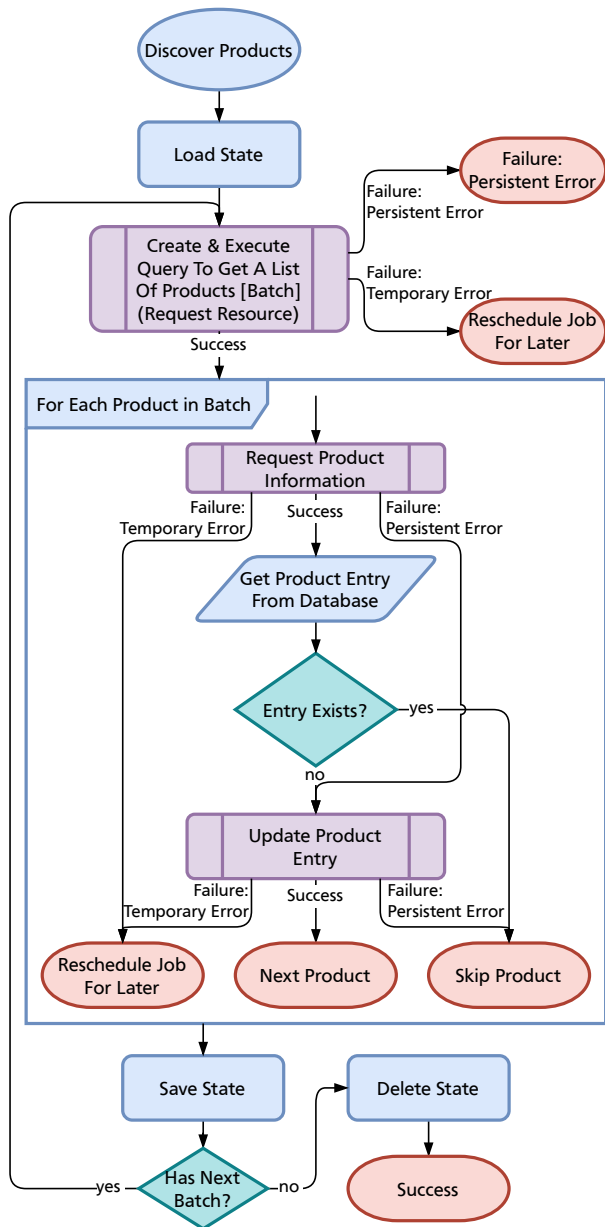


Figure 5.2.: Discover Products Flowchart

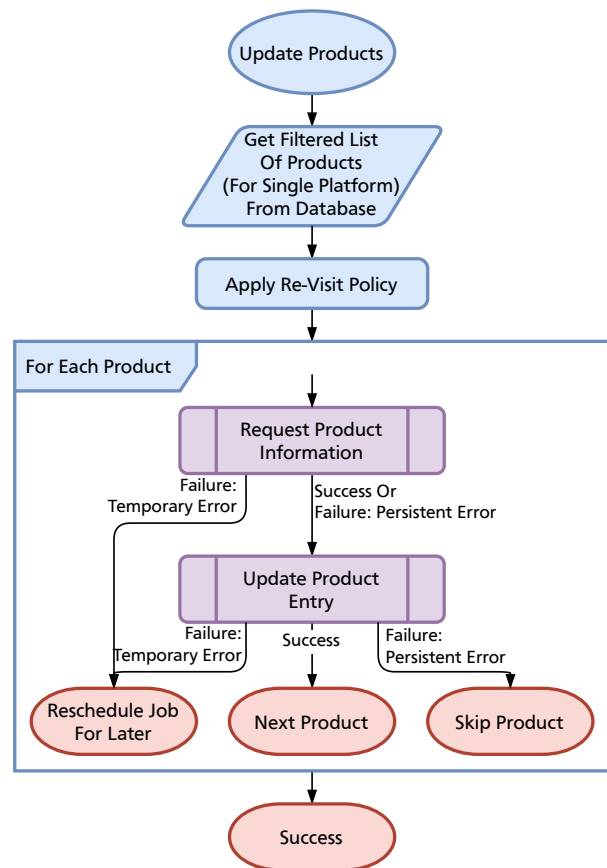


Figure 5.3.: Update Products Flowchart

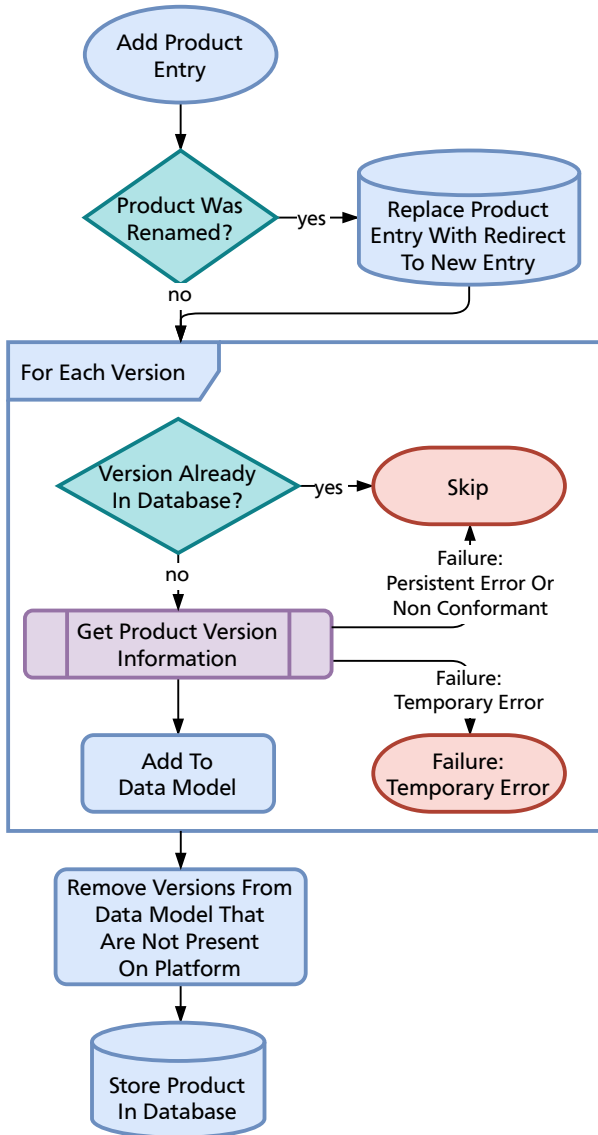


Figure 5.4.: Add Product Entry Flowchart

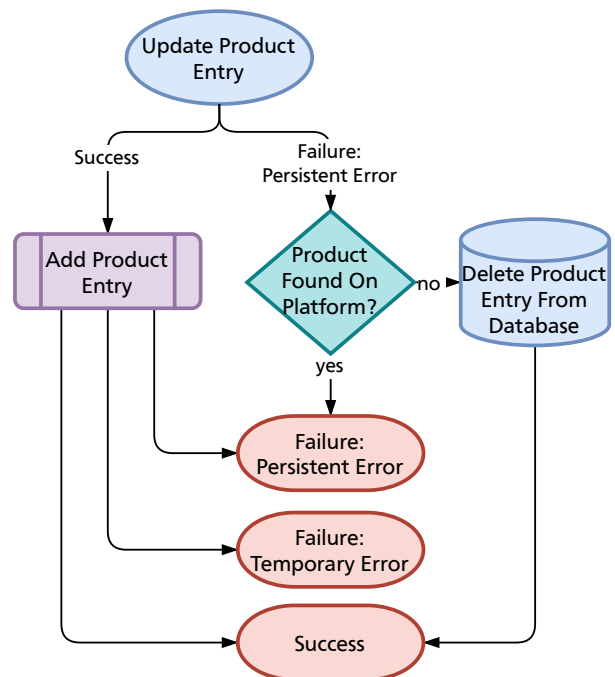


Figure 5.5.: Update Product Entry Flowchart

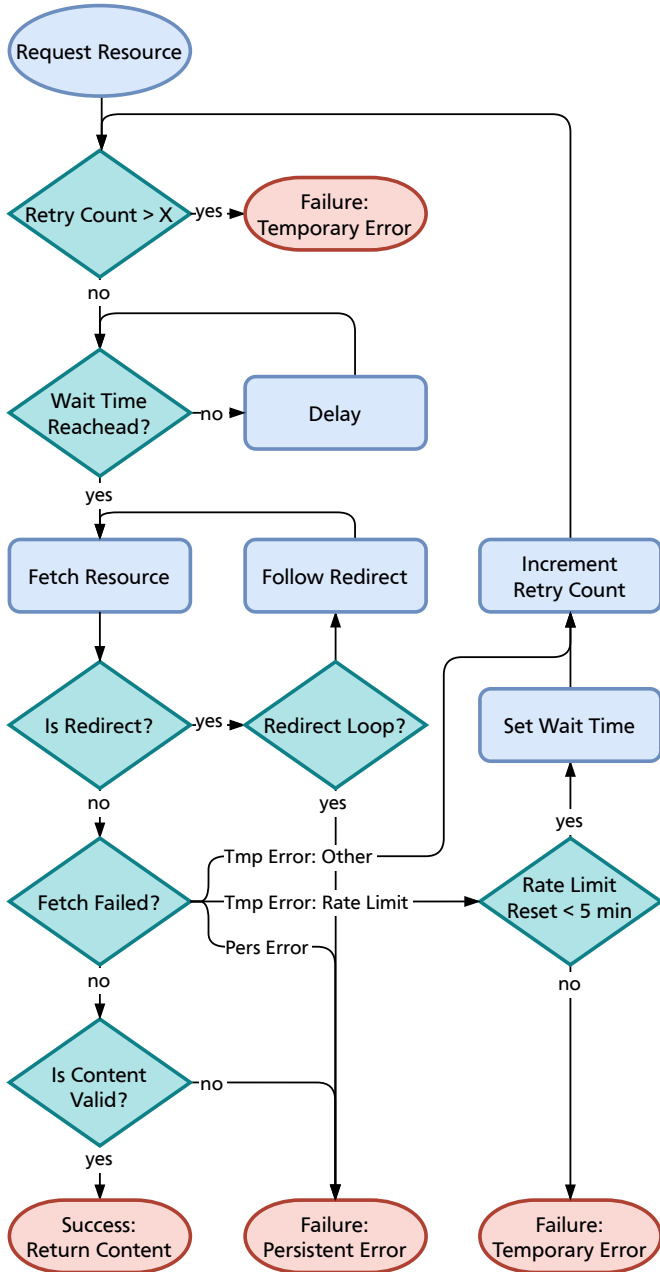


Figure 5.6.: Request Resource Flowchart

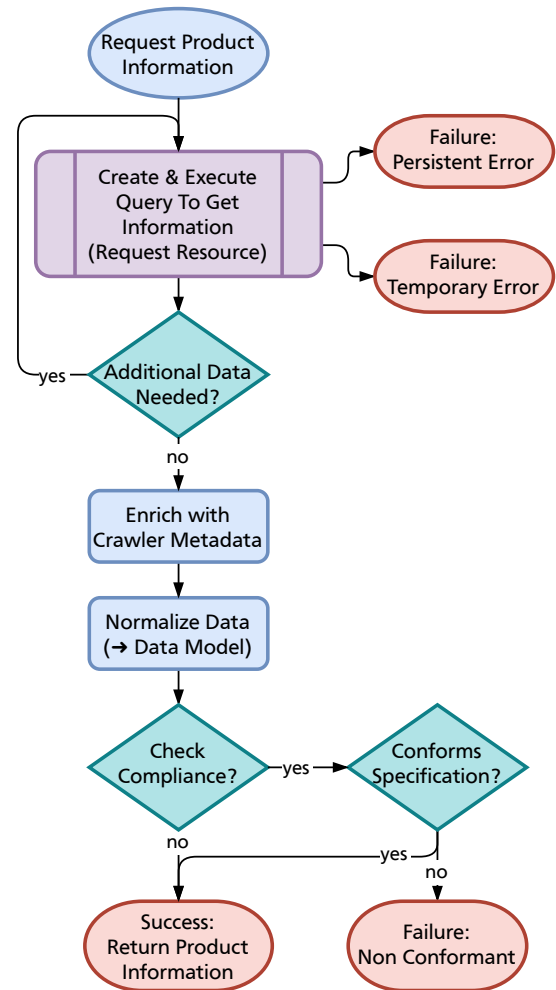


Figure 5.7.: Request Product Information Flowchart

Data Selection Policy

Data selection refers to a scheme of selecting potentially interesting content from a large pool of data. In the context of this thesis, we are only interested in hardware-related information that we find on selected platforms such as Wikifactory, Thingiverse, or GitHub. Thus, the developed crawler is explicitly tailored to this use case. Even though the crawler is limited to a few selected platforms, the search space is rather large and is subject to growth in the future. Despite the amount, newly created products need to be found in a reasonable timeframe, and already indexed products need to be kept up-to-date. Therefore a selection scheme has been implemented to guarantee efficient operation.

Some of the platforms offer access to their data via an API. Considering there is no single standard for data access on these platforms, each platform needs to be treated individually, and thus no single selection scheme can be deployed. To offer a sense of what the data selection entails, we offer two examples with two different hosting platforms:

- **Wikifactory:** Wikifactory is a product development workspace and hardware hosting service. It offers a GraphQL API that allows iterating over all projects on the platform. Furthermore, it lets us query a large set of information about the projects and their creators, making it easy to index the products on the platform. Getting all information of a single project involves most likely multiple database calls and is a rather slow process. Considering that a large number of projects do not include the mandatory information needed to be included by our service, it would be a waste of time and resources to simply request all projects along with the full set of data from the platform. A selection has to be made. For discovering products, a query is issued that requests only the mandatory information to check the conformance conditions of each product. When a product is deemed conformant, another query is issued to receive the entirety of the product's information.
- **GitHub:** GitHub is a hosting service mostly used for software development. It uses Git for version control which can be used for more than software development as some makers already manage and upload their hardware creations using Git. If makers include a OKH (see Section 3.3) or LOSH (see Section 3.4) spec file and name it with a prefix of okh, then it might be eligible for inclusion in the search index. GitHub contains more than 200 million repositories [26], of which only a tiny fraction are hardware-related repositories.

Files can be accessed and downloaded from GitHub via Git. Additionally, GitHub offers a GraphQL API and REST API for accessing repository metadata. Going through all repositories and millions of files is unfeasible. Using Git, we could download each repository's contents and check for files with a name prefix of okh but that would require us to download terabytes of data. Also, iterating through all repositories metadata using the provided APIs is also not doable because those are tightly restricted and limited. Therefore, we rely on GitHub's code search, which allows searching for specific lines of codes or specific files. The crawler uses the code search to find eligible repositories, downloads the spec file, checks the conformance, and indexes the product. The code search has a series of limitations, which we won't discuss, that may result in an incomplete dataset with some of the products on GitHub being missing in our search service.

Re-Visit Policy

Already indexed products need to be revisited to check if any updates are available. The current implementation is kept simple and needs some optimization in the future. The general re-visit and update process is depicted in Figure 5.5.

When a product is discovered and indexed, it is enriched metadata from the crawler to remember the product's source, when it was discovered, and when it was last indexed. This information is used by the crawler in the update process. The update process is periodically triggered by a timer. Once it is triggered, a database query is performed that returns all products from a specific platform that have a *last updated* time greater than one week. For each of these products, the crawler gathers the information from the source platform and compares it to the already indexed data. Any changes will be saved in the index alongside an updated *last updated* timestamp.

To reduce the frequency and amount of products to update each cycle, a future implementation should consider a more refined re-visit policy that takes the update frequency and popularity of products into account. More active products could be updated more often, while less active products could be held off for a longer time.

Politeness Policy

The crawler generates potentially thousands of data-intensive queries in a short time. These requests can strain the target platform and degrade its performance. Best practice dictates that the speed and number of requests per time need to be limited. Otherwise, if the platform providers feel pressured, they may take action and block the crawler entirely. Therefore, a politeness policy was implemented that limits the number of requests and spreads them over a longer period of time.

