

Project Report

Author-formatted document posted on 12/10/2023

Published in a RIO article collection by decision of the collection editors.

DOI: <https://doi.org/10.3897/arphapreprints.e114048>

Establishment of a data visualization interface for the Digital Botanical Gardens Initiative

 Maëlle Wannier

UNIVERSITY OF FRIBOURG

MASTER THESIS

**Establishment of a data visualization
interface for the Digital Botanical Gardens
Initiative**

Author:

Maëlle WANNIER

Supervisor:

Pierre-Marie ALLARD

*A thesis submitted in fulfillment of the requirements
for the degree of MSc in Computational Biology and Bioinformatics*

in the

COMMONS Lab
Department of Biology

October 9, 2023

UNIVERSITY OF FRIBOURG

Abstract

Faculty of Sciences and Medicine
Department of Biology

MSc in Computational Biology and Bioinformatics

Establishment of a data visualization interface for the Digital Botanical Gardens Initiative

by Maëlle WANNIER

The Digital Botanical Gardens Initiative (DBGI) embarks on an innovative journey to curate, manage, and disseminate digital data from living botanical collections, with an emphasis on mass spectrometric evaluations of chemodiversity. Using semantic web technology, this data is linked with relevant metadata, propelling ecosystem research and guiding biodiversity conservation efforts. Central to the success of DBGI is the creation of an interactive platform for both humans and machines to assimilate this knowledge. This report outlines our efforts to design the prototype of a data visualization portal intended to evolve into the DBGI dashboard. Starting with a Plotly Dash application, the project transitioned to a Node.js application leveraging Javascript, HTML, and CSS for enhanced customization. This provides a basis for future improvements, some of which are proposed in the report.

Contents

Abstract	i
Acknowledgements	v
1 Introduction	1
1.1 Background and motivation for the project	1
1.2 Brief overview of the research problem and objectives	2
2 Literature Review	4
2.1 Discussion of existing research on creating data dashboards and using SPARQL queries for data visualization	4
2.1.1 GNPS dashboard	4
2.1.2 LOTUS resources	5
Wikidata	5
Natural Products Online	5
Nprod.net	6
3 Methodology	8
3.1 Description of the data sources and how they were collected and processed	8
3.2 Discussion of the software tools and programming languages used to build the dashboard and interface with the knowledge graph and LCMS profiles	9
4 Results	13
4.1 Presentation of the dashboard interface and its various features and components	13
4.1.1 Home (B.1)	13
4.1.2 Explore whole dataset (B.2)	13
4.1.3 Explore Molecules (B.8)	16
4.1.4 Molecule Page (B.9)	16
4.1.5 Explore Organism (B.10)	16
4.1.6 Organism page (B.11)	17
4.1.7 Download (B.12)	17
4.1.8 Informations	17
4.2 Perspectives	17
4.2.1 Error handling	17
4.2.2 Visualisation options	18
4.2.3 SQL search	18
4.2.4 Sample page	18
4.2.5 Download page	19
4.2.6 Functionalities for DBGI data	19
4.2.7 Cross-Browser and Device Compatibility Testing	19
4.3 Examples of SPARQL queries	19

5 Discussion & Conclusion	21
5.1 Discussion of how the code was optimized for efficiency and speed.....	21
5.1.1 Javascript optimization.....	21
5.1.2 HTML optimization.....	23
5.1.3 Further optimization.....	24
5.2 Challenges of the study and potential avenues for future research	26
5.2.1 Challenges caused by a dual data sources	26
5.2.2 Choice of Software and Programming Language	26
5.3 Conclusion	28
A Softwares & libraries version	29
A.1 Nodejs	29
A.2 Python.....	29
A.3 PostgreSQL	29
B Dashboard pages	30
Bibliography	42

List of Tables

4.1	Examples of SPARQL queries	20
-----	----------------------------------	----

List of Figures

1.1	Main goals of the Digital Botanical Gardens Initiative	2
2.1	Subject-predicate-object structure	5
3.1	Data types and fluxes in the DBGI.	9
4.1	Example of a treemap representation	14
5.1	Data Structure of both data sources	27
B.1	Home page	30
B.2	Explore page	31
B.3	Explore page in text-based search mode	32
B.4	Explore page in structure-based exact match search mode	33
B.5	Explore page in structure-based substructure search mode	34
B.6	Explore page in structure-based similarity search mode	35
B.7	Explore page in SPARQL-based search mode	36
B.8	Compound list page	37
B.9	Molecule page example	38
B.10	Organisms list page	39
B.11	Organism page example	40
B.12	Download page	41

List of Code Snippet

5.1	getTableData function.....	22
5.2	Handling of HTTP Requests on the home page: Example of Caching	22
5.3	HTTP GET Request for the <i>explore</i> page.....	23
5.4	example of prefetching on a script.....	24
5.5	Handling of HTTP Requests on the structure-based <i>explore</i> page:	25

Acknowledgements

I'd like to express my profound gratitude to my advisor, Pierre-Marie Allard, and the rest of the lab team, Emmanuel Defossez, Edouard Brühlhart, and Audrey Le Cabec. Their encouragement, tolerance, and indispensable aid throughout my MSc thesis have been truly beneficial. Additionally, my heartfelt thanks to the DBGI team for their warm reception and support.

I would also like to thank all my library companions - bioinformatician, psychologist and biologist - for all the breaks, coffees and happy hours that restored energy and helped keep motivation.

Special recognition must be given to Axel Giottonini and Marco Visani, whose guidance and assistance with my code were of great help, to Alissa Girard for being my unpaid therapist whenever motivation was lacking and to Yves Steiner for supplying free coffees that sustained focus and energy.

Additional gratitude is extended to my friends: Audrey Rossier, Maël Ravaz, Elia Betschen, and Lucas Orsini. Their companionship, manifested through shared drinks and insightful late-night discussions, has been both a source of enjoyment and intellectual enrichment.

My gratitude is further extended to my work colleagues for their vested interest and support in my thesis, with particular acknowledgment to Florian Progin.

Lastly, I wish to convey my heartfelt thanks to my family: Patrizia, Thierry, Thai and especially my sister, Petra. Their constant support, encouragement, and belief has served as the backbone of this journey. The success of this undertaking would not have been possible without such unwavering assistance.

List of Abbreviations

DBGI	D igital B otanical G arden I nitiative
DOM	D ocument O bject M odel
EMI	E arth M etabolome I nitiative
InChI	I nternational C hemical I dentifier
JBN	J ardin B otanique de N euchâtel
JBUF	J ardin B otanique de l' U niversité de F ribourg
JS	J ava S cript
KG	K nowledge G raph
RDF	R esource D escription F rameworks
SMILES	S implified M olecular- I nterface L ine- E nter S ystem
UNIFR	U niversity of F ribourg

Chapter 1

Introduction

1.1 Background and motivation for the project

The health and stability of ecosystems heavily rely on biodiversity. Numerous studies indicates that ecosystems with a broader range of biodiversity are more resilient and efficient, performing better in areas such as carbon storage, crucial for human well-being. [1]

However, our planet is currently confronting a serious threat to biodiversity. Smith et al. [2] already stated in 1993 that since about 1600, 486 animal species had been recorded extinct. In the same period, 600 plant species are known to have disappeared, which represents about 0.25% of the total.

A study conducted by Bakkenes et al. [3] estimates that by 2050, more than 16% of the European landmass will experience local species losses surpassing 50%. [4] Hence, the immediate need for conservation efforts cannot be overstated.

For these efforts to be meaningful and effective, characterizing and documenting the full scope of biodiversity is a key preliminary step. [1]

With this on mind, the Earth Metabolome Initiative (EMI) aims to record the full spectrum of metabolites found in all living organism, also referred to as the metabolome. [5] As a pilot to the project, the Digital Botanical Garden Initiative (DBGI) specifically focus on botanical collections, using massspectrometry techniques. [1] The choice of plants as a pilot subject is driven by multiple advantages. First, botanical gardens host a remarkable diversity of plant species within a compact area, thus providing a rich sample set for the initiatives. As an example, the Caribbean islands, regarded as a biodiversity hotspot [6], sustain a native flora of approximately 11,000 species spread across a land area of around 229,550 km² [7], leading to a biodiversity density of approximately 0.0479 species/km². Conversely, the JBUF contains about 5,000 species within a 0.018 km² area [8], resulting in a biodiversity density of approximately 277,778 species/km². This value is roughly 5.8×10^6 times greater, emphasizing the impressive diversity.