This mechanism is implemented through a *Requester* component that is used for any kind of outbound requests such as GraphQL or REST queries and file downloads. The Requester can be configured with a limit of total requests or a limit of requests within a specific time frame. It keeps track of the performed requests and delays or stops consecutive requests when the configured threshold limit is reached.

Parallelization Policy

The service is not intended to be the next big player on the market and compete with established search engines such as Google. Still, the underlying architecture shall be efficient and reasonably scalable. At the start, the service implementation only includes support for one or two platforms, but the portfolio is bound to grow in the future. As such, multiple crawler instances need to be executed in parallel to provide fast updates to the index for various source platforms.

Parallelization for the crawler in this work is rather straightforward. Multiple platforms are crawled in parallel, but each platform is only be accessed by one crawler instance at a time. The different crawler instances are configured only to crawl specific platforms without overlap between them. Therefore, there is no need for a synchronization mechanism at this point.

5.4. Web Interface

The web interface is the primary way to interact with the service. It offers the service's full capabilities, including the product search and SW data access. The interface consists of multiple pages and views:

- **Homepage:** The *Homepage*, depicted in Figure 5.8, is the first page the users see when visiting the service. Currently, it only offers a simple *hero* section with a very short introduction and a search bar to get started with the product page. When a user enters a search query, they get automatically forwarded to the search page. In the future, more sections might be added for a more detailed explanation of the service and what platforms are supported for searching.
- **Search Page:** The *Search Page* embodies the main functionality of the service. It allows users to search for OSH products of their interest and displays the results in either a card-style showcase format or in a table/list-style format. The implementation details for the product search are presented in Section 5.4.1. The search page consists of a multitude of input and control elements to perform the search and govern the results' filtering, ordering, and display. Figure 5.9 depicts the page and its elements. The elements are as follows:
 - Search Query Input: ❶ The *Search Query Input* is the primary text input for formulating a search query. The users can start by typing keywords that describe the thing their searching for and later on narrow the results down by adding search operators.
 - Query Cheat Sheet Toggle: ❷ The *Query Cheat Sheet*, depicted in Figure 5.10, offers an overview of the search query syntax and the available search operators. The cheat sheet is quite extensive and might be a little intimidating at first; therefore, users can toggle the cheat sheet on and off, so it won't get in their way. For convenience, the users can click on the options listed in the sheet and it will automatically be added to the search query.
 - Selection of Order: ❸ Ordering of the results is one of the primary features that most data query applications offer. The service lets the user choose the property and direction

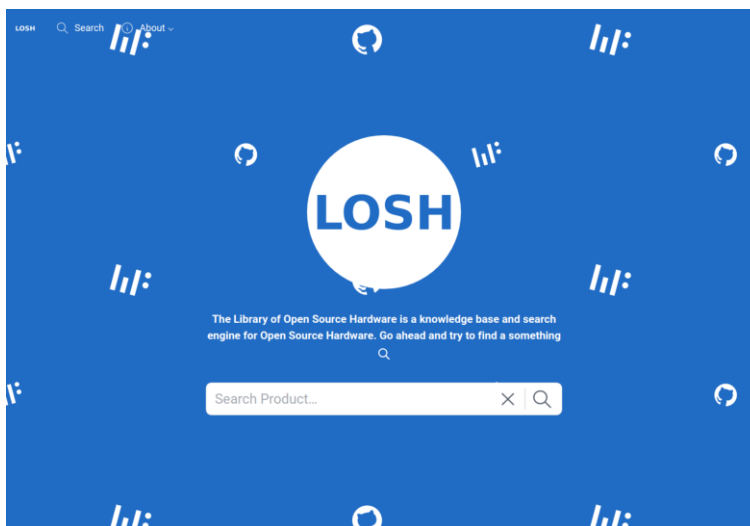



Figure 5.8.: Homepage

-
- (ascending/descending) to order via a simple select box. The table view also allows the users to sort by pressing the table headers of properties.
- **Selection of Presentation Style:** ④ The results can be displayed in one of two forms, either in a card-style showcase or a table-style overview. The users can choose the view by clicking on the button located on the right on top of the results view.
 - **Results in Card Form:** ⑤ The card-style view offers a showcase of the results with a large, prominent image of the product, the product name, description, along with other basic product properties. The view is intended for regular users that want to discover products by appeal.
 - **Results in Table Form:** ⑥ The table-style view offers an overview of the results and allows users to quickly compare properties of different products.
 - **Results Export:** ⑦ The results can be exported in currently two formats, comma-separated values (CSV) and tab-separated values (TSV). All a user has to do for the export is to perform a search and select the desired format from the export drop-down button. The export then exports up to 300 results in the selected format.
 - **Pagination Controls:** ⑧ The pagination controls allow the users to cycle through the result pages and control the number of results to display on each page.
- **Details Pages:** The *Details Pages* display all the information that has been collected by the service for entities such as Products, Licensors and Licenses. The pages are intended to offer coherent, in-depth insights into the different entities without seeking out the source platforms.
 - **Product Details Page:** The product details page (Figure 5.11) presents the product metadata ① in a simple, visually appealing form. In addition, it features a gallery of images ② that are associated with the product, like images from used components, etc. Once the users are satisfied with a product, they can go to the source repository via the dominant button ③ at the bottom of the details page.
 - **Licensor Details Page:** The licensor details page (Figure 5.12) displays the creator's information: name, description, avatar image ①, and a list of their products ②.
 - **License Details Page:** The license details page (Figure 5.13) features an overview of the license properties ① and the actual content of that license ②.
 - **RDF Resource Page:** The *RDF Resource Page* offers an overview of all properties of data entities. It is part of the SW implementation of the service as described in Section 5.4.2. As depicted in Figure 5.14, the page displays the information in list form ① and offers a download of the resource in a specific format ②.


[Search](#)
[About](#)


Product Search

× 🔍

Query Syntax

Showing 1 to 10 of 19 results

Name
🔼
🔽



INACTIVE
🔗 en
☆ 0
🔗 0


Adjustable height desk

I bought an adjustable-height desk with analog switch that allow to manually move desk...

Pawel Zentala

🔗 IoT, table, desk

🔗 Image



ACTIVE
🔗 en
☆ 0
🔗 0


Art easel

Self made art easel attached to the edge of the table.

Fablab Bratislava at Slovak Centre of Scientific and Technical Information (SCSTI)

🔗 wood, art-easel

🔗 Image




INACTIVE
🔗 en
☆ 0
🔗 0

Augmented Reality Board

AR board makes it #safetoplay across the table or across the world. Simply video call your...

shannonmassingill

🔗 safetoplay, 3d-print-cnc-laser-engraving, augmented-reality



ACTIVE
🔗 en
☆ 0
🔗 0





Beanstalk Changing Table

COLLAB WANTEDThis is just an idea as I don't have the ability to move forward...

Jean-Guy Rouleau

🔗 IoT, kids, baby-furniture

🔗 Image

NAME	DESCRIPTION	LICENSOR	LICENSE	🔗	🔗	☆	🔗	🔗
	<p>"Merry Christmas" Tree</p> <p>A quick project that generated our Sesons Greeting postcard. This publication is part of it. Following the...</p>	FabLab Benfica	CC0-1.0	🟢	en	0	0	🔗
	<p>"STUDENT CHEST"</p> <p>Este e o nosso Banco de Benfica. Um projeto no âmbito da disciplina de Design de Produto. Inspirado...</p>	CatarinaFreitas28		🟢	pt	1	0	🔗
	<p>#LS7</p> <p>Architecture is poetry. People can't read.</p>	Make-a-thon		🔴	en	0	0	🔗
	<p>CC-BY-SA</p> <p>modularity, cnc, 3d-print-cnc-laser-engraving, augmented-reality</p>							🔗

Export Results

10 / Page
Go
< 1 2 >

Copyright © 2022 André Lehmann

[Terms of Service](#) ·
 [Privacy Policy](#) ·
 [Report an Issue](#) ·
 🌙

Figure 5.9.: Product Search Results Page

Query Syntax ^	
General Expressions	
<code>word1 word2</code>	Full Text Search
<code>"word1 word2"</code>	Exact Text Search
<code>"some text"</code> <code>'some text'</code>	Exact Text Search
<code>some * text</code>	Wildcard (Any Text or Phrase)
<code>(expr1 OR expr2)</code> <code>(expr1 expr2)</code>	Boolean OR
<code>expr1 AND expr2</code> <code>expr1 & expr2</code>	Boolean AND
<code>NOT expr</code> <code>-expr</code>	Boolean NOT
<code>(expr1 expr2)</code>	Grouping of Expressions
<code>operator:value</code>	Filter Operator
Basic	
<code>name:beehive</code>	Product Name
<code>description:beehive</code>	Product Description
<code>language:en</code>	Language used for documentation
<code>version:1.0.0</code>	Version of latest release
<code>website:</code>	Product Website
<code>starCount:>0</code>	Number of stars
<code>forkCount:>0</code>	Number of forks
General Operators	
Operators allow searching for specific product properties. The allowed operations on each Operator depend on the property type. <code>value</code> can be text, number(e.g. 3.141), date/time (e.g. 2020-01-01 10:30:45), or time duration (format: , e.g. 1w2d).	
<code>operator:value</code> <code>operator:'word1 word2'</code>	Equal Text
<code>operator:==value</code>	Equal
<code>operator:!=value</code>	Unequal
<code>operator:<value</code>	Less
<code>operator:<=value</code>	Less or Equal
<code>operator:>=value</code>	Greater
<code>operator:>value</code>	Greater or Equal
<code>operator:value..value</code>	Range (inclusive, e.g. 2..*, 2022-01-01..2022-04-01)
License	
<code>has:license</code>	Indicates whether Product is licensed
<code>has:hasAdditionalLicenses</code>	Indicates whether Product has other licenses
<code>license:CC-BY-SA-4.0</code>	License SPDX Identifier

Figure 5.10.: Query Syntax Cheat Sheet

A Can-Do Attitude

ACTIVE
en
0
N/A

#diy
#aluminium
#reuse

Distributed Design Studio
+othertodaystudio

Exploring the qualities and properties of aluminium from drinks cans, and the creation of a DIY Christmas tree ornament.

Files

Image
Readme

Repository

1

2

3

Copyright © 2022 André Lehmann

[Terms of Service](#)
[Privacy Policy](#)
[Report an Issue](#)

Figure 5.11.: Product Details Page

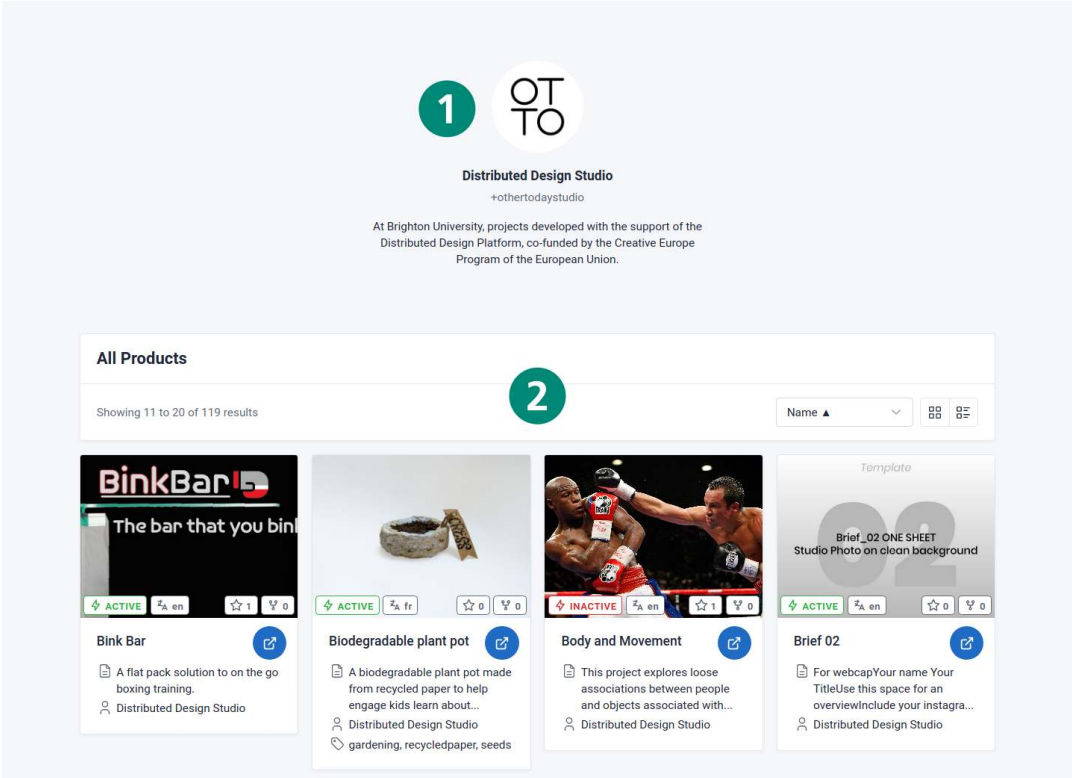


Figure 5.12.: Licensor Details Page

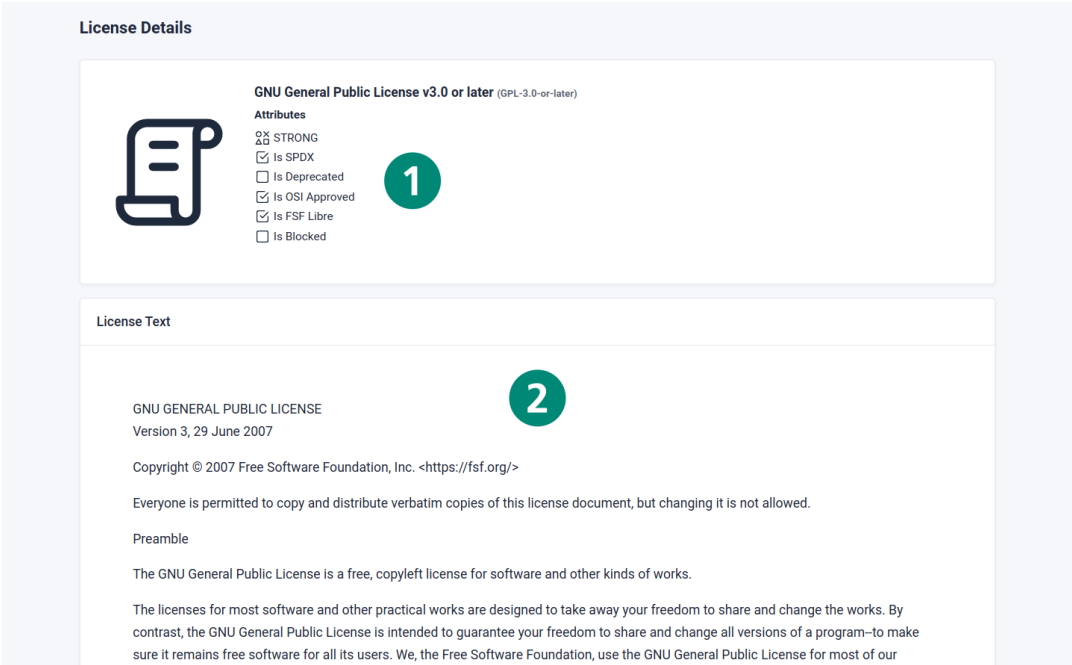


Figure 5.13.: License Details Page

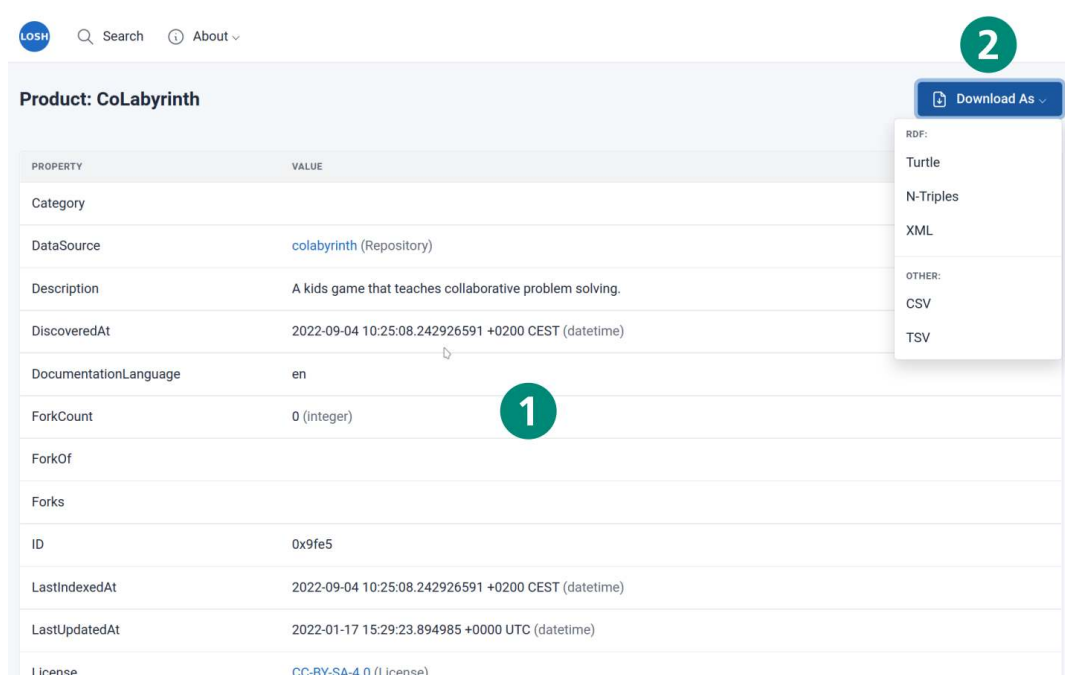


Figure 5.14.: RDF Resource Page

Multiple libraries and frameworks are involved in driving the web interface. First, there is Fiber. Fiber is used as an HTTP framework for the application. When the application is executed, Fiber starts a multithreaded web server and listens for HTTP requests. Once a request reaches the server, Fiber reads the request's URL path and looks for a matching route. The route determines the controller to be used for handling that specific request. If no route was found, Fiber returns a *404 - Not Found* HTTP error code. The selected controller contains the logic for handling the request. It parses any path and/or query parameters, calls the business logic for the desired operation, encodes the results in HTML or another format, and sends a response to the requesting client.

We chose Tabler as a UI kit for the service. It offers a wide variety of ready-to-use UI components and templates in a coherent, modern style, making it a suitable and easy-to-use solution for our service. However, as the UI kit is developed for use with the static site generator Jekyll, it does take some effort to integrate it into our Go application. Jekyll uses the Liquid templating engine and adds its own filters and tags for additional functionality and ease of use. The templates provided with Tabler are all written in Liquid and use some of these additional Jekyll filters and tags. Therefore, we needed a Liquid implementation and some of the aforementioned Jekyll-specific filters and tags in Go to use the templates without rewriting them in a different templating language. Fortunately, there is an open source implementation of Liquid and Jekyll in Go. The Go Liquid implementation was integrated into our application along with some selected filters and tags and is now used to render the Tabler-provided templates. It has to be said that the Go implementation of Liquid is not as mature as the original Ruby implementation and thus required us to add and fix some of its functionality to make it work.

5.4.1. Product Search

The *Product Search* is the main functionality offered by the web service. It allows users to search for OSH products of their interest by issuing a *Search Query* and receiving the results through the web interface. This section covers the implementation specifics of the product search functionality.

The execution flow of the product search is depicted in Figure 5.15. As it can be seen, the search involves multiple processing steps and a call to the database. Generally, it can be said that the database is the component that performs the search, filters and sorts results, and returns them in a paginated manner. The backend acts more or less as a translation layer for the search query. In words, the execution flow is as follows:

1. The user uses their browser to issue a search by sending a query to the web service.
2. The service receives the request and handles it according to the rules as depicted in Figure 5.16. Generally, this involves any kind of error handling.
3. As a next step, the server parses the search query and turns it into a AST representation. The section parsing of the query is explained in more detail in the Query Parsing section.
4. The parsed search query cannot be understood by the database and must therefore be translated. The server takes the AST query presentation and creates a DQL database query. The translation is covered in the DQL Database Query section.
5. Now, the server executes the DQL query by sending it to the database and waiting for results.
6. Once the database returns the results, the server can continue to process the results.
7. For the described search request, the results need to be encoded as an HTML page. The server generates the results HTML page and returns it to the user who requested it.

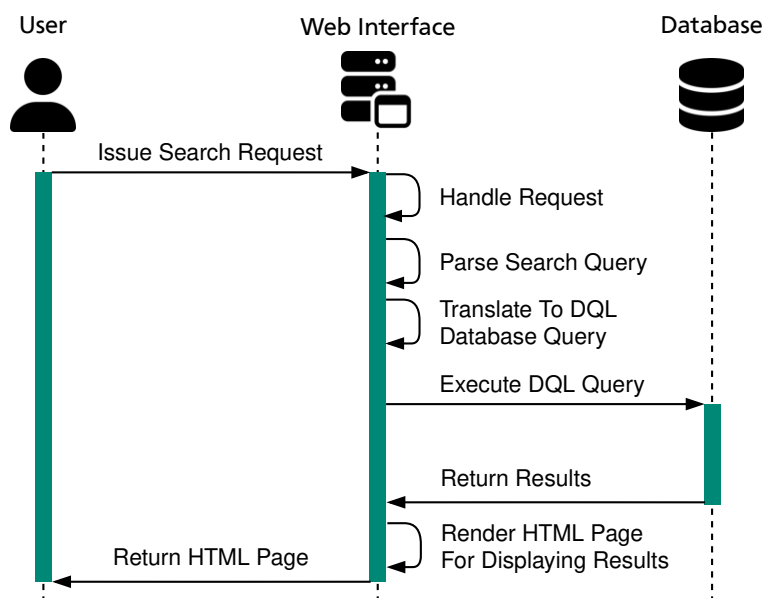


Figure 5.15.: Product Search Sequence Diagram

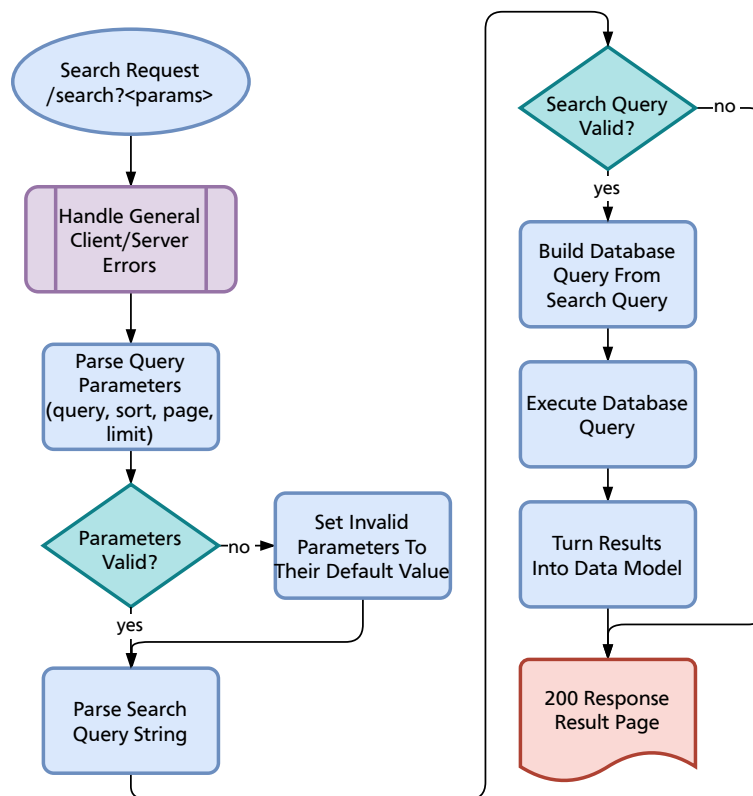


Figure 5.16.: Handle Search Request Flowchart

Search Query

As stated in Section 4.4.2, the *Search Query* is a phrase, a combination of keywords and operators used to retrieve specific information from a data set. In the case of the thesis, the query is intended for searching and finding OSH products that are indexed by the crawler and stored within the database. It is designed to be easy for new-time users to understand and yet powerful enough for advanced users to find OSH products based on complex questions.

Query Definition

The power of the query syntax stems from the full-text search and wide variety of Search Operators built into the service. The syntax is inspired by the syntax of different existing search engines and services on the web, notably Google and GitHub. Users familiar with the Google syntax should not have too big of a struggle and may only need a short time to adjust to the syntax presented by us.

The syntax used by us is explained using a few examples:

```
(tables OR desks) has:license starCount:>0
```

This example translates to a search for a product that has something to do with *tables* or *desks* and has a license and has at least one star. The syntax showcases the ability to combine multiple keywords and

operators with different logical operations (AND/OR/NOT). The text terms *tables* and *desks* are treated as full-text search terms. By the rules of full-text search, the terms are preprocessed as explained in section 4.3. If no logical operation is supplied between the individual terms, then AND is implied. The OR condition is grouped using parentheses (). The grouping is required because the implied AND condition has a higher priority than the OR condition. The operations priority is as follows:

1. Parentheses (Grouping)
2. NOT
3. AND
4. OR

The operators always have the structure `operator:expression`, where the expression refers to a type of value comparison. In the given first example, the `has` operator checks if the data property for the value `license` is defined and non-null. The `starCount` operator accepts numbers and allows comparison operations such as *equal to*, *greater as*, etc.

```
("car engine" OR "electric * motor") is:licensePermissive createdAt:2019-01-01..2022-01-01
```

This example answers the following question: What products do have a permissive license, are created in the period from 2019 to 2022, and have something to do with engines? The terms enclosed with double quotes are searched literally. So the search engine will search for products containing the exact string `car engine` or `electric * motor`, where the asterisk (*) acts as a placeholder that will accept any phrase and thus match `electric tesla motor` for example. The text search is generally case-insensitive.

A summarization of the full query syntax and available operators can be found in the appendices in Section D.

Query Parsing

To analyze and evaluate the search query, a query parser was implemented. The implementation is based on the parser generator *Participle*. The grammar to analyze the syntax is not defined by itself but rather directly combined with the AST data model. That data model is represented as plain Go structs. The query grammar is defined and directly mapped to each field using Go's struct tag annotations. An excerpt of the grammar definition using the Participle approach is shown in Listing 2.

For the sake of presenting, the grammar was converted to EBNF and is showcased in Listing 3. Figure 5.17 shows the same grammar in the form of a railroad diagram. The non-terminals, enclosed in angle brackets (<>), are defined by the lexer and are omitted for brevity.

The grammar captures the general structure of the query language. What is not reflected in the grammar are the operator-specific values. For example, the `createdAt` operator takes a date or time duration and allows for different comparison operations. For instance, `createdAt:<6m` expresses a creation date of less than 6 month. On the other hand, the operator `license` only accepts input as strings and cannot be compared using lower/greater expressions. Instead of defining a grammar for each operator individually, the parsing is deferred to a later stage in the processing pipeline. The parser simply captures the value for each operator expression as text.

Due to the limitations of generated parsers regarding error handling, the grammar is more convoluted than necessary. The reason is that a user may provide an invalid syntax, triggering a parser error if the grammar is too strict. So instead, the parser tries to handle invalid operator syntax by interpreting simply as a full-text search. The goal is to deal with syntax errors gracefully and not confront the user with every little mistake they make. The behavior is designed to be similar to Google's approach, which focuses on providing a robust search.

```
type Operator struct {
    Name      string      `@Identifier ":"`
    Comparison *Comparison ` ( @@`
    Range     *Range      ` | @@`
    Value     *Text       ` | @@ )?`
}

type Range struct {
    OpenStart bool    `( @"*"`
    Start     *string ` | @QuotedString | (@String | @Identifier | @Keyword | @Number
    | @Specials)+ DoubleDot`
    OpenEnd   bool    `( @"*"`
    End       *string ` | @QuotedString | (@String | @Identifier | @Keyword | @Number
    | @Specials)+)`
}
```

Listing 2: Example of Go struct annotations including the query grammar


```

Query = OrCondition ( <whitespace> ("OR" | "|") <whitespace> OrCondition)* .
OrCondition = AndCondition ( <whitespace> ("AND" | "&") <whitespace> AndCondition
    | ( <whitespace> AndCondition))* .
AndCondition = ("NOT" <whitespace> AndCondition) | ("-" AndCondition) | Expression .
Expression = Operator | Text | ("(" Query ")") | <whitespace> .
Operator = <identifier> ":" (Comparison | Range | Text)? .
Comparison = (("=" "=") | ("!" "=") | ("<" "=") | "<" | (">" "=") | ">") Text .
Text = <quotedstring> | (<backtickquotedstring> | ((<identifier> | <number> | <string>
    | <specials>) (<identifier> | <keyword> | <number> | <string> | <specials>)*)) .
Range = ("*" | <quotedstring> | <string> | <identifier> | <keyword> | <number>
    | <specials>)+ <doubledot> ("*" | <quotedstring> | <string> | <identifier>
    | <keyword> | <number> | <specials>)+ .

```

Listing 3: Query Syntax EBNF

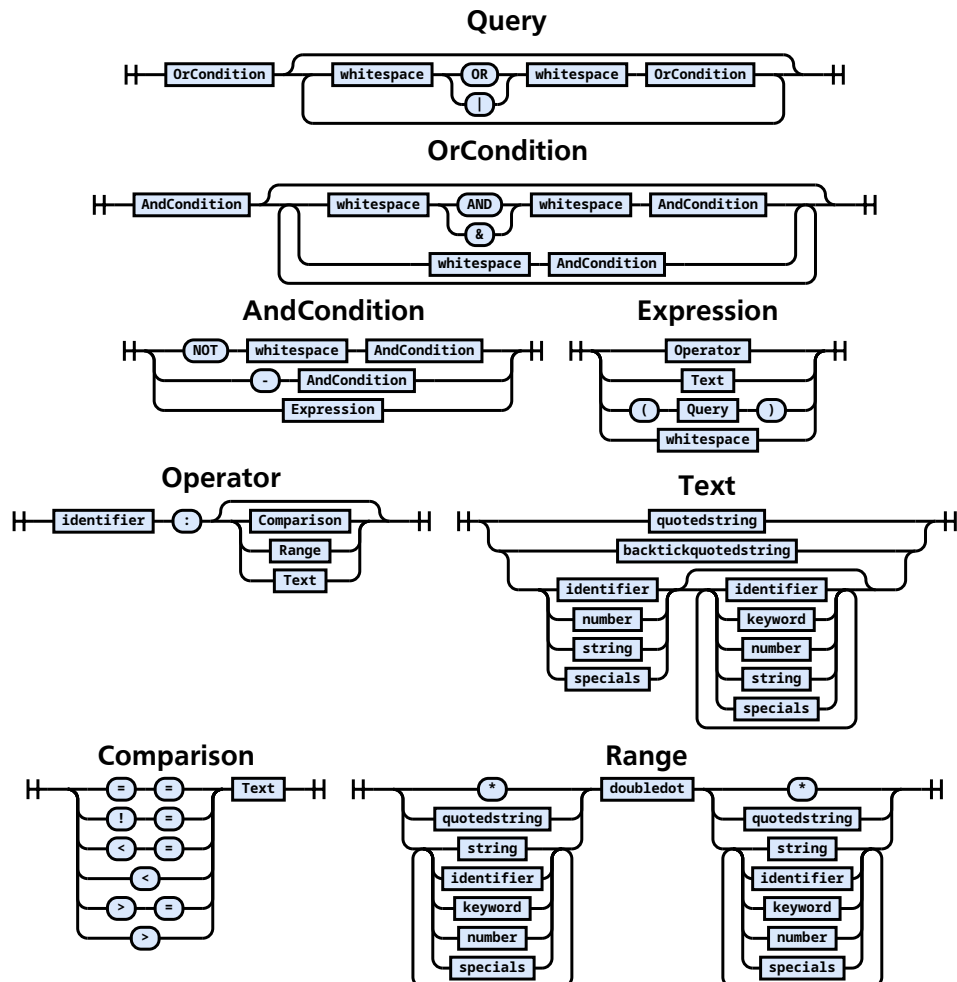


Figure 5.17.: Query Syntax Railroad Diagram

The following is an example of a valid but suboptimal search query. The parser analyzes the query and turns it into an AST, as shown in Listing 4.

```
(table desk) (createdAt:2019-01-01..2022-01-01 is:)
```

```
1  &parser.Query
2    Or: []*parser.OrCondition
3    And: []*parser.AndCondition
4      Operand: &parser.Expression
5      Sub: &parser.Query
6        Or: []*parser.OrCondition
7        And: []*parser.AndCondition
8          Operand: &parser.Expression
9            Text: &parser.Text
10             Words: &"electric",
11             Operand: &parser.Expression
12               Text: &parser.Text
13               Words: &"motor",
14             Operand: &parser.Expression
15             Sub: &parser.Query
16               Or: []*parser.OrCondition
17               And: []*parser.AndCondition
18                 Operand: &parser.Expression
19                   Operator: &parser.Operator
20                     Name: &"createdAt",
21                     Range: &parser.Range
22                       Start: &"2019-01-01",
23                       End: &"2022-01-01",
24                   Operand: &parser.Expression
25                     Operator: &parser.Operator
26                       Name: &"is",
```