Furthermore, botanical assemblages might not possess the exhaustive diversity observed in natural ecosystems but they present a significant advantage due to their accessibility. These botanical collections are meticulously cataloged, labeled, and organized, allowing for an efficient sampling process. The environmental conditions within these collections can be controlled and monitored more readily compared to wild ecosystems, albeit not to the extent of those within laboratory settings. Importantly, these botanical gardens retain a level of diversity substantially greater than that found among laboratory-grown specimens, thus positioning themselves as a valuable intermediary for botanical study.[1]

This pilot has 8 main goals:

1. Establishing chemical extracts libraries of Swiss botanical gardens.

2. Digitize, through mass spectrometry, the chemodiversity of Swiss botanical gardens.
3. Gather chemical information and relevant samples metadata in a tailored knowledge graph.
4. Connect to existing ontologies (bio, chemo) and biodiversity digitization projects.
5. Establish web and programmatic interfaces for the query of the acquired knowledge.
6. Illustrate the feasibility and advantages of an end-to-end Open Science project.
7. Establish robust and scalable workflows for the digitization of wild ecosystems biodiversity. This point is of particular importance for the future Earth Metabolome Initiative.
8. Provide "molecular arguments" for biodiversity conservation policies and initiatives. [1]

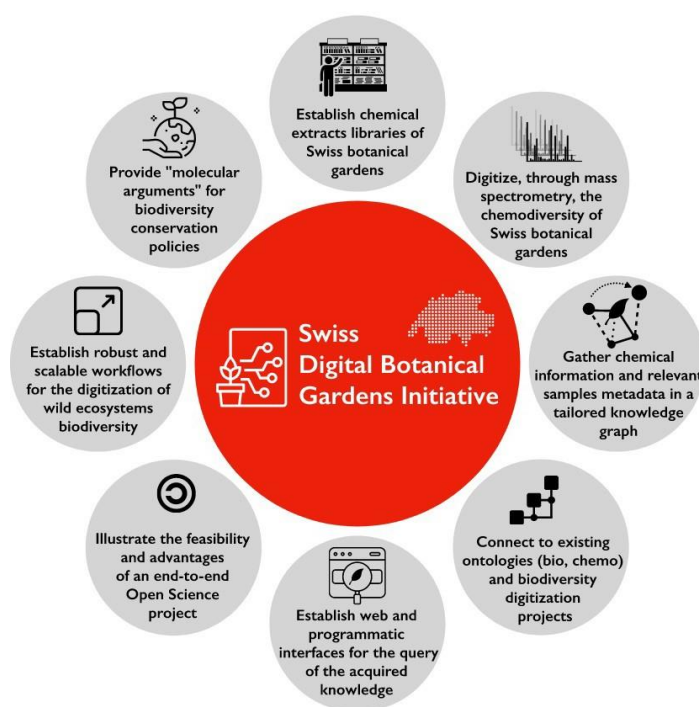


FIGURE 1.1: Main goals of the Digital Botanical Gardens Initiative. [1]

1.2 Brief overview of the research problem and objectives

This thesis aims to participate to the 5Th goal of the Digital Botanical Garden Initiative: *Establish web and programmatic interfaces for the query of the acquired knowledge*. More precisely, the project aimed to construct a dashboard designed to facilitate user searches for organisms or molecules using either textual or structural input. This platform is intended to display pertinent sample information, such as LCMS features, and provide a direct interface for querying the DBGI Knowledge Graph.

Several challenges needed to be addressed in this project. A primary hurdle was the data source. At the beginning of the project, the Knowledge Graph was not entirely constructed and populated. To bypass this, the decision was made to initiate the project using static data derived from the LOTUS initiative. This initiative aimed to catalog structure-organism pairs, essentially documenting the relationships between unique molecular structures and the organisms from which they were identified, on an open platform. [9] Indeed, the data listed on the LOTUS platform shared similarities with the DBGI data, making it an excellent starting point. Furthermore, LOTUS had already implemented a dashboard for data visualization, offering valuable assistance for our project, as detailed in Chapter 2.1.2.

As the project progressed, the decision was made to retain the LOTUS data on the platform, granting users the flexibility to select their desired data source for queries. Therefore the dashboard has SQL and SPARQL based queries depending on the selected source.

As a last objective, the Digital Botanical Garden Initiative is committed to embodying the principles of Open Science in an end-to-end manner. As such, the project's daily progress is chronicled in an **open-notebook**, which is publicly accessible for those interested in tracking its development. Additionally, all relevant code is freely accessible via the **Github DBGI page**.

Chapter 2

Literature Review

2.1 Discussion of existing research on creating data dashboards and using SPARQL queries for data visualization

Numerous dashboards are currently available for the visualization of molecular data. Some offer direct access to the visualization of LCMS data, while others focus on facilitating the search of molecules within a database. Our project strives to amalgamate these two functionalities.

The focus within this section is on dashboards that had a substantial influence on the project, yet many other valuable resources exist and could potentially provide value. Among these, dashboards developed by the **Zakodium team** could hold potential benefits for similar objectives as their company is dedicated to the development of tools for storing, processing, visualizing and exploiting scientific data, with a particular emphasis on the processing of NMR spectra, mass spectra, metabolomics data and statistical analysis.[10]

2.1.1 GNPS dashboard

The GNPS Dashboard, accessible at <https://gnps-lcms.ucsd.edu> [11], is a comprehensive online resource that enables the visualization, inspection, sharing, collaborative analysis, and pedagogical exploration of liquid and gas chromatography-mass spectrometry (LC-MS and GC-MS) data. It supports both private and publicly accessible MS data, including files stored in prominent MS data repositories such as GNPS/MassIVE [12], MetaboLights [13], ProteomeXchange [14], and Metabolomics Workbench [15]. [11]

The Dashboard is a plotly dash [16] app. Therefore it brought ideas and help in the primitive part of the project as the project was initialized with a plotly dashboard (see Chapter 3.2).

Furthermore, the GNPS Dashboard holds significant relevance for our project due to its capabilities in LCMS data visualization. It encompasses a broad range of features, many of which align with our initial expectations for our own dashboard and in some cases, even surpass them. Consequently, instead of duplicating these functionalities within our interface, we opted to directly provide access to the GNPS Dashboard for each sample, a feasible solution as the Knowledge Graph already contains the requisite links. To fully implement this, a *Sample Page* still needs to be developed, which would serve as the platform to display these links.

2.1.2 LOTUS resources

LOTUS represents one of the largest and most well-annotated freely available resources for Natural Products (NPs) occurrences (further details on the database will be provided in Chapter 3.1). LOTUS can be accessed via multiple platforms such as Wikidata and the Natural Products Online platform. The following sections aims to describe some of this resources.

Wikidata

Wikidata is a free, collaborative, multilingual, secondary database that collects structured data, providing support for wikis of the Wikimedia ecosystem and serving as a valuable resource for anyone globally.[17] The data in Wikidata is stored in the Resource Description Framework (RDF) datamodel, which is composed of triple statements (Figure 2.1), just as in a knowledge graph (later defined in chapter 3.2). [18]

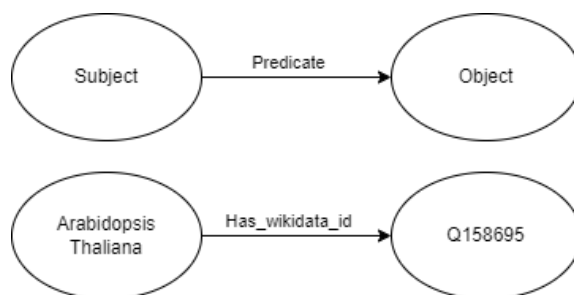


FIGURE 2.1: Subject-predicate-object structure with an example:
An entity (subject) is associated to a simple value (an object) through some property (a predicate).

The relevance of this webpage for this project stems from the fact that data from LOTUS can be found within this database. Additionally, significant portions of the data are linked to the Wikidata database. For instance, all organism and molecular structure Wikidata IDs can be traced within our data. This linkage has facilitated the addition of references to Wikidata from the dashboard.

A noteworthy feature of Wikidata is its query service, which offers a SPARQL endpoint. The design and functionality of this service served as a valuable example for the project, providing insights into how an interface for direct database querying could be implemented in the project's own database.

While Wikidata presents a range of compelling features, its interface is not particularly user-friendly, emphasizing the necessity for a dashboard.

Natural Products Online

Natural Products Online is an open-source, open-data repository for natural products, which is developed and maintained by the Steinbeck group at the University Friedrich-Schiller in Jena, Germany. They offer a range of dashboards for molecular data, as the [Sugar Removal Service](#) [19]. This service allows for the computational elimination of sugar structures from the user-provided molecules, enhancing the efficiency of understanding the primary biological activities of the molecule, since these highly repetitive structures can potentially hinder this process.

Another of their significant resource is the [COCONUT](#) (COLleCtion of Open Natural

ProdUcTs) [20], which aggregates data from over 50 open natural product resources and offers unrestricted and free access. Each record corresponds to a *flat* natural product structure and, where available, links it to its known stereochemical forms, literature, producing organisms, natural geographical presence, and various precomputed molecular properties. It is resumed in a database containing 407,270 unique natural products.

The COCONUT dashboard bears a strong resemblance to the LOTUS dashboard, which is another of Steinbeck group's dashboards and is built on the LOTUS resource, making it of considerable relevance to this project.

The dashboard developed for this thesis borrowed several concepts from the latter. Predominantly, the structure-based search in our project offers nearly identical options. However, our current dashboard lacks an *Advanced Search* function, a notable feature in the LOTUS dashboard. It allows for more precise structure searches, incorporating structural properties. For instance, users can search for a molecule based on a specified weight range, or look for molecules that do or do not contain sugars, or specify a particular number of oxygen atoms, etc. The feature also supports the inclusion of Molecular descriptors such as a natural product-likeness score [21]. This level of specificity significantly enhances the search precision, and therefore this kind of functions will be incorporated into the DBGI dashboard in the future.