Listing 4: Resulting AST for the search query: (table desk)
(createdAt:2019-01-01..2022-01-01 is:)

After analyzing the input string and turning it into an AST, a clean-up routine is performed to remove empty terms and simplify the AST. Of course, the AST could be used as is, and a valid database query could be generated from it. However, considering the overhead for executing a fairly complex database query, it is more efficient to simplify the data model in an early stage of processing. The cleaned and simplified version of the aforementioned AST is presented in Listing 5. As we can see, the depth of the tree has been reduced. Furthermore, the terms *electric* (Listing 4 line 10) and *motor* (Listing 4 line 13) have been combined into a single term (Listing 5 line 15). The latter is passed directly to the database for a full-text search.

```

1 &parser.Query
2   Or: []*parser.OrCondition
3   And: []*parser.AndCondition
4     Operand: &parser.Expression
5     Operator: &parser.Operator
6     Name: "createdAt",
7     Range: &parser.Range
8     Start: &"2019-01-01",
9     End: &"2022-01-01",
10    Operand: &parser.Expression
11    Operator: &parser.Operator
12    Name: "is",
13    Operand: &parser.Expression
14    Text: &parser.Text
15    Words: &"electric motor",

```

Listing 5: Cleaned and simplified AST for the search query: (table desk)
(createdAt:2019-01-01..2022-01-01 is:)

DQL Database Query

DQL is Dgraph's query language that allows for full data access and is used to retrieve the results for the product search.

Performance wise it is infeasible to retrieve all the data from the database and performing the search in the service backend. It would require a large amount of memory resource to keep the product data in memory all at once and would be highly CPU intensive to perform a full read every time a search is performed. The data could be kept and cached in memory, but then a cache invalidation strategy must also be implemented to keep the data from the database and the data in the backend's memory in sync. Thus, all the searching, filtering and sorting is performed solely by the database. The backend is responsible for parsing the search query, creating and running a database query, and processing and presenting the search results.

The AST of the parsed and simplified search query has to be turned into a DQL database query. This is done by a custom encoder, that takes the parsed AST and turns it into a serialized DQL query. At the time of writing there were no production ready solutions for creating DQL queries programmatically, therefore a custom solution was developed. The encoder translates the concepts of the search query into the equivalent DQL constructs. Listing 6 and Listing 7 show the translation result for the example search query (license:MIT OR license:CC-BY*) furniture has:image.

```

1 query q($a1: string, $a2: string, $a3: string, $a4: string, $a5: string, $first: int, $offset: int) {
2   # license:MIT - translated to full-text search in License.xid field
3   v1 as var(func:type(Product)) @cascade {Product.release {Component.license
4     @filter(allofterms(License.xid, $a1)) {uid}}}
5   # license:CC-BY* - translated to regular expression search in License.xid field
6   v2 as var(func:type(Product)) @cascade {Product.release {Component.license
7     @filter(regex(License.xid, $a2)) {uid}}}
8   # license:MIT OR license:CC-BY* - combines results of aforementioned license searches
9   v3 as var(func:uid(v1,v2)) {uid}
10
11   # has:image - translated to 'has' filter for Component.image field
12   # logical 'AND' combination created by simply filtering previous matches
13   v4 as var(func:uid(v3)) @cascade {Product.release @filter(has(Component.image)){uid}}
14
15   # furniture - translated to full-text search in Product.name, Product.description
16   # and Tag.name
17   v5 as var(func:uid(v4)) @filter((alloftext(Product.name, $a3))
18     OR (alloftext(Product.description, $a4))) {uid}
19   v6 as var(func:uid(v4)) @cascade {Product.tags @filter(alloftext(Tag.name, $a5)) {uid}}
20   v7 as var(func:uid(v5,v6)) {uid} # OR combination
21
22   # define variable for ordering by product name
23   var(func:uid(v7)) {order as Product.name}
24
25   # data selections and ordering
26   q(func: uid(v7), first: $first, offset: $offset, orderasc: val(order)) {
27     # data selectors
28     Product.name
29     ...
30   }
31 }

```

Listing 6: Search query translated to DQL query

```

1 {
2   "$a1": "MIT",
3   "$a2": "/CC-BY(\\s*\\S*)?/i",
4   "$a3": "furniture", "$a4": "furniture", "$a5": "furniture",
5   "$first": 100,
6   "$offset": 0
7 }

```

Listing 7: Input Variables for translated DQL query

5.4.2. Semantic Web

This work focuses primarily on offering a usable and satisfactory OSH search engine. Providing access to the dataset via SW technologies is only a secondary goal. As such, the implementation of SW technologies is kept rather simple and only features basic functionality. RDF is used for data interchange. As described in Section 2.2, RDF is a key technology of the SW and is used to describe and exchange data in graph form. Because the service already stores and handles product information as graph data, it is relatively easy to translate it into an RDF format. The data model of the service is inspired by the LOSH specification (Section 3.4) but is not quite the same. Therefore, there doesn't exist a well-defined RDF schema which is required to describe the RDF resources. So, we chose to omit the schema and settle on a free-form representation for now.

Users can look up data entities by sending a request to the RDF service's resource endpoint. A unique ID and URI identify the entities. The execution flow is depicted in Figure 5.18 and Figure 5.19. In words, the execution flow is as follows:

1. The client requests a resource from the service RDF endpoint (`/rdf/resource/<ID>`) and specifies the desired return format using the `Accept` and/or `Accept-Language` HTTP headers.
2. If the format specified by the user is known to the server, the server dereferences the resource and responds with an HTTP redirect to the location responsible for handling the resource representation and data exchange. That location would be either a HTML page (`/rdf/page/<ID>`) or a machine-readable RDF format (`/rdf/data/<format>/<ID>`).
3. The client then has to follow the redirect to get to the data by sending a new HTTP request.
4. At last, the server receives the request and either creates an HTML page for viewing the resource with a web browser or, if requested, returns the data in an RDF serialized format.

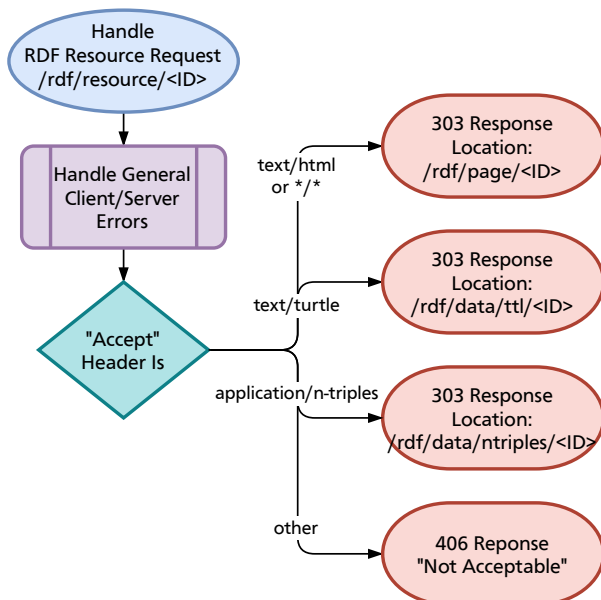


Figure 5.18.: Handle RDF Resource Request Flowchart

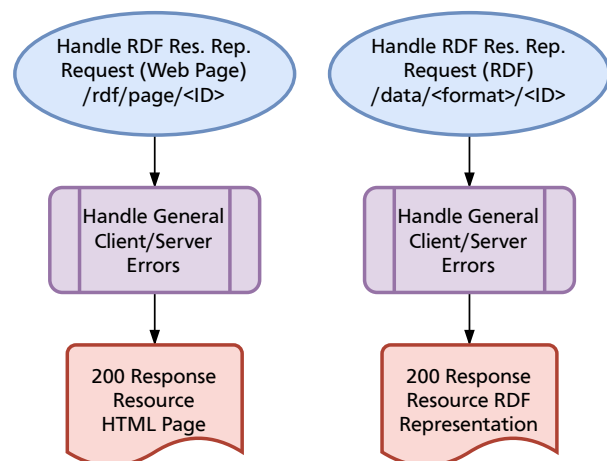


Figure 5.19.: Handle RDF Resource Representation Request Flowchart

6. Evaluation and Validation

6.1. Introduction to Usability Testing

The concepts of usability and user experience are defined by the *International Organization for Standardization* as [27]:

- **Usability:** "extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use."
- **User experience:** "person's perceptions and responses resulting from the use and/or anticipated use of a product, system or service."

In order to work well, a system must provide a function to solve a goal, be easy to use, and offer a good user experience. A poorly designed product or service exhibits low usability and a poor user experience leading to confusion, annoyance, stress, and ultimately to delays, erroneous decisions, and failures in the user's workflow.

Usability Testing is an activity in the product design process to evaluate the design of a product. The goal of the testing is to check if the product or service meets the requirements and satisfies the user's needs. The results help identify problems in the design, implementation, and other aspects of the product or service and provide a set of actions to improve the system.

Usability testing involves a researcher/evaluator knowledgeable in the product or service domain and a sufficient number of users with preferably diverse backgrounds, skills, and preferences. The participants are asked to solve a set of tasks to interact with the product or service. In terms of software, the users have to interact with different aspects of the applications to solve the given tasks. They have to concern themselves with the software's available views, forms, and modals while telling their experiences with it. This method is referred to as "thinking aloud." The users speak about their expectations, reasoning behind any actions, and experience with the application. The evaluator takes a passive role in observing the participants and taking notes. Usually, the testing is recorded to capture every moment of the interaction and allow a more in-depth performance evaluation.

Usability testing can be differentiated into qualitative and quantitative testing. The focus of qualitative testing is on collecting insights into how users interact and use a product or service. This method helps to reveal functional or usability problems. On the other hand, quantitative usability testing focuses on collecting metrics for benchmarks. For example, the success/failure rate is measured, or the total required time for solving a task.

For this thesis, a qualitative usability testing has been performed. The goal was to determine the user's satisfaction in working with the developed web service and point out chances for future improvements. The results are presented in Section 6.3.

6.2. Methodology

The usability testing was conducted with five previously interviewed volunteers from the requirement assessment phase. The profiles of the participants can be found in the appendices in Section C. The participants were asked to solve a set of prepared tasks (Section B in appendices) and instructed to think out loud while interacting with the application. After each task, the participants were asked to elaborate on their experience and describe any difficulties they might have encountered. During testing, the communication and commentary from our side have been kept to a minimum to get unbiased, genuine insights into the first-time experience of the participants with the software. Only after the testing has been concluded more in-depth information about the development has been offered, and further questions have been answered.

The interviews were analyzed by going through the recordings and extracting relevant issues. The issues are identified by listening to the verbalized thoughts and observing the participant's behavior and interaction with the applications. For example, signs of issues can be:

- Direct verbal articulation of problems
- Expressions of surprise
- Expressions of frustration, struggle and uncertainty
- Needing multiple attempts and performing unnecessary actions to solve a task
- Assistance is required to solve a task
- User missing information or steps to solve a task
- Mental overloading due to complex information
- Unmet assumptions because the application violates common standards and conventions
- Unexpected results even if the tasks were correctly solved

Not all issues strictly stem from the participants' performance or are even noted as issues by the participants. For example, when going through the recordings, some problems that were legitimate bugs in the software became apparent to us but remained unnoticed by the users.

Not every issue is equally important. Rating the issues and prioritizing them helps to identify the most crucial problems and gives a measure of importance to resolving the issues. For this purpose, the issues identified in the usability testing have been rated and categorized into the following categories [28]:

- **Positive:** Positive issues are qualities of the application, functional or presentational, that the participants appreciate. Improvements are not ruled out, but in the current state, no action is required. The qualities can be kept as is.
- **Idea:** Ideas reflect the participants' desire to change the application's functionality or appearance. These suggestions target properties that were not previously in design scope and can serve as opportunities to improve the user experience.
- **Bug:** Bugs are functional or visual application flaws. Anything that doesn't work or appear as designed can be considered a bug.

- **Minor:** A short moment of hesitation and uncertainty of the participants indicate a minor issue. The application works as designed, but the participants need a brief moment to determine how the controls work.
- **Major:** Major issues substantially delay and frustrate the participants. However, most of the time, the participants can recover from the complication and solve the task.
- **Critical:** A critical issue is indicated by an unsolvable task or major annoyance to the participants.

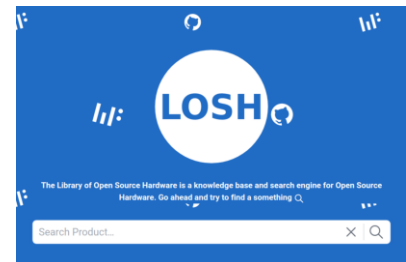
The issues were documented and indexed. The documentation contains an issue description of what happened, an image that shows the location of where the issue occurred, the category and a possible solution for resolving the issues.

6.3. Results

This section contains the results of the usability testing. As explained, the issues are categorized, sorted and indexed. For each addressable issue a redesign proposal is offered. This proposal explains a possible solution and how it might be implemented.

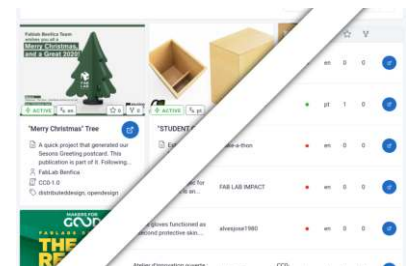
I1 Positive: First Impression

On the homepage, the first visible section is the *hero* section, with its notable blue animated background. The hero is simplistic, visually pleasing, and was overall well received by the participants (MH, JP, PJ). As stated by PJ, the first impression is an important factor for mainstream adoption. Users are more likely to use the service and get familiar with it when the user interface is visually pleasing and easy to use.



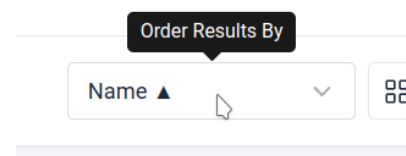
I2 Positive: Visual Pleasing Interface

The card style gallery and table view for displaying the search results were praised for their pleasant and clean appearance. Especially the prominent product image is said to help the showcasing tremendously compared to the OPENNEXT LOSH Demonstrator. (MH, PJ, TW)



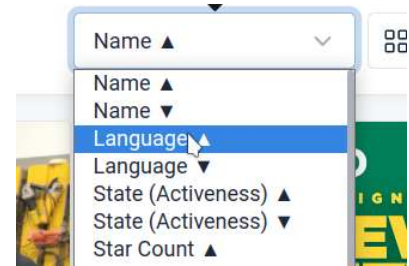
I3 Positive: Interface Explained Through Tooltips

All elements do have a tooltip and explain their purpose. Thus, users do not have trouble understanding what the elements are used for and feel confident using the service. (PJ, JP, TW)



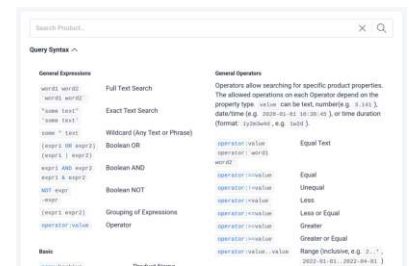
I4 Positive: Simplistic Ordering

The results are orderable by a multitude of different properties. Defining the order is easily done using the select box or in the table view by clicking on the appropriate table headers. (MH, JP, OS, PJ, TW)



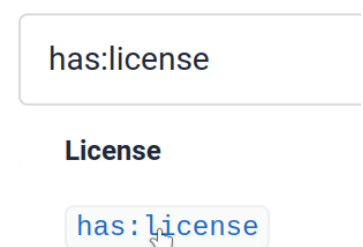
I5 Positive: Query Syntax Cheat Sheet

The query syntax cheat sheet offers a good overview of the available operators and how to use the syntax. The included examples give a hint about how each operator might be used. Even participants with little experience building queries report that it was reasonably easy to create working queries. (MH, JP, OS)



I6 Positive: Clickable Query Syntax Cheat Sheet

Clicking on a syntax option in the query syntax sheet adds it directly to the search query input. Thus, the user can create queries by clicking on the desired filters. This feature was well received. (MH, JP, PJ, TW)



I7 Positive: Long Description Text

Only the first few lines of description texts are displayed by default. Longer description texts are cut off. The full text is shown by hovering over the description with the mouse cursor. This feature was well received. (MH, JP, PJ)



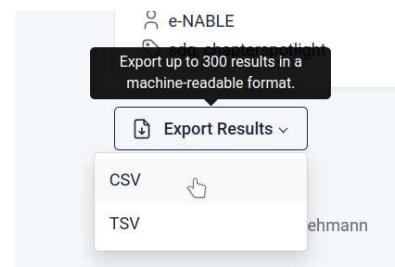
I8 Positive: Activeness Indicator

The product *Activeness* indicator on the results was noted and was well received. (MH, JP, OS, TW)



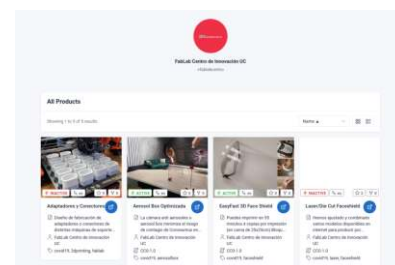
I9 Positive: Export of Results

The service offers the option to export search results as CSV. The CSV export contains a great deal of product information. (MH, JP)



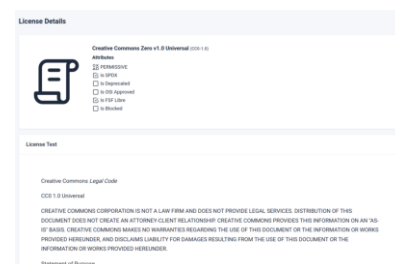
I10 Positive: Licensor Details Page

The licensor details page featuring basic information about the user or group and an overview of all the products of that user or group was deemed useful. (MH)



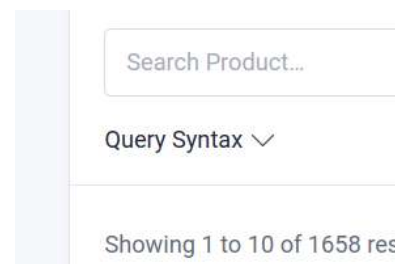
I11 Positive: License Details Page

The license details page offering an overview of the licenses, including the license text, information about OSI approval, and more, was well received. (MH)



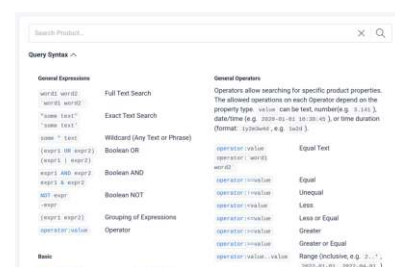
I12 Idea: Query Syntax Button Naming

The button that opens the syntax cheat sheet should be named something different than *Query Syntax*. JP and TW suggested *Advanced Query*. MH opposes the name *Advanced Query* because he associates the term with a search form with multiple input fields and not the syntax cheat sheet. Additional PJ suggested moving the button to the right side, where the inputs of the other query options are located.



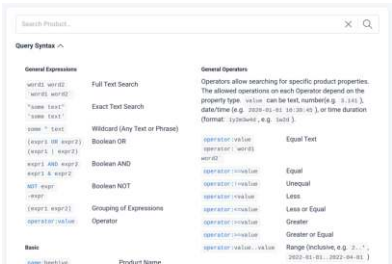
I13 Idea: Simplified Query Syntax Cheat Sheet

Instead of showing the full extent of the search capabilities in the query cheat sheet, it might be better to show only a set of common operators and reveal the full cheat sheet only if the user clicks on an additional option. (JP, PJ)



I114 **Idea:** Query Syntax Cheat Sheet Examples

The idea of adding one or more example queries to the query cheat sheet was proposed. It is argued that an example helps to convey the query possibilities in a short and easily digestible manner. For example, the query (table OR desk) has:license createdAt:<6m demonstrates the usage of multiple operators and logical combinations, which already would be more than enough for most use cases. (MH, JP)



I115 **Idea:** List of Operator Values

It was suggested to add a list of all available options for operators to the syntax cheat sheet. It has been said, that operators, such as repositoryHost, can benefit from a list of all options being directly available to the users. (TW)



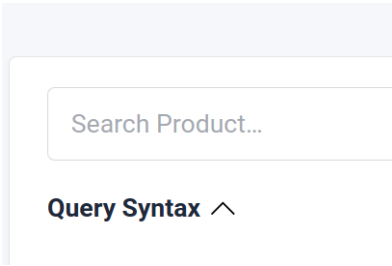
I116 **Idea:** Popularity instead of Stars

Not all platforms include the notion of stars and forks but may instead use views as an index of popularity. A suggestion was made to replace stars with a generalized popularity measurement. (JP)



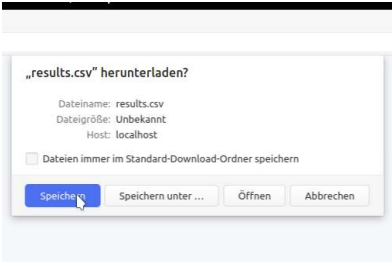
I117 **Idea:** Dedicated Filter Inputs

Less technical-versed users might be intimidated and discouraged by the query syntax. Offering dedicated filter inputs (select boxes, slider, etc.) might lower the entry barrier for using the service and enable less technical-versed users to operate the search as desired. (JP, PJ)



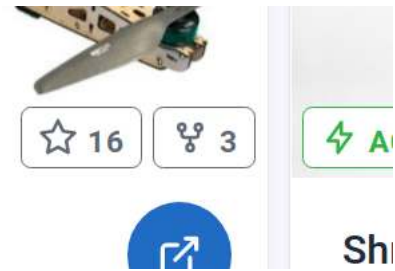
I118 **Idea:** Timestamp in Export File Name

The export file should contain a timestamp in the name so that it is immediately apparent when it was exported. (MH)



I19 Idea: Forks Indicator

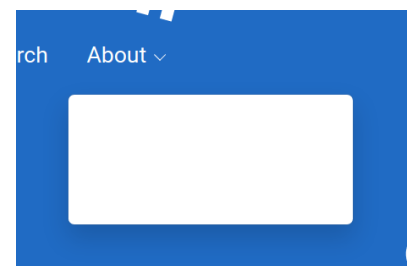
The notion of forks is not something that every hardware platform provides. Therefore, there will potentially be a lot of products that do not have such kind of information. **OS** doesn't see any value in displaying the number of forks on the details page. He argues that it is rarely of any interest to the users. On the other hand, **TW** uses the number of forks as an indicator for adoption and thus wants to keep it.



I20 Bug: Homepage Menu Drop-Down

On the homepage, the *About* drop-down menu does have white-colored text on a white background, making it unreadable. (**MH, OS**)

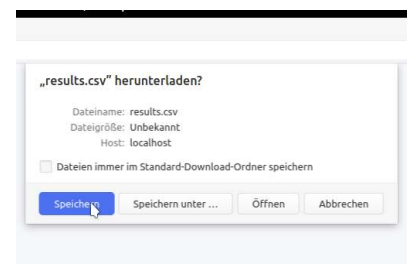
To make the text readable again, the CSS definition for the homepage must be corrected.



I21 Bug: Unexpected Results for Exact Text Search

An exact text search for the license CC-BY-4.0 using the query `license:"CC-BY-4.0"` also returns results for the CC-BY-SA-4.0 license. This is not intended and confused the participants. (**MH**)

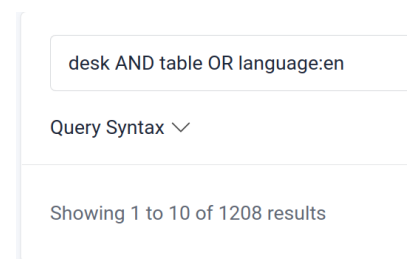
The parser correctly recognizes the need for an exact text search, but the database query is built incorrectly. The translation must be checked and corrected.



I22 Bug: Unexpected Results for Query With OR and AND Conditions

The combination of OR and AND conditions resulted in an incorrect database query, and in turn, completely unrelated results were returned.

The `table OR desk has:image` (same as: `table OR (desk AND has:image)`) is translated to a database query equivalent to `table OR desk OR has:image`. The query encoder must be corrected.



I23 Bug: Caching of Assets

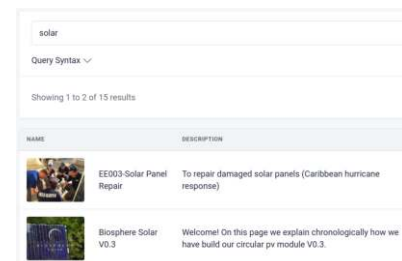
Server caching of assets (CSS, JS, etc.) leads to some assets not being properly delivered. As long as the browser has the assets cached, the user experience won't be affected. Although, when requesting the assets anew, the server will only respond with a *304 - Not Modified* status code without delivering the actual content, thereby breaking the application. (OS)

The caching mechanism might not handle the *If-Modified-Since* header correctly, which signals the server to check modification timestamps and respond with a Last-Modified. Correcting the header check might resolve the issue.

304 - Not Modified

I24 Bug: Order By License

When sorting by license, the results that do not contain a license are eliminated from the results set. Expected is that results without a license are kept and sorted to the back. This leads to an unexpected number of results being displayed on a page. In some cases, simply a server error is returned. This issue stems from undesired behavior of the database, which has been discussed¹ before but not entirely resolved yet.

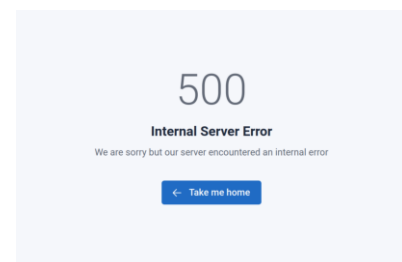


NAME	DESCRIPTION
EE003 Solar Panel Repair	To repair damaged solar panels (Caribbean hurricane response)
Biosphere Solar V0.3	Welcome! On this page we explain chronologically how we have build our circular pv module V0.3.

I25 Bug: Beyond Last Page

When going beyond the last page of results, it is expected to be greeted with an empty page. But instead, a page with all results without any limitations is returned. The only limitation seems to be the database itself, which stops with an error if the resulting data gets too large. In the latter case, the application returns a *500 Internal Server Error* to the user.

Specifying an *offset* parameter in the database query that is larger than the number of resulting elements, then the *offset* and *first* parameters are seemingly ignored altogether, and all results are returned. This behavior of the database must be changed.



¹ <https://discuss.dgraph.io/t/expected-behaviour-for-sort-queries/7157>

I26 **Bug:** Floating Numbers for Time Durations

When using a floating point number in time durations (e.g. 0.5y), the operator is simply ignored and incorrect results are returned. (JP)

The parser for time durations must be changed from working only with integer numbers to floating point numbers.

createdAt:<0.5y

Query Syntax ▾

Showing 1 to 10 of 1658 results

I27 **Minor:** No Feedback When Clicking on Syntax Option

When using the service for the first time and clicking on a syntax option in the query cheat sheet, it is not immediately apparent that the option is added to the query input if it is currently not in the view. (MH, OS)

This issue can easily be resolved by providing feedback when clicking an option that tells the user it was appended to the search query.

License

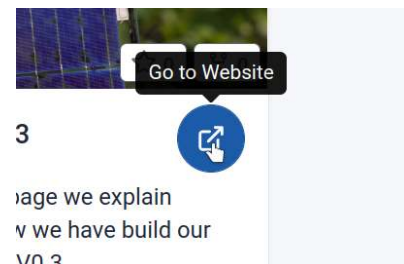
has:License

has:hasAdditionalLi

I28 **Minor:** Different Pages When Clicking on Product Image and Website Button

Clicking on the product image and clicking on the blue button of a result lead to different pages. The former leads to the product details page, the latter to the product repository (website). The participants were confused at first. It must be noted that the product details page was unavailable at the time of the usability test. (MH, JP, OS, TW)

The button on the results overview is intended as a quick link to skip the details page if desired. The details page itself must also provide a link to the product repository so that the users can get to the original files from there.



I29 **Minor:** Too Long Operator Names

The operators listed in the syntax cheat sheet are considered too long and should be shortened. (PJ)

The operator names can be shortened by using abbreviations. This would make them faster to type and maybe also easier to remember.

is:licenseStrong

is:licenseWeak

is:licensePermissive

repository:Wikifactory

repositoryOwner:

I30 Minor: Confusion Because of Aliases

Aliases for some syntax options create confusion amongst the participants. The aliases were not explicitly labeled as aliases; thus, the participants expected a difference but did not know what that difference would be. (MH, JP)

Aliases do not create additional value for the user. Simply removing the aliases should be sufficient.

```
name: beehive
description: beehive
language: en
documentationLanguage: en
version: 1.0.0
website:
```

I31 Minor: Unclear Time Duration Definition

At first glance, it seems unclear how to correctly define a time duration. None of the participants saw the available description on that matter in the *General Operators* section. This might be just a matter of the limited time of the testing procedure. Intuitively the participants used the correct comparison operator and added m as a unit for a month to the time duration. (MH, TW, PJ)

The definition of time durations needs to be adequately explained. A highlighted legend for explaining possible values at the end of the sheet might be sufficient. However, a better option might be to add more examples and utilize tooltips to give a more in-depth description of the options.

The allowed operations on each Op property type. value can be text, date/time (e.g. 2020-01-01 10:30) (format: 1y2m3w4d, e.g. 1w2d).

```
operator: value      Equa
operator: `word1`
```

I32 Minor: Confusing Full ISO 8601 Time Formatting

The ISO 8601 standard is used for formatting dates and times. Currently, the full time information, including seconds, nanoseconds, and timezone, is displayed (e.g. 2020-10-10 16:20:24.402918 +0000 UTC). The full format was deemed cryptic. (TW)

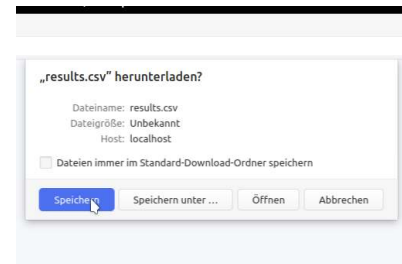
The time format should be changed to something more easily human-readable. Reducing the format to just date, hours, and minutes (2020-10-10 16:20) might be the first step. We might also try to change it to something like Di, 20. Sep 2022. Converting the UTC time to the user's local time might also benefit the user experience. The original full ISO 8601 format could still be displayed as a tooltip when hovering over the time field.

```
filterless mask for personal protection
Helpful Devices
CERN-OHL-S-2.0
opensource, uvc, covid
2021-08-02 16:21:58.346473 +0000...
```


I33 Minor: Unexpected Export File Name

Sometimes, the export file filename is set to search instead of the server configured `results.csv`. (MH)

This issue might be somewhat related to the caching problem. The export function must be checked to set the default export filename consistently.