Moreover, the download feature within the LOTUS interface exhibits a more advanced functionality compared to the corresponding page created for this project. Its refined capability enables users to selectively retrieve specific data, a contrast to the DBGI's platform which currently permits only the complete dataset download. Therefore, potential enhancements to the DBGI download feature could be guided by the exemplary model of the LOTUS interface.

However, it is important to note the disparity in technical implementation. The LOTUS dashboard utilizes Javascript, Java, and Kotlin, while our project relies solely on Javascript for backend development.

Kotlin, despite being a relatively new language (having been launched in 2016), boasts full compatibility with Java and operates across various platforms. The language is concise, easy to learn, and is predominantly used for mobile application development, though it's also seen application in web development, server-side applications, data science, among others. [22] Nevertheless, owing to its youth, Kotlin does not have as large a user community as Javascript does. [23]

Accordingly while the ideas may be inspired by the LOTUS dashboard, the codebase is distinct and tailor-made to fit our project's requirements.

Considering the time constraints of our project (6 months), and given our late consideration of Kotlin, we decided to stick with Javascript as our primary backend language. This decision was primarily to leverage the vast resources and community support available for Javascript, ensuring we could maximize our productivity and output within the project's timeline.

Nprod.net

Another web interface, *LOTUS* [24] utilizing the LOTUS database was developed with *Streamlit*¹ - a less feature-rich tool, but one that simplifies the creation and sharing of custom web apps for machine learning and data science. [25]

This dashboard includes a particularly interesting attribute allowing users to contribute their own compounds if they are missing from the database. This feature enables community contributions to enhance the data quality of LOTUS, although it depends on the

¹Since the writing of this thesis, they transitioned to a Plotly Dash application.

user's integrity to verify that the data is not already present and the molecule is accurately represented.

While no features from this dashboard were directly incorporated into the DBGI dashboard as it was released in the late state of the project, it could potentially offer valuable ideas for further refinement of the DBGI platform.

Despite this, Node.js appears to remain a better fit for the project scope, offering broader possibilities than Streamlit. Nonetheless, future exploration of features inspired by this second dashboard could be beneficial.

Chapter 3

Methodology

3.1 Description of the data sources and how they were collected and processed

As mentioned before, the LOTUS database was used as a first source of data as it had the advantages of being static and already populated with a lot of data. In the subsequent phase of the project, the code was modified to query the continuously evolving data of the Digital Botanical Garden Initiative organized as a Knowledge graph.

LOTUS

The LOTUS database is made of 750,000+ referenced structure-organism pairs. [9]

The data was extracted from thirty-eight electronic Natural Products resources, enumerated in Rutz et al. [9]. The project did therefore not directly collect the samples; instead, only the metadata was gathered from the LOTUS team.

Given the substantial variability in the format and contents across each Natural Products resource, a process of standardization was necessitated. This standardization was primarily achieved through scripts adept at harmonizing and categorizing knowledge from each respective source. [9]

DBGI

Data collection for the Digital Botanical Garden Initiative (DBGI) is currently underway at the Botanical Gardens of Fribourg (JBUF) and Neuchâtel (JBN). These two gardens serve as pilot sites before expanding the project to other Swiss and international botanical gardens. They were selected for pragmatic reasons and also due to their unique features, as they each specialize in different types of plants. [1]

Two categories of objects are considered: physical and digital (Figure 3.1). Physical objects are sampled from the botanical garden and then subjected to Ultra High Performance Liquid Chromatography coupled with High Resolution Mass Spectrometry (UHPLC-HRMS) to derive fragmentation data. Subsequent annotation utilizes standards, experimental libraries, and ISDB - a database of in-silico predicted MS/MS spectra of Natural Products. All procedures involving physical objects, such as sampling, conservation, and extraction, generate metadata which are meticulously collected and stored in various locations. The majority of the sampling data - pictures, species, geolocation, etc. - is stored using the iNaturalist platform. In addition, a PostgreSQL database manages all species, specimens, and experimental metadata and can be accessed via a no-code database (Directus [26] - Only accessible when connected to the UNIFR network for now -). Lastly, the mass-spectrometry data is made accessible through the GNPS-dashboard for visualization and further exploration. [1, 27, 28]

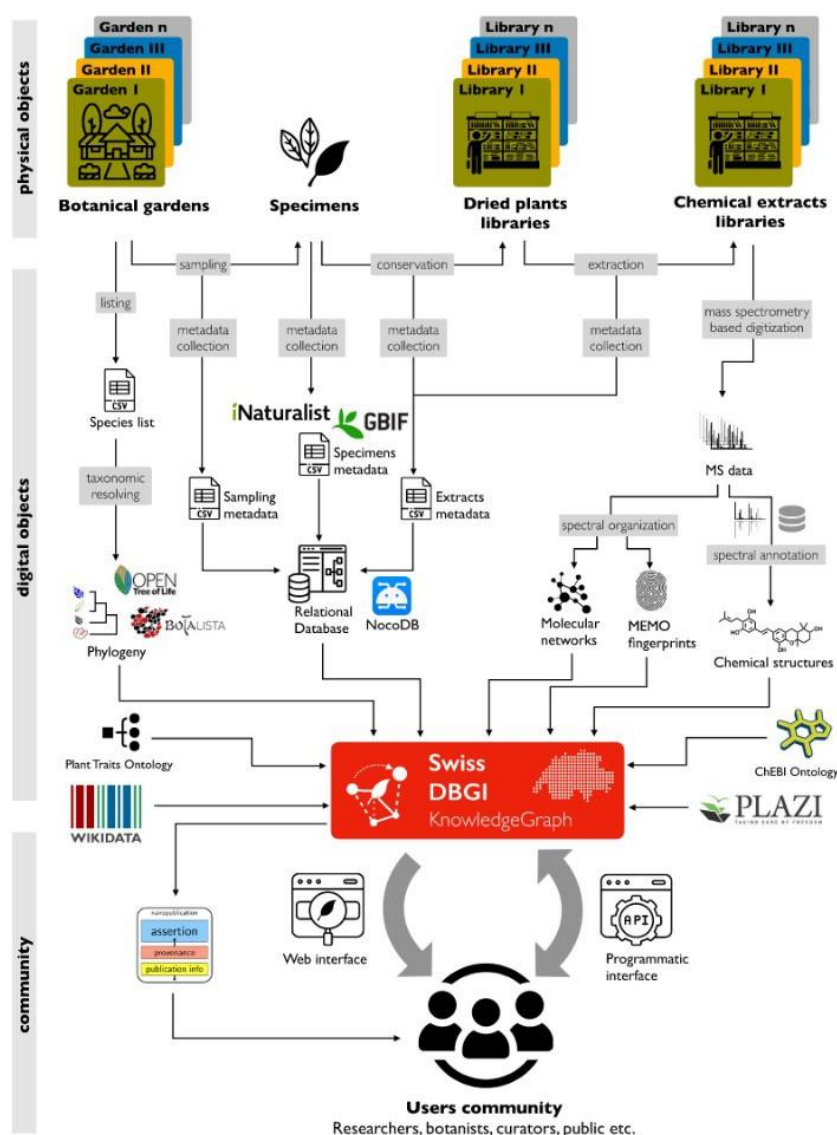


FIGURE 3.1: Data types and fluxes in the DBGI. [1]

3.2 Discussion of the software tools and programming languages used to build the dashboard and interface with the knowledge graph and LCMS profiles

The project was initialized with a Python Plotly dashboard. This choice was initially appealing due to Python's simplicity and the wealth of libraries offering built-in visualizations. However, given that both the Plotly dash library and several molecular tools were based on Javascript, sticking to Python could lead to various constraints. Thus, the strategy pivoted towards converting the Plotly dashboard into a Node.js application. The Python code is still accessible on the [GitHub](#) repository, but it's no longer being updated.

As said before, considerable emphasis was placed on the use of open-source programs. Despite certain drawbacks, such as occasionally incomplete documentation, open-source

softwares offers a supportive community ready to assist, complementing the software itself. [29] They also provides usage flexibility, potentially enhanced security and permits reusability.

JAVASCRIPT / NODE.JS

Node.js is an open-source server environment utilizing JavaScript. Its architecture provides several advantages [30, 31]:

1. **Single-threaded:** Node.js operates primarily on a single thread using the V8 JavaScript engine. It means there's only one main thread to execute all your JavaScript code. This reduce the complexity and allow easier programming and debugging.
2. **Non-Blocking & Asynchronous programming:** Node.js uses an event-driven architecture. When an operation could take a while (like reading from a database), rather than waiting for it to complete, Node.js registers a callback and then continues with processing other tasks. This way, it doesn't block the main thread. The callback is then executed after the operation concludes, allowing for simultaneous processing of other tasks.

In summary, while Node.js is single-threaded, its non-blocking and asynchronous nature ensures that this single thread can handle many tasks efficiently without waiting for one to complete before moving to the next. This allows to be memory-savvy and time-efficient, which is beneficial when managing large datasets, just as in this project.