I34 Minor: Real-World Example Values/Usage

Some examples on the query syntax cheat sheet likely do not reflect real-world usage. For instance, the date/time properties are all presented with an example value of `>1y`, expecting the results to be older than one year. This is most likely not something a user wants to search. (MH, JP)

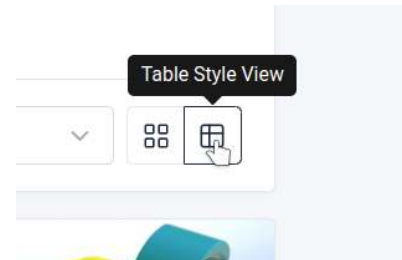
Better examples should be used. For the previously mentioned example, it could be simply to change it to `<1y`, meaning a product is expected to be younger than one year. Other operators in the cheat sheet could also benefit from a more realistic example.

```
createdAt:>1y
lastUpdatedAt:>1y
discoveredAt:>1y
lastIndexedAt:>1y
is:active
is:inactive
```

I35 Minor: Naming Scheme for Views

The naming of the table view was criticized. It was expected to be named "list view" instead, and the symbol on the toggle button should be something that reflects the style better. (TW)

The view name and icon visualization should be changed to "list."



I36 Minor: Alphabetically Sorting of Order Options

The options for ordering in the select box are not presented in an alphabetically ordered fashion. Furthermore, the *State (Activeness)* order option was expected to be named like *Activity State* or similar and thus was not found immediately. (MH, JP, TW)

The options should be ordered, and the *State (Activeness)* be renamed to reflect the user's expectations better.



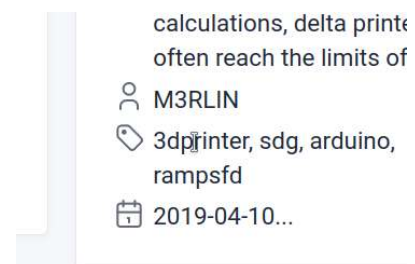
I37 Minor: Confusion About Ascending and Descending Ordering

The options to order products by activeness seem somewhat confusing. Participants expressed that the arrows indicating ascending or descending are slightly misleading. (JP, TW)
Instead of using arrows, we could write out the meaning to indicate ascending or descending ordering.



I38 Minor: Non-Clickable Tags

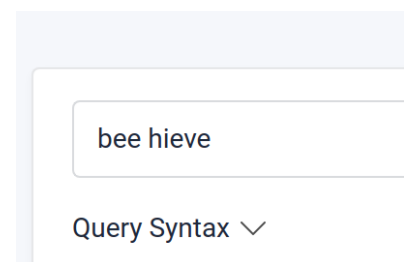
The tags listed on the results page are not clickable as the other elements and thus do not offer additional value. Users expect an overview of products within a tag category. (MH)
When clicking on a tag, a search for products with this particular tag should be performed. This way, the user gets a showcase of all products tagged with the selected tag.



I39 Minor: Hints For Misspelled Words

When faced with the first challenge, some of the participants misspelled *beehive*, either leading to a reduced set of results or no results at all (JP, PJ, TW).

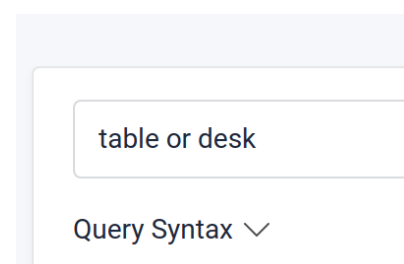
The service could try to offer hints for spelling, synonyms, and other search terms. The user could then change their search query and get more accurate search results.



I40 Major: Uppercase Naming Convention For Logical Operators

It seems unclear that the logical operators (OR/AND/NOT) must be written in upper case to work. The task to fix the invalid query, where the participants had to change them or to OR, proved quite challenging. Through removing terms and running multiple queries, some of the participants managed to find the correct solution. (MH, JP, OS, PJ, TW)

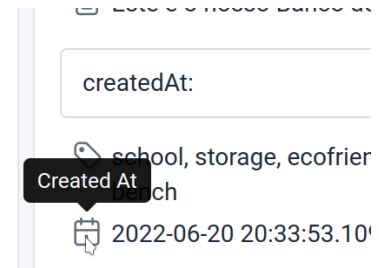
The documentation must be improved to make the naming convention more obvious. Another approach could be to remove the word terms and instead use only their equivalent syntax (|,&,-).



I41 Major: Displaying Additional Properties

The properties defined in the search query will also be displayed in the results view. This sometimes leads to confusion. When the participants have not picked up the connection between the query input and the properties showing up in the results view, they assume unintentional and uncontrollable behavior. When tasked to display additional properties, the participants first resorted to ordering by that property and expected it to show up. (MH, JP, OS, PJ, TW)

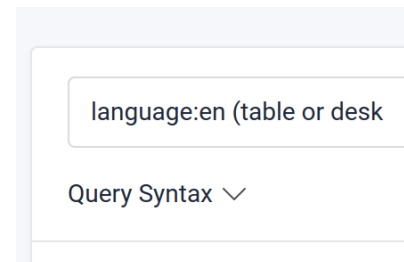
To resolve this issue, the properties should also be displayed when they are selected for ordering. Furthermore, there should be an option to explicitly choose the properties to display.



I42 Major: Hints for Errors

The search is designed to be robust and work in cases of syntax errors. Results will be returned even if the query couldn't be fully parsed. This behavior is similar to Google Search and other search engines. The participants expressed the need for some kind of assistance to help identify errors and potential mistakes (e.g. or instead of OR). (MH, JP, PJ, TW)

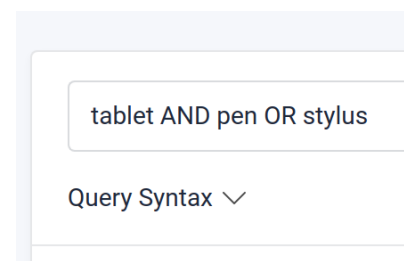
It was suggested multiple times to return results, although potentially wrong, and hint that the query might be faulty. The hint should be placed near the input field to be easily spotted.



I43 Critical: Priority of OR and AND Operators

It is unclear how the logical OR works when combined with AND and other terms. The priority of operations is not documented; thus, the participants struggled to understand that the AND has a higher priority than OR and thus binds more strongly, leading to unexpected search results. It was unclear that the use of grouping through parenthesis was required. (MH, OS, PJ, TW)

As a first quick fix, the syntax cheat sheet should be improved by adding parenthesis directly to the example for the logical OR statement. Furthermore, the documentation and cheat sheet must be improved to make the operator priority more apparent and hint at using parenthesis grouping. Providing examples containing parenthesis should help tremendously.



6.4. Discussion

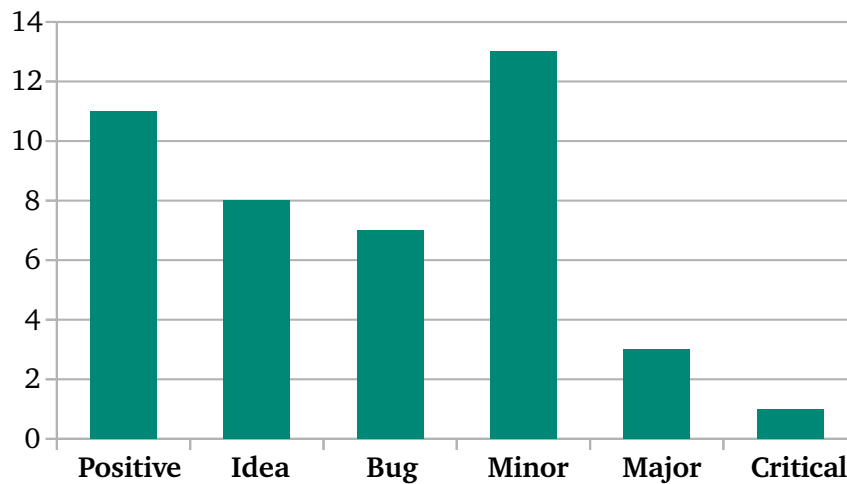


Figure 6.1.: Number of issues by category

Overall, 43 issues have been reported in the usability testing. Of these 43 issues, 11 are positive mentions, 7 problems due to derivations from the designed behavior and 6 major and critical usability findings. Not all of these issues do require a change or improvement; afterall, ideas are suggestions that may or may not be considered and positive findings are used solely for verifying the concepts and usability of the implementation. The positive feedback also helps to identify the general sentiment towards the application, for example, if the participants are inclined or rather reluctant to use the service in the future. The Figure 6.1 summarizes the number of issues per category. As can be seen, most addressable issues are in the *Minor* category. These minor issues resulted in a short moment of confusion and hesitation but are generally not a deal breaker. The application is still usable despite these issues. The amount of positive feedback in relation to major and critical issues demonstrates the overall good impression of the participants.

Several factors must be considered when designing and implementing an application with a good usability factor and a satisfying user experience. Respecting these factors is key in order to offer the users something that they consider to be something of value. Therefore, we reflected on the aspects that make an application usable and satisfying to use, implemented the service, and validated our assumptions with the usability testing. The following points summarize our thoughts and approach:

- **Useful:** First and foremost, the service must fulfill a need. In our case, we created a search engine for OSH that allows the users to search for products across platforms, and thus we improved the general discoverability of products. Our service offers a full-text search and a powerful query syntax that allows users to search and filter by specific properties. Our service provides a more fine-grained and faster search functionality than most hardware hosting platforms, making it convenient and fun for users to find exactly what their are searching for.
- **Usable:** We provide a manageable and appealing web search interface that users can use with their preferred web browser. Thus, the users can use the tools they are already familiar with without needing additional software. The interface only presents the control elements that are required to provide the search functionality. The search bar is most prominent at the top

of the search page, which is the primary input element. It is selected by default, so users can immediately start typing when they visit the page. To not be overwhelmed by the sheer amount of syntax options, the syntax cheat sheet is hidden away but can easily be accessed by the *Query Syntax* button. The control elements for ordering the results and changing the display style are grouped together. They are placed on top of the results and thus are quickly accessible by the users. When the users scroll down and reach the end of the results, they can retrieve more results with the pagination controls. Those let the users control the page to display the number of results to show on each page. The layout of the control elements aligns with the experiences and expectations of the users and thus helps to avoid unpleasant surprises. Another point we like to mention is that the service is also usable on mobile devices. The participants in the usability interview were all able to easily navigate the page and gave positive feedback concerning the usability.

- **Desirable:** Our service was created using the Tabler web UI kit. The kit allows for building responsive, mobile-ready web front-ends with a coherent, modern look and feel. The elements and color scheme resemble the style of regular desktop applications and convey familiarity to the users. The high-quality, appealing interface excited the testing participants and enticed them to try everything themselves. Furthermore, the site offers a dark mode to which users can switch by clicking on the corresponding symbol.
- **Findable:** When users first visit the site, they are greeted with the hero banner on the homepage. The banner explains the purpose of the service and offers a search bar similar to the one on the search page. From there, they can start their search. Another approach is to navigate to the search page directly by clicking on the appropriate menu entry. Users can also access the details pages from the search page to gain more in-depth information about the results. The participants had no trouble finding pages and resources they were looking for.
- **Accessible:** It must be said that the application has not been developed and tested with people with disabilities. People suffering from some form of colorblindness will most likely not encounter any problems using the service. However, no guarantees can be given for users with more severe disabilities.

Previous attempts to create such a search engine have not gained much traction in the community. The original LOSH demonstrator has problems architecture and usability-wise. The offered functionality does not provide significant value. The search seriously lacks advanced features to filter results by properties, and the display of results is plain boring, making it unappealing for most users. Compared to the demonstrator, our service offers far more value.

- **Search Capabilities:** While in generalized web searches, users mostly search content by keywords and full-text, in use cases such as ours, the users most likely require more fine-grained control over what to include in the search. Our advanced query syntax offers an operator-based search that allows users to filter results by various product properties like the used license or the date of the last update. Furthermore, the filters can be combined through logical operators (and/or/not), allowing users to answer complex questions and trim down results. On the other hand, the OPENNEXT LOSH demonstrator only provides a keyword-based search that returns results matching a subset of properties. Users cannot control what properties are searched, making it impossible, for example, to search for products with a specific license. The demonstrator employs three filters for the license strength, repository host, and organization of the products. Only one option for each filter can be selected at a time; thus, users cannot

freely combine these and only filter by one value at a time. Other product properties are simply inaccessible in the search. It is a similar story for the OKH search. It offers a keyword-based search in addition to a simple set of filters for the product source and design file types. OHO, as another contender, offers a keyword-based search as well. It does include far more filters than the previously mentioned services. It lets the users filter by properties such as certification status, if it contains software or has a bill of materials. The strength of the OHO search lies in its category system and partly manually managed dataset. Using the category search allows for easy exploration of related products. Our service includes user-provided tags, but those are often far less descriptive and unbecoming. As such, categorization is a topic that we do have deficiencies and need to improve on. In general, our service offers superior search capabilities and allows for more complex questions to be answered compared to competing implementations.

- **Display of Results:** By default, our web interface offers a pleasant showcase-style overview of the product search results. The results are each displayed with a large image, the product name, description, tags, used license, and the licensor. The users can switch to a table/list-style view, where the results are displayed in rows instead. It is possible to add more properties to be shown if, for example, the users want to compare the date of creation of products. Furthermore, the users can control the order in which the results are displayed, like sorting by name in ascending or descending order. Comparing our service to the OPENNEXT LOSH demonstrator and the OKH search, we see higher flexibility on our side. The demonstrator has a table-style view, displaying the product's name, version, license, repository, and organization. As our user testing indicated, the users deemed the version and repository information irrelevant and would like to see a description instead. So the demonstrator's interface presents meaningless information in a rather plain-looking and boring style. The OKH search does a better job. It offers a similar-looking card-style overview to ours, displaying an image, product name, description, creator name, and tags. Still, it is less flexible and does not provide options such as ordering or other features and displaying different product properties. The OHO search also shows its results in a card-style view. It contains an image, the product name, and information about included files. The interface is very plain, old-looking, unresponsive and thus does not work well on mobile devices. Overall our implementation does offer a more appealing and flexible display of results than the competitors mentioned above.

Considering our achievements in providing a usable and satisfying to use search engine for OSH hardware, we built a service that offers great value to the community, and we think it has a decent chance to be adopted and used regularly in the future.

7. Conclusion and Future Work

7.1. Conclusion

The concept of open-sourcing hardware designs, implementation, and documentation is believed to be a driver of future innovations and technological advancements, as open source in the software sector has proven before. However, OSH is less prevalent than acOSS and still not widely accepted by industries. To that end, the first step for widespread adoption is to improve the discoverability of hardware designs and documentation.

In this thesis, we designed, implemented, and tested a semantic knowledge base and search engine for OSH. The goal was to create a service that is easy and satisfying to use, improves the discoverability of open hardware, and presents value to the OSH community. We interviewed members of the community (Section 4.2) to evaluate the already existing OPENNEXT LOSH demonstrator (Section 3.4) and assess the requirements for our system. Some of the reported needs and wishes were quite unexpected; many of these are not even offered by any other service today. After the requirements analysis had been concluded, we designed and implemented a service with an entirely new architecture and components. The created service offers an advanced query syntax that allows users to search hardware products by various properties and answer complex questions. The search is available as a web service that can be used with any browser and device, including mobile devices. The interface was created with a strong focus on usability and user experience to provide appealing and easy-to-use software.