The project utilized the **Express** module, a robust web application framework layered atop Node.js, which significantly streamlines the management of servers and routes. Express notably reduces coding time, yet allows for the development of efficient applications. One of its key advantages is the ability to handle requests using Middleware, a request handler with access to the application's request-response cycle. Furthermore, Express facilitates swift integration with databases and supports templating engines to generate dynamic web page content by crafting HTML templates on the server. [32] This feature enables a Model-View-Controller architecture, partitioning extensive applications into distinct sections, each serving a unique purpose, thereby enhancing code readability and maintainability. [33]

The analysis of molecular structures was conducted utilizing the **JSME** [34] library, an open-source molecular editor written in JavaScript. It is the direct successor to the JME Molecule Editor applet which was used on the dashboard for presenting the molecular structure as an image on the user interface. This tool offers comprehensive support for the drawing and modification of molecules and reactions, enhancing the efficiency and speed of molecule creation, even for complex and sizable molecules. This is achieved through an integrated substituent menu and several keyboard shortcuts that grant rapid access to the most frequently used editing functions. Notably, the JSME library allows for the export of molecules as SMILES. Moreover, the utility of the applet extends to its function as a query input tool for molecular database searches, as it assists in the construction of complex substructure queries.

The **plotly** [16] library constituted one last significant JS tool employed in this project. It is a sophisticated, declarative charting library, constructed on the foundations of **d3.js** [35] and **stack.gl** [36]. It encompasses a broad repertoire of over 40 distinct chart types, inclusive of 3D charts, statistical diagrams, and SVG maps. It is free and open source and was used in the app to create interactive charts such as treemaps.

HTML

HTML, or Hypertext Markup Language serves as the basis for views in Node.js. It is the foundational language used for creating webpages. It is responsible for structuring the content on the web, providing a means to describe the semantics of document text and embedded resources such as images and scripts. [37]

CSS

Cascading Style Sheets (CSS) govern the visual presentation of HTML elements across different media types. It significantly streamlines web design by uniformly controlling the layout of multiple web pages at once, thereby ensuring consistency and efficiency in the visual representation of the website content. [38]

POSTGRESQL / SQL

LOTUS data storage was managed via a PostgreSQL relational database, with data retrieval within the Node.js application facilitated by the pg library. [39] As an open-source, community-driven system, PostgreSQL offers compatibility with a multitude of programming languages and platforms, a feature that optimizes its integration with various tools. This versatility significantly simplifies the process of migrating the database to alternative operating systems or integrating it with specific tools, a substantial advantage, especially for pilot projects that may require iterative adjustments and scalability. [40] SQL, a fundamental programming language for accessing and manipulating databases [41], was used to execute queries on the PostgreSQL database.

GRAPHDB

GraphDB is a powerful, scalable RDF database solution that facilitates the efficient loading and utilization of linked data cloud datasets and personal resources. It operates in alignment with the RDF4J framework interfaces, the W3C SPARQL Protocol specification, and supports all RDF serialization formats. [42]

One of GraphDB's distinctive capabilities is its ability to conduct semantic inferencing at scale, enabling users to generate new semantic insights from pre-existing facts. It handles large loads, queries, and inferencing in real-time, working over a persistent storage layer. Consequently, GraphDB can process tasks rapidly, even when dealing with large ontologies and knowledge bases. [42]

GraphDB is capable of managing billions of explicit statements on standard desktop hardware and tens of billions on commodity server hardware. [42]

SPARQL

Data within the Digital Botanical Garden Initiative (DBGI) was stored within a Knowledge Graph hosted on GraphDB and accessed via SPARQL queries.

Knowledge Graphs are a type of graph database that efficiently encapsulates structured data via interconnected subject (e.g. *Arabidopsis Thaliana*) - predicate (e.g. *has_wikidata_id*) - object (e.g. *Q158695*) triplets. This model surpasses conventional alternatives by structuring data more effectively than text formats, eliminating redundancy of tabular data, and easing the querying of complex data, typically challenging in relational databases.

[43] Knowledge Graphs focuses on the relationships between entities, which enables inference from existing data and it also excels in handling massive datasets. [44] SPARQL, the standard query language for Linked Open Data and RDF databases, is designed to navigate a wide array of data types, proficiently extracting information concealed in heterogeneous data formats and sources. Offering functionalities comparable to SQL for NoSQL graph databases like Ontotext's GraphDB, SPARQL surpasses SQL by permitting queries to extend beyond a singular database. This capability to execute federated queries enables access to multiple data stores, also known as endpoints. [45] In this project, the significance of SPARQL querying becomes evident, as both Wikidata, and therefore LOTUS resource utilize it. Federated queries facilitate the interlinking of these resources which will be of interest in the project's continuation.

PYTHON

Python libraries like RDKit [46] simplified the process of handling molecular data. Consequently, Python scripts were employed as the back-end for computations or transformations, such as converting SMILES into InChi and InChiKey, on the data.

Chapter 4

Results

4.1 Presentation of the dashboard interface and its various features and components

The dashboard is organized as a multi-page application, enabling users to explore both databases via various search methods. It can be accessed on <http://dashboard.dbgi.org/> when connected on the UNIFR network. Furthermore, screenshots of the pages can be found in the appendix [B](#).

The top of each page features a navigation bar, ensuring accessibility to each section from any page within the app. The navigation bar contains the title, which when clicked, redirects the user to the home page. Two buttons are present, one also directing the user to the home page and the other to the download page.

Furthermore, there are two dropdown menus. The first, titled *Explore*, allows users to navigate the databases in different ways, such as *Explore Whole Dataset*, *Explore Molecular Diversity*, and *Explore Organisms Diversity*.

The second dropdown menu, *Information*, provides access to essential documentation and background information about the DBGI. It also houses a link to the DBGI organization's GitHub page, enabling direct access to the project's codebase and updates.

4.1.1 Home ([B.1](#))

The Home page serves as an introductory interface for the user, from where they can navigate to various sections of the application.

Additionally, it provides insightful project statistics, such as the proportion of various taxonomic categories - including phylum, class, and species - that have already been profiled within the project. This feature offers users a snapshot view of the current state of the project's data collection and profiling progress.

The values for the total number of taxon known were taken from the *Catalogue of Life* website [[47](#), [48](#)].

4.1.2 Explore whole dataset ([B.2](#))

This section gives the user the ability to access and navigate through the LOTUS and DBGI databases using a variety of methods. The primary methods include a text-based search where the user can hunt for specific words within the databases, a structural search that allows the user to sketch a molecule's structure and look for it, and direct SPARQL queries. It should be noted, however, that direct SPARQL queries can currently only be used with the DBGI data as the LOTUS database is housed in a relational database that relies on SQL.

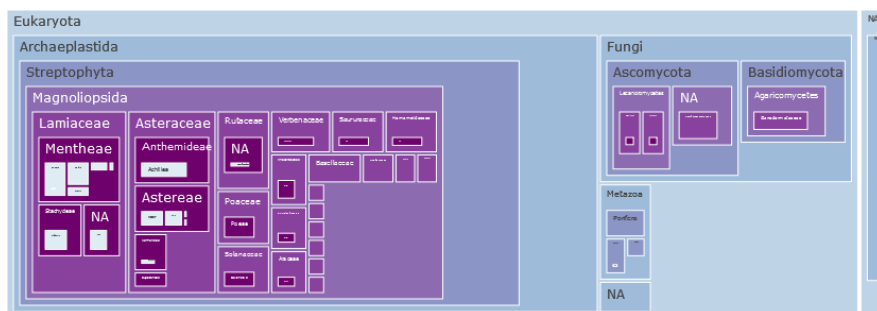


FIGURE 4.1: Example of a treemap representation obtained by doing a structure-based search on LOTUS. Each rectangle represents a taxon, with nested rectangles representing subtaxon, in which the search had a hit. The size of the rectangles are an indication of quantity of hits in this taxon. The interactivity of plotly allows to navigate through the taxons by clicking on the rectangle of interest.

The user can choose which dataset to browse by selecting the datasource in both text- and structure-based search. When browsing the DBGI data, the back-end queries are SPARQL queries and when browsing the LOTUS data, the back-end queries are SQL queries.

The results from both search methods can be visualized in two distinct formats based on user preference: tabular form or treemap representation (Figure 4.1). The treemaps are generated using the Plotly [16] library for JavaScript. The organization of the treemaps mirrors the phylogeny of the organisms where the molecule or substructure can be identified. A colorblind-accessible color scheme is employed for this visualization. Owing to Plotly's interactive features, users can engage with the graph, zooming in on sections that pique their interest and export the plot as a png if wanted.

The output of the search operation is constrained to a user-defined quantity of rows.

The next sections go through the specificity of each search method.

TEXT-BASED (B.3)

The text-based search operation conducts a case-insensitive scan for the text input within a user-selected column. The resulting output is contingent on the specified row limit and the chosen display format.

STRUCTURE-BASED

In the structure-based search page, the user can draw a molecule and then search for hits using different methods. The JSME object provides a variety of shortcuts to facilitate user interactions when drawing molecules. Detailed instructions for these can be accessed on the [JSME help page](#).

One of these shortcuts allows users to insert the SMILES notation of the molecule they want to represent, eliminating the need for manual drawing.