We conducted a usability testing to evaluate our implementation and validate our assumptions regarding usability and user experience. The testing showed that our service left a generally good impression on the participants. The overall positive feedback in relation to the number and severity of reported problems signaled us that we created a usable system with a high degree of user satisfaction. The advanced search functionality stood out above, allowing users to answer complex questions regarding open hardware products. The search offers more options for fine-grained filtering than other search engines in the same field can offer. In addition, the designed web user interface provides a more detailed and inviting overview of product information than the competing services. Furthermore, our service includes Semantic Web (SW) technologies to allow humans and machines alike to access the dataset and enable further research on that topic. But the user testing also revealed some issues. Most of the usability-related problems were minor issues and could be resolved easily. Only a few issues were in the major and critical category. We conclude our service presents a valuable asset and has a good chance of adoption by the OSH community.

7.2. Future Work

It will take more effort to make the service ready for production use and a valuable addition to the digital toolset of the OSH community. First and foremost, the issues raised in the usability testing (Section 6.3) need to be addressed. Hereinafter, more platforms need to be added to the search index, and the amount and quality of the collected metadata be improved.

Usability testing with a selected group of people from the OSH community was the first step to creating a usable application and offering a satisfying user experience. However, the community's involvement must be strengthened further to increase the chances of adoption. It is vital to regard the expectations and needs of the community. A key aspect is involvement through collaborative open source development, acceptance of bug reports and feature proposals, and other means. Building an environment that empowers community involvement is something to be done as part of future efforts. We are already in contact with representatives of the OPENNEXT research project to discuss the continuation of the work under their umbrella.

Product categories and tags help tremendously to find desired products and alternative solutions. Currently, tags are collected alongside the product information, if the hardware hosting platforms offer any. The number of tags is relatively sparse, and there is no categorization yet. The OHO collected hardware designs from various sources and categorized the results in ~500 categories. The categorization was performed partly automatically and partly manually. For the LOSH search engine, a manual categorization seems impractical because of the sheer amount of results. A fully automated categorization and tagging system would be a useful addition to our service. It could be implemented with a machine learning model that uses the product metadata to derive an appropriate category and/or tags.

Information such as technology or documentation readiness is usually determined by assessing the product and its documentation. The metrics are relatively new, and no hardware platform includes them in their data model. Thus, there is currently no data available for these kinds of metrics. Considering how useful this information could be to guide the decision process for the commercial use of a product, it would be beneficial to determine these metrics automatically. Such an automatic product assessment requires research and is something to be done as part of future work.

As described, Semantic Web (SW) technologies can help to span a net of data by connecting information across the boundaries of datasets. A sufficiently dense net of information can be of immense value. In the field of OSH, this can reveal hidden truths, improve discoverability, encourage reuse and lead to quicker technological advancements. On that note, our implementation currently lacks an RDF schema to properly describe the vocabulary we used. This needs to be addressed in the future. Furthermore, we need to consider how the LOSH specification fits into this equation if the offered RDF should reflect the structure used by the service or rather should be leaned on the LOSH specification. Another open research topic is the cross-linking of products, users, and other data entities. This needs to be addressed in future work to tap into the true potential of the SW paradigm.

A. Requirements Analysis Interview Questions

A.1. Introduction

What is the Library of Open Source Hardware (LOSH) about?

- Purpose:
 - Tackling the lack of discoverability across several Open Source Hardware (OSH) content-hosting platforms
 - Providing a public semantic knowledge base and search engine for OSH
 - Improving the Open Source Hardware (OSH) toolset, helping the growth of the open-hardware community and helping OSH gain wider acceptance amongst industries
- How:
 - Providing a specification for OSH product metadata
 - Collecting metadata and technical documentation of OSH from various platform
 - Checking compliance with the specification (guarantee some level of quality)
 - Providing a web service for searching and exploring OSH
- Demonstrator was developed, but due to the lack of features and usability it was not marketed as ready to use

What is this first interview for?

- Identifying user requirements/wishes for the new LOSH service
- Involving the community and end users directly in the process of creating the service

What to expect from the interview?

- Bunch of qualitative questions, that I like you to answer (it is ok, if you have no answer for some questions)
- Challenges using the current available LOSH demonstrator and other web services

A.2. Interview

Facts

- Name:
- Contact Information:
- Maker/Designer/Other?:
- Professional/Hobby:
- Experience with OSH? (Usage, Development, Documentation, Issue Reporting, ...):
- What kind of hardware do you work with?:
- Experience with OSS? (Usage, Development, Documentation, Issue Reporting, ...):
- What kind of software do you usually work with?:
- Consider yourself a versed computer user?:

What OSH Platforms have you heard of and used before?

What did you use the OSH platform for?

Did you notice major differences?

Challenges

- Set of tasks and questions
- Not necessarily solvable with the current demonstrator
- To find out:
 - How you would approach this challenge?
 - How you feel about the current implementation?
 - What functionality would be required to solve the challenges?
- Disclaimer: I created large parts of the crawler component, which gathers the OSH information. I was not involved in the storage or management of the data, also I am not responsible for the current web interface and its functionality.

Challenge: Find a product that has something to do with "labyrinth" and is hosted on the Wikifactory host and get to the source web page of the product

Challenge: Find a product that has something to do with "drawing" and has a Strong Copyleft license and export the search results as CSV

What is your first impression on the ability to filter search results?

Do you think the filter ability needs to be improved and how? (multiple option, and/or/not combinations)

Should more fields be considered for searching/filtering? (https://github.com/OPEN-NEXT/OKH-LOSH/blob/master/sample_data/okh-TEMPLATE.toml)

What importance does exporting results play in your opinion?

Challenge: A ventilator consists of 4 submodules, one of them is a motor. It turns out that a specific version of this motor is faulty. Which ventilator versions are affected?

What importance does versioning play and should the service index all available versions and make them searchable?

When a product gets deleted on the host platform, should the index also be deleted? What becomes of the projects, that link to that?

Challenge: There exists a semantic web page for every resource, which lists all the properties of the resource and can be used to discover other relationships. Find the associated semantic web page for the product "OHLROOM" (<https://losh.ose-germany.de/>)

What do you expect of an overview page of a product?

Would you prefer a semantic resource page like that?

How would you like the search results to be displayed like? List? Cards? Other?

- Table Style: <https://preview.tabler.io/tables.html>
- List/Card Style: <https://preview.tabler.io/lists.html>
- Semantic Overview <https://losh.ose-germany.de/wiki/Item:Q8684>

How do you feel about the current implementation? What are the good and bad parts?

What would you use the new LOSH service for, and what do you think other users would use it for?

Would companies use it differently than individual users?

For usability/functionality/look of the new LOSH service, do you have any examples of search engines or general web interfaces that could serve as an inspiration? And why? (similar functionality/interface/behavior)

Anything else you want to say?

Would you be willing to participate in another interview for evaluating the new LOSH service?

B. Usability Testing Tasks

Stelle dir vor du möchtest Bienen züchten und möchtest daher ein Bienenstock (beehive) bauen. Möglicherweise gibt es freie Baupläne, die du nutzen kannst. Gehe auf die Webseite, finde ein passendes Projekt zum Nachbauen und besorge dir den Bauplan dazu.

- What was easy or difficult about solving that task?
- What is your first impression?

Purpose:

- Introduction to the web interface, what the elements are and how to get to the product repository from the search results
- Simple full-text search

Solution:

1. Use simple full-text "beehive"
2. From the search results, go to the products web page to get the build instructions

Du hast auf deiner Suche nach "3D print" einige Resultate erhalten. Nun bist du eher an aktiven entwickelten Projekten interessiert und möchtest daher aktive Projekte in der Ansicht zuerst anzeigen lassen. Führe das beschriebene Szenario durch.

- What was easy or difficult about solving that task?

Purpose:

- Get to know the ordering capability and how to use it

Solution:

1. Search for 3D print
2. Use the "Order By" select box to select "State (Activeness) ▾"

Du hast einige Suchergebnisse für die Suche nach "3D print" erhalten. Nun bist du interessiert an der Popularität der Produkte. Verschaffe dir eine schnelle Übersicht ("auf einen Blick") über die Popularität der gefundenen Produkte.

- What was easy or difficult about solving that task?

Purpose:

- Engage with the different styles of presentation
- Use the table view for an easy overview

Solution:

1. Search for 3D print
2. Switch to table view
3. Sort by "Start Count"

Es gibt die Möglichkeit Produkte nach ihren verschiedenen Eigenschaften zu suchen, wie z.B. nach der verwendeten Lizenz oder welche Sprache für die Dokumentation verwendet wird. Führe eine Suchanfragen für folgende Fragestellungen durch:

Purpose:

- Engage with operators
- One simple search query involving full-text and one or two operators
- Complex query combining and grouping operators

Produkte die etwas mit *Solar* zu tun haben und in den *letzten 6 Monaten ein Update erhalten*

Solution:

- `solar lastUpdatedAt:<6m`

Einen *Table* oder *Desk* das ein *Bild* hat und in *Englisch dokumentiert* ist.

Solution:

- `(table | desk) has:image language:en`

Du hast folgende Suchanfragen ausgeführt, aber scheinbar erhältst du nicht die Ergebnisse, die du erhofft hast. Versuche herauszufinden was das Problem zu sein scheint und behebe es.

Purpose:

- Use documentation about syntax to understand and fix the syntax

Produkte, die etwas mit print zu tun haben, aktiv und in Englisch dokumentiert sind und zusätzlich noch eine CC-BY-XXX Lizenz haben.

```
print is:active language:en (license:"CC-BY-SA-1.0" OR license:"CC-BY-SA-2.0" OR license:"CC-BY-SA-2.0-UK" OR license:"CC-BY-SA-2.1-JP" OR license:"CC-BY-SA-2.5" OR license:"CC-BY-SA-3.0" OR license:"CC-BY-SA-3.0-AT" OR license:"CC-BY-SA-3.0-DE" OR license:"CC-BY-SA-4.0" OR "license:CC-BY-1.0" OR "license:CC-BY-2.0" OR "license:CC-BY-2.5" OR "license:CC-BY-2.5-AU" OR "license:CC-BY-3.0" OR "license:CC-BY-3.0-AT" OR "license:CC-BY-3.0-DE" OR "license:CC-BY-4.0")
```

Solution:

- `print is:active language:en (license:"CC-BY-SA-*")`

Ein Produkt, das irgendetwas mit "Motor" zu tun hat, lizenziert ist und eine gewisse Popularität hat.

```
motor has:license (starCount:>0 or forkCount:>0
```

- As Google and other search engines, the search will silently ignore bad syntax and treat most of it as full-text search instead. This might lead to undesired results. In your opinion should the user rather be confronted with an error or return some results, even if those won't be as expected?

Solution:

- `motor has:license AND (starCount:>0 OR forkCount:>0)`

Du möchtest eine Auswertung darüber machen, welche Produkte von einer CC-BY-SA Lizenz Gebrauch machen. Suche nach entsprechenden Produkten und speichere die Ergebnisse in einer Excel-Tabelle ab.

- What was easy or difficult about solving that task?

Purpose:

- Make use of Export functionality

Solution:

1. Define query and execute query
2. Use the "Export Results" button to receive results as CSV and open it in Excel/Libre Office/other

Du hast einige Suchergebnisse für die Suche nach "3D print" erhalten. Du möchtest dir einen Überblick verschaffen wann die Produkte erzeugt wurden. Wie gehst du vor?

- What was easy or difficult about solving that task?
- Is this method good enough, or do you prefer a different style of selecting result fields?

Purpose:

- Is it obvious enough to use operators to add more displayed properties?

Solution:

1. Search for 3D print createdAt: (the empty operator will be ignored in the search, but the field will be listed in the result overview)
2. Switch to table view

C. Profiles of the Interviewees

All interviewees do have some experience and background in working with OSH and/or OSS. This section contains the individual profiles of these interviewees. The profiles reveal the OSH affiliation and technical background.

Interviewee: FR

- **OSH Affiliation:** hobby maker
- **Experience with OSH DIY making**
- **Usually Working With:** wood
- **Known OSH Platforms:** OHO, YouTube
- **OSH Platforms Used Primarily For:**
 - getting inspirations
- **Versed Computer User:** yes
- **Experience with OSS:** usage, development, documentation, issue reporting
- **Software Used (in Context of Working With OSH):** Linux, LibreOffice, Inkscape, FreeCAD
- **What Could the LOSH Search Engine Be Used For:**
 - getting inspiration
 - evaluation of OSH (research, political)

Interviewee: JP

- **OSH Affiliation:** hobby maker
- **Experience with OSH DIY making, usage, documentation**
- **Usually Working With:** RaspberryPi + sensors and such
- **Known OSH Platforms:** GitHub, GitLab, Thingiverse, Instructables, Hackster
- **OSH Platforms Used Primarily For:**
 - search/explore hardware/software
 - categorize projects

-
- **Versed Computer User:** yes
 - **Experience with OSS:** usage, development, documentation, issue reporting
 - **Software Used (in Context of Working With OSH):** Linux, Vim, VSCode, RaspberryPI (libs), Python, GitLab, Draw.io
 - **What Could the LOSH Search Engine Be Used For:**
 - search/exploring products
 - find products that are based on standard X or do have license Y (mostly companies and professionals)
 - find inspiration on how to do X
 - find products where one can contribute to
 - research what are the current trends in OSH

Interviewee: MH

- **OSH Affiliation:** professional project manager, scientist, hobby maker
- **Experience with OSH** usage, development, documentation, issue reporting
- **Known OSH Platforms:** Wikifactory, GitHub, GitLab, Thingiverse, OSHWA, OHO, Hackaday, DocuBricks, Wikifab, OSE(G) Wiki, Appropedia and more
- **OSH Platforms Used Primarily For:**
 - search hardware
 - research related stuff
 - create documentation
- **Versed Computer User:** yes
- **Experience with OSS:** usage, issue reporting, some documentation
- **Software Used (in Context of Working With OSH):** OpenSCAD, Scilab, Geogebra
- **What Could the LOSH Search Engine Be Used For:**
 - searching for functionality and thereby find technologies
 - research what other products in the same field integrate parts wise
 - research what licenses do the other products use

Interviewee: NW

- **OSH Affiliation:** professional project management, hobby OSH user, researcher, developer of standards
- **Experience with OSH** documentation/standardization, project management
- **Usually Working With:**
 - Bee Wax Tissues
 - Zink-Luft-Brennstoffzelle
 - Webstuhl
 - Tandemfahrrad
 - Projects with a lot of software components
- **Known OSH Platforms:** Wikifactory, GitHub, GitLab, GrabCAD, GOSH, OSE(G) Wiki
- **OSH Platforms Used Primarily For:**
 - search for projects
 - documentation of projects
 - project management
 - gather contacts for help to review stuff
- **Versed Computer User:** yes
- **Experience with OSS:** usage
- **Software Used (in Context of Working With OSH):** LibreOffice, Gimp, OpenLCA (life cycle assessment), OpenCAD, Etherpad, Hedgedoc, Zettlr, BBB (interviews), GitLab (git), Linux
- **What Could the LOSH Search Engine Be Used For:**
 - get a quick overview of products with standardization
 - for research questions
 - high-level function, users will most likely use the platforms they are familiar with and have their community

Interviewee: OS

- **OSH Affiliation:** professional (Freelancer) / hobby OSH developer
- **Experience with OSH** usage, development, documentation, issue reporting
- **Usually Working With:**
 - UniPro-Kit
 - Zink-Luft-Zelle
 - 3D Druck
 - Fräse
 - Nanogrid
 - LibreSolar
- **Known OSH Platforms:** OSE(G) Wiki, OHO, Wikifactory, OpenNext, Thingiverse, YouMagine, Instructables, YouTube
- **OSH Platforms Used Primarily For:**
 - hosting projects
 - searching for solutions
 - looking for inspiration/advice
 - getting in contact with other developers/makers
- **Versed Computer User:** very versed
- **Experience with OSS:** usage, development, documentation, issue reporting
- **Software Used (in Context of Working With OSH):** CAX, FreeCAD, KeyCAD, Arduino, IDE for software development
- **What Could the LOSH Search Engine Be Used For:**
 - with only knowledge about a project name find more information such as project site
 - find similar projects
 - find components for reuse
 - to make own projects more visible

Interviewee: PJ

- **OSH Affiliation:** hobby maker
- **Experience with OSH** presentation/documentation of projects
- **Usually Working With:**
 - optics
 - open agricultrue
- **Known OSH Platforms:** OSHWA, GitHub, OHO
- **OSH Platforms Used Primarily For:**
 - search for projects
 - research
- **Versed Computer User:** no
- **Experience with OSS:** almost none
- **Software Used (in Context of Working With OSH):** Latex
- **What Could the LOSH Search Engine Be Used For:**
 - technology trend analysis (like evaluating patent database)

Interviewee: TW

- **OSH Affiliation:** hobby maker/designer, professional project manager/coordinator
- **Experience with OSH** usage, development, documentation, issue reporting, coordination
- **Usually Working With:**
 - mechanical eng.
 - renewable energy
 - 3D print
- **Known OSH Platforms:** Wikifactory, GitHub, GitLab, Thingiverse, OHO, Wikifab, OSE(G) Wiki, Printables, Envienta
- **Versed Computer User:** yes
- **Experience with OSS:** more user site, contributor (issues, ideas)
- **Software Used (in Context of Working With OSH):** FreeCAD, LibreOffice, Hedgedoc, GitLab
- **What Could the LOSH Search Engine Be Used For:**
 - end users: search for solutions to use or reproduce
 - product developers: market analysis, looking for existing solutions
 - researcher: what is the current state of OSH projects; what projects are there

D. Query Syntax Definition

General Expressions

Term	Example	Description
word1 word2 `word1 word2`	nikola tesla	Full Text Search - Text must contain all search words in an arbitrary order
"" ''	"nikola tesla"	Exact Text Search - A word or phrase inside single or double quotes are treated as input for an exact text search. A text must contain the given text (case-insensitive) to produce a match
AND &	tablet AND pencil	Logical AND - Combining a pair of two terms with AND yields results containing both terms. This is the default and doesn't have to be explicitly specified
OR 	motor OR turbine	Logical OR - Combining a pair of two terms with OR yields results containing either one or both of those terms
NOT -	NOT tablet -tablet	Logical NOT - A term preceding NOT or - will be negated and thus explicitly excluded from the results
()	(motor OR turbine) AND engine	Grouping Terms - Terms wrapped inside parenthesis () will be treated as a group and searched together
*	CC-BY-*	Wildcard Text Search - An asterisk * acts as a wildcard, meaning any phrase or word in between will be matched
operator:expr	license:MIT	Operator Term - The associated properties must match the provided expression.