Exact Match (B.4)

The search for an exact match can be accomplished using either the InChI or SMILES representation of the drawn molecule, each offering distinct advantages:

SMILES strings are relatively concise and easily interpretable. They permit various representations for the same molecule (canonical and non-canonical SMILES), which introduces the potential for search inconsistencies but also allows for greater flexibility. Additionally, SMILES notation includes useful features such as isotopic specification, chiral centers, and aromaticity. [49, 50]

InChI strings, on the other hand, provide a unique and consistent descriptor for each compound, ensuring precise searches across databases. They are particularly adept at handling larger molecules more efficiently. InChI also incorporates tautomeric and isotopic information and can specify absolute stereochemistry. However, the trade-off is that InChI strings tend to be longer and more intricate than SMILES strings, possibly making them less user-friendly. [49, 50]

In summary, for optimal searching accuracy, InChI is preferable as it minimizes the chance of ambiguity during searches. Conversely, if a more human-readable and adaptable system is required, SMILES might be more suitable.

Based on the selected representation, the dashboard will query the designated database to retrieve all relevant matches where the InChI or the SMILES align with the depicted structure.

Substructure Search (B.5)

The structure-based search feature empowers users to query for molecules that contain a specific drawn substructure.

In practice, the user drafts a unique structural design, which the application then uses as a query to search the database for molecules incorporating this substructure.

The backbone of this search process is likely a variant of the Ullmann subgraph isomorphism algorithm, one of the most frequently employed algorithms for subgraph isomorphism search, as pointed out in the Ehrlich and Rarey [51] paper. Although the rdkit [46] (used for LOTUS substructure search) and sachem [52] (used for DBGI substructure search) libraries' documentations do not explicitly mention the algorithm implemented in their respective functions (*Chem.HasSubstructMatch()* for rdkit and *sachem:substructureSearch* for sachem), indications from the source code, at least for Rdkit, lend support to this supposition.

This algorithm basically helps determine whether a given pattern graph is isomorphic to any subgraph of a larger target graph. It provides an efficient solution to the subgraph isomorphism problem by utilizing backtracking and pruning techniques. It works by building a binary matrix that represents possible matches (edges) between nodes in the two graphs, and then gradually refines this matrix by removing edges that are not consistent with a one-to-one mapping. The remaining matrix then represents a valid subgraph isomorphism if such exists. [53]

Similarity Search (B.6)

The Jaccard-Tanimoto coefficient is utilized in conducting the similarity search, a decision influenced by its widespread application and notable performance, as highlighted in the Todeschini et al. [54]. This coefficient remains among the most commonly used metrics in the field, hence its selection for this project.

The Tanimoto coefficient, denoted as $\frac{c}{a+b-c}$ [55], represents the ratio of shared features between two compounds to their combined features. Here, c denotes features common to both compounds, whereas a and b correspond to features unique to each individual compound, respectively. The coefficient ranges between 0 and 1, with values closer to 1 suggesting higher similarity. [56] However, it's crucial to note that a Tanimoto coefficient of 1 doesn't imply identical compounds, but rather that their structural descriptors or on-bits in a binary fingerprint match. [57] There might be other aspects of the molecules not captured by the specific fingerprinting method used that differentiate them.

For example, two molecules could have the same functional groups in the same locations (resulting in the same "on-bits"), but they might differ in 3D conformation or stereochemistry, which can significantly affect their biological activity or other properties.

SPARQL (B.7)

The concluding segment of the *Explore Whole Dataset* division involves the SPARQL search. This section empowers users to directly interface with the DBGI Knowledge Graph by crafting and executing their own SPARQL queries.

However, it is important to note that this search functionality currently presents some security vulnerabilities, as elaborated in Chapter 4.2, which require remediation.

4.1.3 Explore Molecules (B.8)

This segment of the application allows users to query the LOTUS dataset for specific molecules via their names. Additionally, it showcases the ten most frequently encountered molecules within the LOTUS database, thus highlighting key molecules of interest. The molecules name displayed are links to a page that display informations about the molecule.

4.1.4 Molecule Page (B.9)

Upon choosing a molecule from either the *Explore Molecules* page or the *Explore* table results, the user is redirected to a page offering detailed information about the molecule. Adjacent to the name of the molecule, the Wikidata ID is displayed, which serves as a direct link to the Wikidata entity of the respective molecule. Following this, the 2D and 3D structures of the molecule are presented. Next, the chemical taxonomy of the molecule is listed. LOTUS employs two distinct chemical classifiers: NPclassifier [58], which categorizes based on the producing organism's taxonomy, biosynthetic pathway, biological properties, and presence of chemical substructures, and Classyfire [59], which is structure-based. The Classyfire taxonomy was chosen arbitrarily between both to be displayed on this page.

Lastly, the page provides a list of organisms in which this molecule has been identified in the LOTUS database. Furthermore, the names of these organisms serve as links to respective organism pages, offering an in-depth view into each organism.

4.1.5 Explore Organism (B.10)

Similar to the *Explore Molecules* section, this segment provides the user the ability to search for a specific organism within the LOTUS database.

In order to aid the user and provide a starting point, a list of the top 10 most prevalent organisms is also displayed.

Importantly, each result acts as a direct link to a dedicated page for the specific organism, offering a deeper exploration of each individual organism.

4.1.6 Organism page (B.11)

This page presents information pertaining to organisms catalogued in the LOTUS database, with each page dedicated to a specific organism. Access to these pages can be achieved through either the *Explore Organism* page or the *Explore* table results.

Alongside the organism's name, its Wikidata ID is displayed, which serves as a direct link to the corresponding Wikidata entity page.

Furthermore, the Wikidata image of the organism is displayed below, accompanied by its phylogenetic information.

Lastly, a list of all molecules identified within this organism, as recorded in the LOTUS database, is displayed. Notably, each molecule name serves as a hyperlink to a dedicated page for that specific molecule, thereby facilitating further exploration.

4.1.7 Download (B.12)

This segment provides users with the capability to download the LOTUS dataset in either CSV or JSON format. Up to now, the user can only download the entire dataset but further enhancement will allow to download only a subset (see Chapter 4.2.5).

4.1.8 Informations

The *Information* dropdown menu presents users with three distinct options: *About DBGI*, *Documentation*, and a GitHub link. The *About DBGI* page provides users with an overview of the project along with a hyperlink to the official webpage. The *Documentation* page encompasses comprehensive instructions and guidance for utilizing the dashboard. The final option is a direct link to the *Digital Botanical Garden Initiative* Organisation's GitHub page, allowing users easy access to the source code and project updates.

4.2 Perspectives

As expected, 6 months wasn't enough to get all the features wanted. This section enumerates some necessary improvements, some of which are more urgent than others. It should be noted, however, that additional enhancements are undoubtedly needed beyond those listed here. Several potential enhancements were already suggested in Chapter 2.1.

4.2.1 Error handling

A crucial enhancement required for the application involves improving the error handling of the code. Errors can stem from several sources, such as coding mistakes by the programmer, incorrect user input, or other unpredictable circumstances.

Some progress has already been made in this area, primarily through the implementation of basic *try-catch* blocks but a lot still needs to be done.

The most important being the need to incorporate input validation functions. At present, the querying management (for both SPARQL and SQL) is inadequate and poses a significant risk to the data. For instance, it would be possible for a user to initiate a database query that could modify or even delete the data. Therefore, incorporating robust input validation measures is imperative to ensure data integrity and protection.

Another critical perspective for enhancing error handling involves the adoption of unit

testing. As of now, the application lacks a comprehensive unit testing framework. Incorporating such a framework would enable the development team to test individual pieces of code in isolation, making it easier to identify and handle errors at an early stage. This would not only improve code quality but also make the application more maintainable and easier to update in the future.

Finally, although conventional testing methods can catch many issues, they often rely on predetermined scenarios which might not cover all edge cases. Fuzzing, a testing technique that involves feeding a program with random data to identify vulnerabilities and crashes, could be incorporated to enhance error handling. By adopting fuzz testing, the application would be better equipped to handle unexpected inputs or scenarios, increasing its resilience and reliability.

These additional measures will significantly bolster the application's error-handling capabilities, ensuring a more stable and secure user experience.

4.2.2 Visualisation options

Several enhancements could be made to the current visualization methodologies. To begin, the data representations on the homepage could be refined for a more engaging user experience. Additional visual elements could also be introduced to provide a more comprehensive overview of the data immediately upon page entry.

A treemap could be integrated into the organism identity page (Chapter 4.1.6). This visual tool would facilitate an immediate understanding of the types of molecules associated with a given organism.

Furthermore, displaying the Tanimoto coefficient for each hit in a similarity search could significantly augment the user experience. By indicating the degree of similarity between molecules, users would gain a clearer understanding of their search results.

Lastly, it should be noted that the aesthetic design of the pages received limited emphasis during development. By embracing enhancement opportunities, such as integrating the Bootstrap CSS library [60] and exploring other potential avenues, the user experience can significantly be improved. A well-designed interface not only offers visual appeal but also facilitates ease of navigation and interaction, ultimately enhancing the overall usability and effectiveness of the dashboard for its users.

4.2.3 SQL search

Similar to the provision of a dedicated page for direct SPARQL querying, incorporating a page for direct SQL querying on the LOTUS database could serve as a valuable addition, enhancing the overall functionality of the application.

4.2.4 Sample page

To mirror the application's current structure where individual pages are allocated to each molecule and organism, it would be advantageous to also create a unique page for each sample, accessible directly from the *explore* results.

This bespoke page could display comprehensive information about a particular sample, including its species, the period and season of its collection, the part of the plant that was harvested, among other specifics.

Furthermore, a meaningful addition to this page could be the inclusion of LCMS features,

complete with a link for users to examine their mass spectrum on the GNPS dashboard. Such an enhancement would directly address one of the dashboard's primary objectives: creating a seamless link between the visualization of taxonomic data and LCMS data, thereby facilitating a comprehensive understanding of the sample data.

4.2.5 Download page

At the moment, the download page presents some issues that need to be addressed. A probable cause appears to be the excessive data download attempted by the page, necessitating an optimization process to streamline the data retrieval operation. Moreover, the introduction of functionality to download more granular, specific data, instead of the entire dataset, would greatly enhance user convenience. This can potentially be achieved via implementation of dropdown menus and customized queries. Additionally, a viable strategy could involve incorporating a download button directly within the *Explore* results section. This would enable users to download solely the results corresponding to their specific search, enhancing both user experience and data management efficiency.

4.2.6 Functionalities for DBGI data

Certain functionalities available for the LOTUS database have not yet been integrated for the DBGI data.

In particular, the *Explore Organisms* and *Explore Molecules* pages are currently configured to utilize only the LOTUS data, thus, restricting searches to the LOTUS resource. Likewise, the text-based *explore* page shares this limitation, as it only yields results from the LOTUS resource, not the DBGI database.

Moreover, the current search functionality within the DBGI data set is limited to displaying results in a table format, with the capability to produce graphical output yet to be implemented.

4.2.7 Cross-Browser and Device Compatibility Testing

Ensuring the compatibility of web applications across different browsers and devices is paramount to providing a universally accessible user experience. Up to this point, the dashboard has been primarily tested on Mozilla Firefox versions 110 to 116, utilizing a laptop.

However, the digital landscape is diverse. Users access applications via a multitude of browsers—like Chrome, Safari, and Edge—and on a plethora of devices ranging from smartphones to tablets to desktop computers. The variance in rendering engines, screen sizes, and device capabilities can introduce unforeseen issues or challenges, which may compromise the dashboard's functionality or appearance.

Given this, there is a recognized need for extensive testing across various browsers and devices. Such comprehensive testing will identify potential issues and ensure that the dashboard remains consistent and fully functional, irrespective of how users choose to access it.

4.3 Examples of SPARQL queries

The subsequent section presents a table featuring a variety of SPARQL query examples that users can execute within the knowledge graph on the SPARQL *explore* page.

TABLE 4.1: Examples of SPARQL queries

#	Query description	Returns	Link
1	What are the ENPKG classes?	The list of ENPKG classes: LCMS-Feature, ChemicalEntity, InChiKey, ...	SPARQL
2	Which species served as the source for the sample?	135 pairs of sample name and associated taxon	SPARQL
3	How many peaks are detected within the mass spectrum of each respective sample?	132 pairs of sample name and number of peaks found.	SPARQL
4	What quantity of samples exhibit a particular peak? The analysis should be constrained to only those instances with more than five samples.	10'113 peaks with the number of samples that exhibit that peak. The maximum count of samples correlated with a single peak reaches up to 132.	SPARQL
5	What is the count of unique kingdoms, orders, families, genera, and species represented within the DBGI Knowledge Graph? This question necessitates a federated query that integrates data from ENPKG and Wikidata to generate the results.	The DBGI Knowledge Graph comprises one kingdom, 32 orders, 48 families, 76 genera, and 86 unique species. This query is designed to extract the data that is then employed in the graph displayed on the <i>home page</i> .	SPARQL

Chapter 5

Discussion & Conclusion

5.1 Discussion of how the code was optimized for efficiency and speed

The optimization of code constitutes a significant component in the development of an application, conferring multiple advantages. Initially, optimization augments code readability, aiding any developers who might need to work with the code subsequently. Additionally, optimization facilitates a more rapid execution of the code, an aspect critical for enhancing user experience. Finally, effective error handling necessitates improved testability, which is another vital benefit of optimization.

5.1.1 Javascript optimization

Regrettably, the potential for negative performance impact is higher in JavaScript. This language can substantially affect download times, rendering performance, and CPU and battery usage. [61] Consequently, specific strategies were employed to optimize the code to the greatest extent feasible. Some of these strategies are summarized in the following section.

The initial strategy involved a global approach to separating routes into multiple files. Several advantages are associated with this method: it enhances maintainability by allowing each file to concentrate on a distinct part of the application, simplifying the process of locating and modifying specific routes without the need to navigate a large file. [61] This separation not only facilitates debugging but also aids in comprehending the code and encourages reusability. In some cases, splitting routes can contribute to performance optimization through the lazy loading of particular segments of the application. This lazy loading ensures that only the requisite code is loaded when necessary, potentially reducing the initial load time. Finally, this approach often results in a more organized and cleaner code structure.

At the level of a JavaScript file, the initial phase of optimization involves the removal of superfluous code. Indeed, the most performant and least blocking JavaScript code is the code that remains unused, emphasizing the need to utilize as little JavaScript as possible. [61]

Several methods exist to accomplish this reduction. Initially, the usage of modules can significantly minimize the volume of code. Subsequently, the establishment of variables and functions where redundancy occurs can also markedly enhance code efficiency. This was mostly the case in the *explore* framework (Code Snippet 5.1). For instance, the SQL queries of the structure-based search are nearly identical. As a result, the query was initialized as a basic query (line 5) that every search required, and portions were appended

to this fundamental query depending on the search criteria, such as whether the search was filtered by taxon or not (line 10). The queries were also parameterized, permitting the use of the same query with varying values (line 2), e.g. the *max return* value.

```

1 // Function to fetch data for table display
2 async function getTableData(column, structure, group, max) {
3   try {
4     // SQL query to select data where column value is in the provided
      structure
5     let query = 'SELECT * FROM data WHERE ${column} = ANY($1)';
6     let params = [structure];
7
8     // If group is provided, append to SQL query
9     if (group) {
10      query += ' AND $2 = ANY (array[${taxonomyColumns.join(', ')}]';
11      params.push(group);
12    }
13
14    // Limit the query results
15    query += ' LIMIT $' + (params.length + 1);
16    params.push(max);
17
18    // Execute the query
19    const result = await db.query(query, params);
20    return { result };
21  } catch (error) {
22    console.error('Error in getTableData function with column ${column},
      structure ${structure}, group ${group} and max ${max}:', error);
23    throw error; // Re-throw the error if you want to handle it further up
      the call stack
24  }
25 }

```

CODE SNIPPET 5.1: getTableData function

This function is employed within the *explore* routes to facilitate the extraction of result tables from the LOTUS database. It is systematically applied to all search methods associated with the LOTUS data source.

Functions were designed to accommodate various scenarios, such as rendering tables or graphs and managing disparate data sources. These tailored functions enhance the code's flexibility and adaptability, contributing to a more robust and responsive application.

The loading of data also challenged the optimization of the application. For example, the graphs displayed on the home page (Figure B.1) necessitated computationally intensive queries to ascertain the number of species sampled. Given that this number seldom changes, continuous re-querying upon each page load is redundant. This issue was addressed using **caching**, which creates a temporary copy of the data. If the cache contains the required data, the browser will prioritize it to load the page; otherwise, it will default to re-querying the data (Code Snippet 5.2).

```

1 router.all('/', async (req, res) => {
2   try {
3     // Fetch the data from CountTaxon
4     let data = Cache.get( "myKey" ); // Initialize data with cached
      value
5     if (data == undefined){
6       data = await CountTaxon(); // Assign directly to the data
      variable

```



```

7      Cache.set("myKey", data, 100000);
8      }
9      // Assume data returns a row of results as an object e.g., {
    species_count: 10, order_count: 20, ...}
10     // Pass this data into the 'home' view
11     res.render('home', {results: data});
12   } catch (error) {
13     console.error('An error occurred:', error);
14     res.status(500).send('An error occurred');
15   }
16 });

```

CODE SNIPPET 5.2: Handling of HTTP Requests on the home page:
Example of Caching

Moreover, the incorporation of consistent error handling within try-catch blocks (Code Snippet 5.3) establishes a robust mechanism to gracefully address unexpected situations. This approach safeguards the application's stability and user experience by providing controlled responses to potential failures or errors.

```

1  // Router for '/explore' endpoint, renders the explore page with table
    columns
2  router.get('/explore', async (req, res) => {
3    try {
4      const columns = await getTableColumns();
5      res.render('explore', { columns });
6    } catch (err) {
7      console.error(err);
8      res.send('Error while fetching columns names');
9    }
10  });

```

CODE SNIPPET 5.3: HTTP GET Request for the *explore* page
This code demonstrates a try-catch block, where the try block executes if no errors occur, and the catch block executes if an error is encountered.

Finally, the integration of promise constructs and async/await functions (Code Snippet 5.1:line 19, Code Snippet 5.3:line 4) plays a crucial role. These asynchronous programming techniques enable the immediate loading of critical assets while deferring the execution of non-critical JavaScript code. By prioritizing essential elements, this method improves the application's loading efficiency and responsiveness, ensuring a smoother interaction for end-users. [61]

5.1.2 HTML optimization

A vital component of the optimization process is HTML optimization, which encompasses various strategies to enhance both efficiency and readability.