Table D.1.: Query Syntax - General Expressions

Text Operator Type

Properties of type *Text* usually combine Full-Text Search, Term Search, and/or Exact Text Search.

Expression	Description
operator:word operator: `word1 ... wordN`	Full Text Search: Text must contain all search words in an arbitrary order
operator:"word1 ... wordN" operator:'word1 ... wordN'	Exact Text Search: Text must contain the exact search string. Matching is performed case-insensitive
operator:w*d operator:"word1 * wordN"	Wildcard Text Search: Text must contain the search string. * acts as a wildcard, meaning any phrase or word in between will be matched
operator:=="word1 ... wordN"	Exact Text Search: Text must match the whole search string
operator:!="word1 ... wordN"	Exact Text Search: Text must not match the whole search string

Table D.2.: Query Syntax - *Text* Operator Type

Boolean Operator Type

Properties of type *Boolean* are different from other types. The following expressions are defined:

Expression	Description
is:property	Boolean property must be true
has:property	Property must be set (non-nil)

Table D.3.: Query Syntax - *Boolean* Operator Type

Number Operator Type

Properties of type *Number* can be compared in multiple ways:

Expression	Description
operator:42	Number property must match the given value
operator:==42	Number property must match the given value
operator:!=42	Number property must not match the given value
operator:<42	Number property must be less than the given value

Table D.4.: Query Syntax - *Number* Operator Type

Expression	Description
operator:<=42	Number property must be less or equal to the given value
operator:>42	Number property must be greater than the given value
operator:>=42	Number property must be greater or equal to the given value
operator:20..42	Number property must be between the given values (inclusive)

Table D.4.: Query Syntax - *Number* Operator Type

DateTime Operator Type

Properties of type *DateTime* can be compared like numbers. The format for specifying date and time can either be ISO 8601 or a time duration in the form of 1y2m3w4d (equals one year, two months, three weeks, four days).

Expression	Description
operator:2022-09-30 operator:6m	DateTime property must match the given date and time
operator:==2022-09-30 operator:==6m	DateTime property must match the given date and time
operator:!=2022-09-30 operator:!=6m	DateTime property must not match the given date and time
operator:<2022-09-30 operator:<6m	DateTime property must be less than the given date and time
operator:<=2022-09-30 operator:<=6m	DateTime property must be less or equal to the given date and time
operator:>2022-09-30 operator:>6m	DateTime property must be greater than the given date and time
operator:>=2022-09-30 operator:>=6m	DateTime property must be greater or equal to the given date and time
operator:2022-04-01..2022-09-30 operator:6m..1y	DateTime property must be between the given values (inclusive)

Table D.5.: Query Syntax - *DateTime* Operator Type

Basic Operators

Name/Example	Type	Description
name:beehive	Text	Product Name
description:beehive	Text	Product Description
language:language	Text	Alias for documentationLanguage
documentationLanguage:language	Text	Language used for documentation
version:1.0.0	Text	Version of latest release
website:"https://github.com"	Text	Product Website
starCount:>0	Number	Number of stars
forkCount:>0	Number	Number of forks
releaseCount:>0	Text	Number of releases
createdAt:<6m	DateTime	Date/Time of when the latest release was created
lastUpdatedAt:<6m	DateTime	Date/Time of when the latest release was created
discoveredAt:<6m	DateTime	Date/Time of when it was last updated
lastIndexedAt:<6m	DateTime	Date/Time of when it was indexed
is:active is:inactive is:archived is:deprecated is:missing	Boolean	Activeness of the product.

Table D.6.: Query Syntax - Basic Operators

Categorization

Name/Example	Type	Description
has:category	Boolean	Indicates whether it has a category assigned
category:Robotics	Text	Alias for categoryfullname
categoryFullName:Robotics	Text	Full category name (e.g Computer/PSU)
categoryName:Robotics	Text	Alias for documentationLanguage
has:tags	Boolean	Indicates whether it has tags assigned
tag:3dprinting	Text	Tag name

Table D.7.: Query Syntax - Categorization

Name/Example	Type	Description
tagCount:>0	Number	Number of tags

Table D.7.: Query Syntax - Categorization

Repository

Name/Example	Type	Description
host:Wikifactory	Boolean	Alias for repository
repository:Wikifactory	Text	Alias for repositoryHost
repositoryHost:Wikifactory	Text	Host where the product is developed
repositoryOwner:John	Text	Full name of the repository owner
repositoryName:	Text	Name of repository
datasource	Text	Alias for datasourceHost
datasourceHost	Text	Host where the product was found
datasourceOwner	Text	Full name of the data-source owner
datasourceName	Text	Name of the data-source repository

Table D.8.: Query Syntax - Repository

License

Name/Example	Type	Description
has:license	Boolean	Indicates whether Product is licensed
has:hasAdditionalLicenses	Boolean	Indicates whether Product has other licenses
license:CC-BY-SA-4.0	Text	Alias for licenseId
licenseId:CC-BY-SA-4.0	Text	License SPDX Identifier
licenseName:value	Text	License Full Name
is:licenseSpdx	Boolean	License is well-known and listed on SPDX.org
is:licenseDeprecated	Boolean	License is marked as deprecated
is:licenseOsiApproved	Boolean	License is approved by OSI
is:licenseFsfLibre	Boolean	License is FSF approved
is:licenseBlocked	Boolean	License is not approved by LOSH
is:licenseStrong	Boolean	License is considered strong

Table D.9.: Query Syntax - License

Name/Example	Type	Description
is:licenseWeak	Boolean	License is considered weak
is:licensePermissive	Boolean	License is considered permissive

Table D.9.: Query Syntax - License

Licensors

Name/Example	Type	Description
licensor:John	Text	Alias for licensorFullName
licensorFullName:John	Text	Licensors Full Name
licensorName:jhondoe42	Text	Licensors Username
is:licensorUser	Boolean	Licensors is a person
is:licensorGroup	Boolean	Licensors is a group (organization)

Table D.10.: Query Syntax - Licensors

Files

Name/Example	Type	Description
has:software	Boolean	Indicates whether it contains software
has:image	Boolean	Indicates whether it has an image
has:readme	Boolean	Indicates whether it has a readme file
has:contributionGuide	Boolean	Indicates whether it has a contribution guide
has:bom	Boolean	Indicates whether it has bill of materials
has:manufacturingInstructions	Boolean	Indicates whether it has manufacturing instructions
has:userManual	Boolean	Indicates whether it has a user manual
has:source	Boolean	Indicates whether it has a source file
has:export	Boolean	Indicates whether it has export files
has:auxiliary	Boolean	Indicates whether it has auxiliary files

Table D.11.: Query Syntax - Files

Standard, Publication, Maturity, etc.

Name/Example	Type	Description
has:attestation	Boolean	Indicates whether it was attested
has:publication	Boolean	Indicates whether it has a publication
has:issueTracker	Boolean	Indicates whether it has a dedicated issue tracker
has:complieswith	Boolean	Indicates whether it has a standard associated

Table D.12.: Query Syntax - Standard, Publication, Maturity, etc.

Name/Example	Type	Description
compliesWith:"DIN SPEC 3105"	Text	Complies with a standard (DIN, ISO, etc.)
has:cpcPatentClass	Boolean	Indicates whether it has a CPC patent class associated
cpcPatentClass:A01B	Text	Cooperative Patent Classification (CPC)
has:tsdc	Boolean	Indicates whether it has a tsdc associated
tsdc:MEC	Text	Technology-specific Documentation Criteria (TSDC)

Table D.12.: Query Syntax - Standard, Publication, Maturity, etc.

Bibliography

- [1] OPENNEXT. “OPEN!NEXT – Transforming Collaborative Product Creation.” (), [Online]. Available: <https://opennext.eu/> (visited on 12/03/2021).
- [2] Paul Cormier. “The State of Enterprise Open Source: A Red Hat report.” (), [Online]. Available: <https://www.redhat.com/en/resources/state-of-enterprise-open-source-report-2022> (visited on 09/20/2022).
- [3] C. a. T. (C. Directorate-General for Communications Networks, K. Blind, S. Pätsch, *et al.*, *The Impact of Open Source Software and Hardware on Technological Independence, Competitiveness and Innovation in the EU Economy: Final Study Report*. LU: Publications Office of the European Union, 2021, ISBN: 978-92-76-30980-2. [Online]. Available: <https://data.europa.eu/doi/10.2759/430161> (visited on 09/21/2022).
- [4] U.S. Department of Energy. “Renewable Energy,” Energy.gov. (), [Online]. Available: <https://www.energy.gov/eere/renewable-energy> (visited on 09/20/2022).
- [5] Internet of Production Alliance, “Open Know-How Specification,” *Internet of Production Alliance*, Jan. 26, 2022. [Online]. Available: <https://standards.internetofproduction.org/pub/okh/release/1> (visited on 09/05/2022).
- [6] OPENNEXT, OKH-LOSH, OPEN-NEXT, Nov. 25, 2021. [Online]. Available: <https://github.com/OPEN-NEXT/OKH-LOSH> (visited on 11/29/2021).
- [7] Martin Häuer, Erik Konietzko, Cansu Tanrikulu, Pen-Yuan Hsing, and Max Kampik, *Deliverable 3.5 - Finalised demonstrators - usability-tested, verified and validated demonstrators*. [Online]. Available: https://github.com/OPEN-NEXT/D3.5-Report/files/9584935/D3.5_Validation.of.demonstrators.pdf (visited on 09/17/2022).
- [8] World Wide Web Consortium. “Semantic Web.” (), [Online]. Available: <https://www.w3.org/2001/sw/> (visited on 02/14/2022).
- [9] OSHWA. “About,” Open Source Hardware Association. (Apr. 7, 2012), [Online]. Available: <https://www.oshwa.org/about/> (visited on 09/01/2022).
- [10] OSHWA. “Definition of Open Source Hardware,” Open Source Hardware Association. (May 26, 2012), [Online]. Available: <https://www.oshwa.org/definition/> (visited on 09/01/2022).
- [11] J. Bonvoisin, J. Molloy, M. Haeuer, and T. Wenzel, “Standardisation of practices in Open Source Hardware,” *Journal of Open Hardware*, vol. 4, no. 1, p. 2, Aug. 19, 2020, Comment: 9 Pages without abstract and references (else 13), no figures, ISSN: 2514-1708. DOI: 10.5334/joh.22.arXiv: 2004.07143. [Online]. Available: <http://arxiv.org/abs/2004.07143> (visited on 02/20/2022).
- [12] “Semantic Web - W3C.” (), [Online]. Available: <https://www.w3.org/standards/semanticweb/> (visited on 02/14/2022).

-
- [13] Tim Berners-Lee, *Linked Data - Design Issues*, Jun. 18, 2009. [Online]. Available: <https://www.w3.org/DesignIssues/LinkedData.html> (visited on 02/14/2022).
- [14] World Wide Web Consortium. "Linked Data." (), [Online]. Available: <https://www.w3.org/standards/semanticweb/data> (visited on 09/08/2022).
- [15] M. K. Tarakeswar and D. Kavitha, "Search engines: A study," *Journal of Computer Applications (JCA)*, vol. 4, no. 1, pp. 29–33, 2011.
- [16] "About the OHO Search Engine - OHO - search engine for sustainable open hardware projects." (), [Online]. Available: https://en.oho.wiki/wiki/About_the_OHO_Search_Engine (visited on 01/14/2022).
- [17] "About OHO Open Hardware Observatory - OHO - search engine for sustainable open hardware projects." (), [Online]. Available: https://en.oho.wiki/wiki/About_OHO_Open_Hardware_Observatory (visited on 01/15/2022).
- [18] "The Open Know-How Manifest Specification Version 1.0," Barbal. (), [Online]. Available: <https://barbal.co/the-open-know-how-manifest-specification-version-1-0/> (visited on 11/29/2021).
- [19] Internet of Production Alliance. "About us." (), [Online]. Available: <https://www.internetofproduction.org/about-us> (visited on 09/05/2022).
- [20] Internet of Production Alliance. "Open Know-How." (), [Online]. Available: <https://www.internetofproduction.org/open-know-how> (visited on 09/05/2022).
- [21] Sonika Gogineni, Martin Häuer, Elena Aleynikova, and Conny Kawohl, *Wikibase instance demonstrator - Semantic database for open linked knowledge on OSH products and services*, Jan. 10, 2022. [Online]. Available: https://github.com/OPEN-NEXT/D3.3-Report/blob/502d5dd5595acb4cf125d63e20dcf0dfb7ababcd/D3.3_%20Wikibase%20instance%20demonstrator.odt (visited on 04/08/2022).
- [22] "Wikibase - MediaWiki." (), [Online]. Available: <https://www.mediawiki.org/wiki/Wikibase> (visited on 02/15/2022).
- [23] Dgraph Labs Inc. "Dgraph - Products," Dgraph | GraphQL Cloud Platform. (), [Online]. Available: <https://dgraph.io/products/> (visited on 09/13/2022).
- [24] Cloudflare. "What is a web crawler? | How web spiders work," Cloudflare. (), [Online]. Available: <https://www.cloudflare.com/learning/bots/what-is-a-web-crawler/> (visited on 09/08/2022).
- [25] Yeong Su Lee. "Web Crawling." (), [Online]. Available: https://www.cis.uni-muenchen.de/~yeong/Kurse/ss09/WebDataMining/kap8_rev.pdf (visited on 05/10/2022).
- [26] GitHub Inc. "GitHub About," GitHub. (), [Online]. Available: <https://github.com> (visited on 09/23/2022).
- [27] International Organization for Standardization, *Ergonomics of human-system interaction — Part 210: Human-centred design for interactive systems*, 2010.
- [28] M. Hertzum. "Usability testing : A practitioner's guide to evaluating the user experience." (2020), [Online]. Available: <https://doi.org/10.2200/S00987ED1V01Y202001HCI045>.