Firstly, akin to JavaScript code, the HTML code must be clean and concise. This not only promotes more readable code but also lessens the amount of data to be loaded, thereby enhancing the application's efficiency.

To achieve conciseness, inline styles were eschewed. Instead, links to stylesheets were incorporated into the document's `<head>` section. Similarly, inline scripts were avoided as much as possible, with sources to the scripts appended to the `<script>` tags (Code Snippet 5.4). This alignment with best practices contributes to a more modular and maintainable code structure. [62]

The sequence in which files are loaded plays a crucial role in web performance. When a browser encounters a `<script>` tag, its default behavior is to halt the parsing of HTML and proceed to download, parse, and execute the script. If the CSS `<link>` tag has not been processed by that point, the browser is unable to download the file until the JavaScript has been processed. [62] This can lead to delays in rendering and adversely impact user experience. As a result, to optimize loading efficiency, it is generally advisable to place CSS file links before script tags in the HTML document. This ordering ensures that styling information is available early in the page rendering process, allowing for a smoother and more visually coherent loading sequence.

Additionally, the reduction of blank lines, spaces, and unnecessary indentation was pursued. While this may impair readability to some extent, the resultant increase in efficiency was deemed a higher priority, striking a balance that serves the application's overall performance goals.

An attempt was also made to minimize external HTTP requests, excluding any features that did not enhance the user experience. This reduction in requests can lead to quicker load times, providing a more seamless interface for users.

Lastly, the implementation of prefetching further elevated the browsing experience (Code Snippet 5.4). By fetching necessary resources and related data in advance of need, prefetching ensures that users can navigate between pages with minimal loading times. This predictive strategy thus contributes to a more responsive and engaging user experience. [63]

1 `<script rel="prefetch" src="js/homeCharts.js"></script>`

CODE SNIPPET 5.4: example of prefetching on a script

5.1.3 Further optimization

As outlined in the preceding section, efforts were undertaken to optimize the code. Yet, additional measures could potentially be implemented to further enhance the speed and efficiency of the application. This section attempt to give some ideas for further optimization.

Certain functions, such as the POST handler for the structure-based explore search (Code Snippet 5.5), remain relatively large and address multiple concerns. Consequently, these could be subdivided into smaller components to enhance both readability and testability. More input validation could also be added to improve security and robustness. This concept is connected to the error-handling issues discussed in section 4.2.1 and could lead to increased responsiveness and prevention of application failure in the event of errors.

Finally, minimizing DOM¹ manipulation can be beneficial, as frequent access and updates to the DOM are computationally intensive. Adopting this strategy could enhance the application's overall performance.

¹A programming interface that represents an HTML document's structure as a tree of nodes, enabling developers to interact with and modify the content, structure, and style of web pages using languages such as JavaScript [64]

```

1 // Define a router for the '/explore/structure' path which handles all
  types of HTTP requests
2 router.all('/explore/structure', async (req, res) => {
3   try {
4     // Get the table columns
5     const columns = await getTableColumns();
6     let display = 'table';
7     let datasource = 'lotus';
8
9     // Check if the HTTP request is a POST request
10    if (req.method === 'POST') {
11      // Extract information from the request body
12      const smiles = req.body.smiles;
13      display = req.body.display;
14      const max = req.body.maxNum;
15      const group = req.body.taxo;
16      const activeTab = req.body.activeTab;
17      const radio = req.body.radio;
18      const tanimoto = req.body.tanimoto / 100 ;
19      datasource = req.body.data_source;
20
21      let tabHandler
22
23      if (datasource === 'lotus'){
24        // Get the appropriate handler function for the active tab
25        tabHandler = tabHandlers[activeTab][0];
26      } else if (datasource === 'dbgi'){
27        tabHandler = tabHandlers[activeTab][1];
28      } else {
29        res.send('Unknown source of Data');
30      }
31
32      if (tabHandler) {
33        // If the handler function exists, call it and get the results
34        const results = await tabHandler(radio, display, smiles, group, max
, tanimoto) ;
35
36        // Based on the 'display' value, render the response
37        if (display === 'table'){
38          if (datasource === 'lotus'){
39            res.render('exploreStructure', { columns, results: results.
result.rows, hits: results.result.rows.length , display: display,
source: datasource});
40          } else if (datasource === 'dbgi') {
41            res.render('exploreStructure', {results: results.results,
headers: results.headers, hits: 0 , display:display, source: datasource
});
42          }
43          } else if (display === 'graph'){
44            hits = results.totalCount;
45            res.render('exploreStructure', { columns, results: results.result
, hits: hits , display: display, source: datasource});
46          }
47          } else {
48            // If the handler function does not exist, send an error message
49            res.send('Unknown tab');
50          }
51        } else {
52          // If the request is not a POST request, render the default '
exploreStructure' page
53          res.render('exploreStructure', { columns, hits: 0 , display: display,
source: datasource});

```

```

54     }
55   } catch (err) {
56     // If there's any error, log it and send an error message
57     console.error(err);
58     res.send('Oops... something went wrong!');
59   }
60 });

```

CODE SNIPPET 5.5: Handling of HTTP Requests on the structure-based *explore* page:

A multi-faceted function that could benefit from further subdivision for code optimization.

5.2 Challenges of the study and potential avenues for future re-search

Reflecting on the past six months, this study faced certain limitations that slowed down certain stages of the project. The limitations mainly revolved around the handling of both LOTUS and DBGI data sources and selection of programming languages.

5.2.1 Challenges caused by a dual data sources

Beginning with the LOTUS data offered an insightful and easier foundation into data management thanks to its static nature. However, there is a consideration to be made that starting with a specific subset of the DBGI data might have streamlined the process -keeping in mind that the first goal was to implement a DBGI dashboard-. Both the LOTUS and DBGI datasets, although broadly similar, exhibited differences in their structures, particularly in column names(Figure 5.1). These structural disparities added complexity when transitioning between the two data sources, resulting in a more resource-intensive procedure.

Furthermore, storage differences between the LOTUS and DBGI data added difficulties. With LOTUS stored in a relational database and DBGI in a Knowledge graph (Figure 5.1), variations emerged in query languages and the modules needed for data extraction and manipulation. This disparity increased the intricacies of transitioning between the two sources.

Due to the combined structural and procedural differences and the existing time constraints, there was a limitation in seamlessly integrating all LOTUS functionalities into the DBGI system. However, it is noteworthy that possessing implementations for both data sources and query types presents considerable advantages and opportunities. Rather than viewing this as a mere consumption of time and energy, it should be seen as a strategic advantage, expanding the potential for diverse data management and extraction techniques.

5.2.2 Choice of Software and Programming Language

Selecting the appropriate software and programming language for a project can dictate its efficiency and outcome. A significant challenge faced during this study was the transition from Python to JavaScript. However, while the initial decision to adopt JavaScript appeared justified, not all options were considered and other software and programming languages might present distinct benefits.

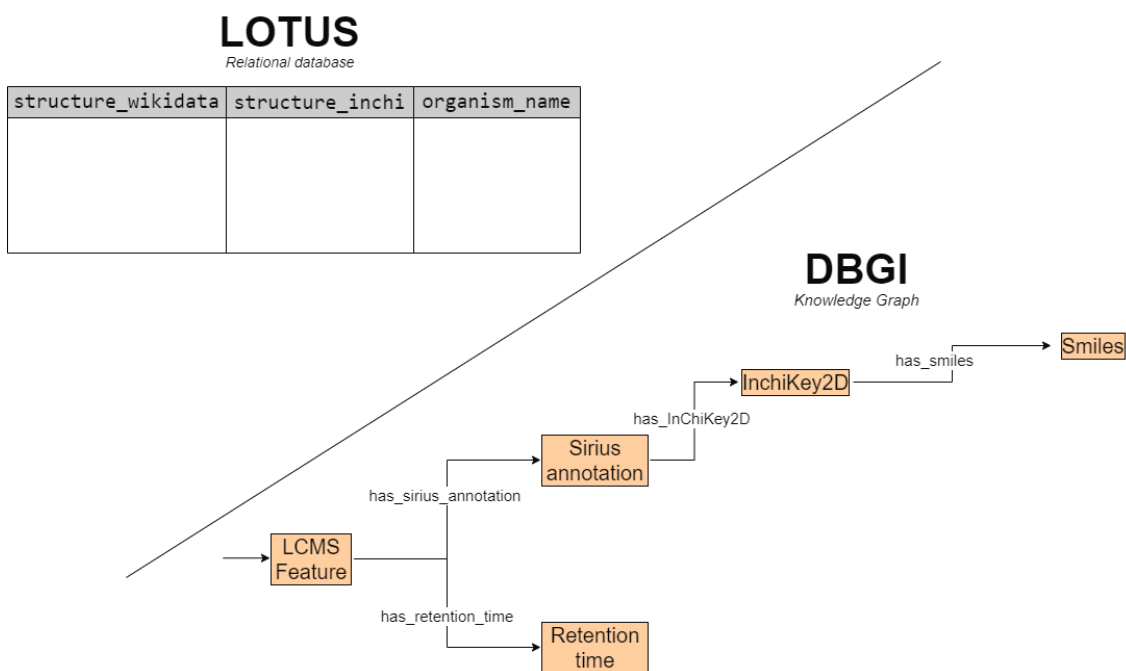


FIGURE 5.1: Data Structure of both data sources:
The LOTUS database uses a tabular format, with each row symbolizing a sample. On the other hand, the DBGI data is arranged in a Knowledge Graph, where entities (arrow starts) connect to their values (arrow ends) via specific attributes (arrows). It can also be seen that column names vary between sources; 'structure_inchi' in LOTUS becomes 'InchiKey2D' in DBGI.

For instance, during the course of the project, the Nprod.net's *LOTUS* dashboard[24] was unveiled. It was developed using *Streamlit*, a library that could have potentially streamlined the development process, enabling a feature-rich dashboard in a shorter time span.

Future researchers or developers entrusted with this project should critically evaluate the choice of node.js/JS. They may decide to retain it or transition to another platform or language that could provide superior advantages or align better with evolving project requirements.

To conclude, a potentially more effective approach may have been to initially aim for a more modest scope, focusing on a single-page dashboard with fewer but fully implemented features. In fact, the resulting dashboard was marked by the presence of numerous features, many of which still require improvement, and some that do not fully work. A more focused strategy that emphasized complete implementation of one feature at a time, using both data sources, before moving on to another feature, might have alleviated these issues. Instead of pivoting to a new feature at the first sign of difficulty, persisting with each feature until it was fully operational could have led to a dashboard with a more polished and finished appearance.

5.3 Conclusion

The primary objective of designing a dashboard for the Digital Botanical Gardens Initiative was to provide a clear visualization of the project's progression and to effectively display data from both LOTUS and DGBI. The development journey, however, encountered challenges, especially given the contrasting data structures and content between these two sources. The envisaged dashboard was meant to span multiple pages, but its execution does yet not entirely reflect the depth and potential of the data from both datasets. For example, it still lacks the link to the LCMS data.

Retrospectively, a more thorough preliminary planning phase might have better steered the development trajectory, ensuring a comprehensive overview and a clearer roadmap. Instead of immersing directly into the project, a structured planning process would have facilitated smoother transitions and greater alignment of data sources and dashboard functionalities.

However, it is essential to acknowledge that the effort and knowledge invested in this project have laid the foundation for ongoing improvements and advancements. Even though the continuation and expansion of this work may pose challenges during transition, especially given its complexity, it offers a fertile ground for exploration and innovation.

Future endeavors might require revisiting foundational decisions. It might be of interest for subsequent teams or individuals to critically assess the software and programming languages utilized in this project. They might find merit in opting for an entirely fresh approach, leveraging alternative tools and methodologies that best serve the evolving needs of the Digital Botanical Gardens Initiative.

Appendix A

Softwares & libraries version

A.1 Nodejs

Version 20.5.0 [65]

- @innotrade/enapso-graphdb-client v1.9.0 [66]
- axios v1.4.0 [67]
- d3 v7.8.4 [35]
- dotenv v16.0.3 [68]
- ejs v3.1.9 [69]
- express v4.18.2 [32]
- favicon v0.0.2 [70]
- graphdb v3.0.0 [71]
- jsme-editor v2022.9.26 [34]
- json2csv v6.0.0-alpha.2 [72]
- jsonfile v6.1.0 [73]
- node-cache v5.1.2 [74]
- node-fetch v2.6.12 [75]
- nodemon v2.0.22 [76]
- pg v8.10.0 [39]
- svg v0.1.0 [77]

A.2 Python

Version 3.10.9

- RDKit v2023.03.1 [46]
- sys v3.10.9

A.3 PostgreSQL

Version 12.15

Appendix B

Dashboard pages



FIGURE B.1: Home page

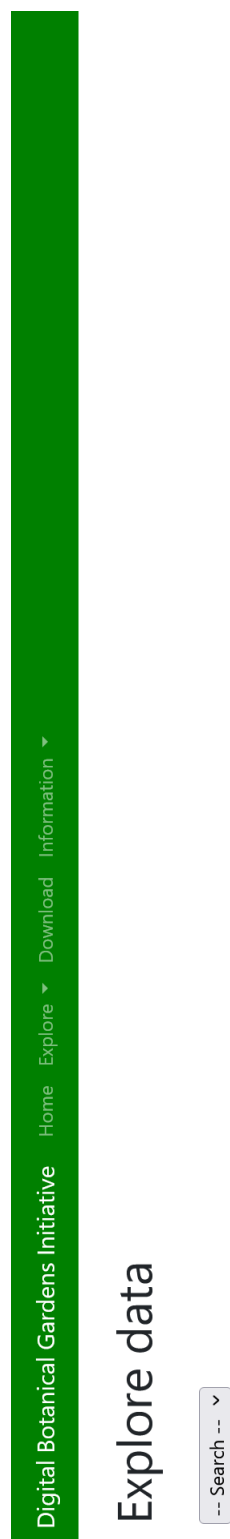


FIGURE B.2: Explore page

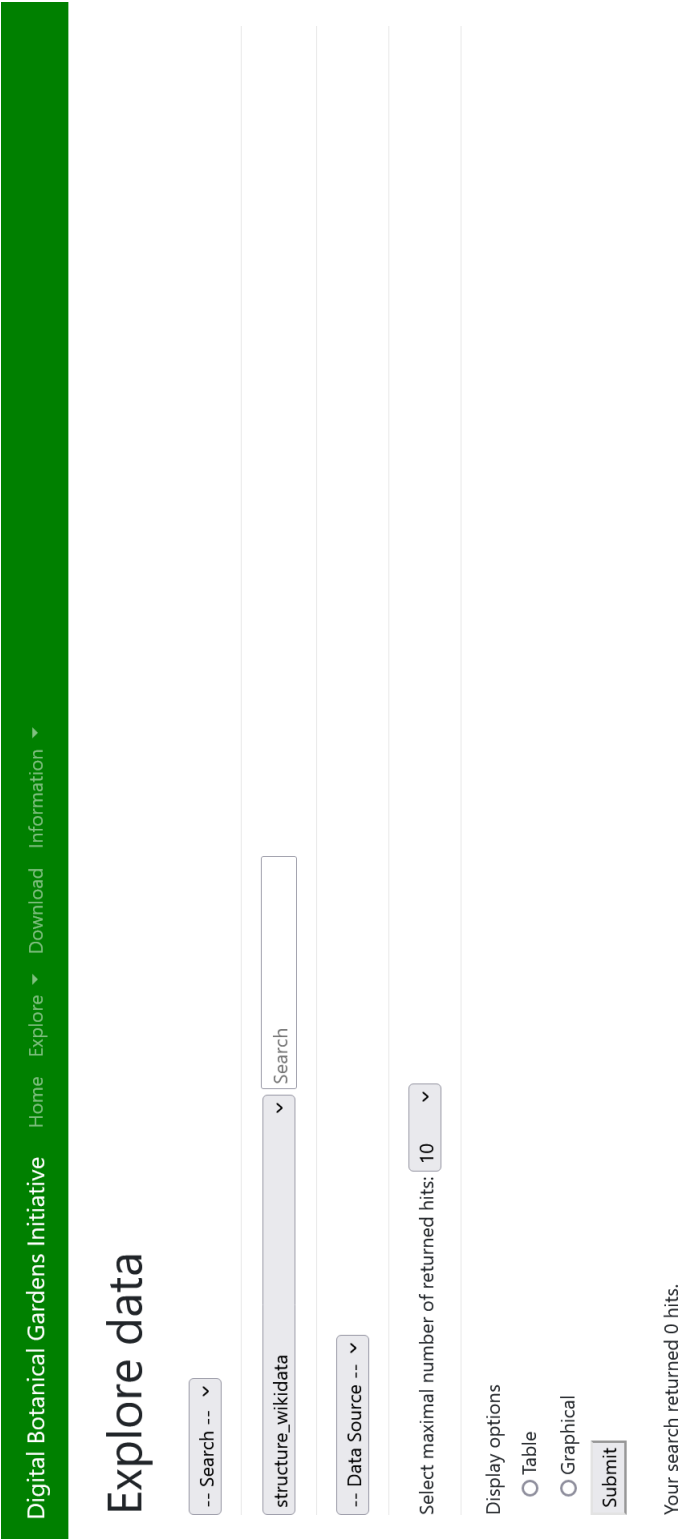
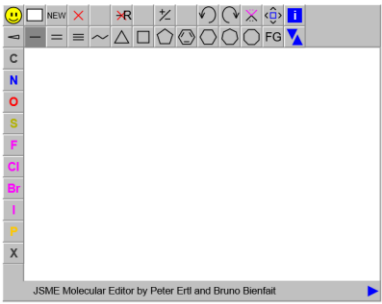


FIGURE B.3: Explore page in text-based search mode

Digital Botanical Gardens Initiative
Home
Explore
Download
Information

Explore data

-- Search --



Exact match
Substructure search
Similarity search

Exact match type: using InChI (recommended) or SMILES

☐ Exact match (by InChI)
☐ Exact match (by canonical SMILES)

-- Data Source --

(Optional) Structure search only in a taxonomic group:

Select maximal number of returned hits:

Display options

☐ Table
☐ Graphical

Search

Your search returned 0 hits.

FIGURE B.4: Explore page in structure-based exact match search mode

Digital Botanical Gardens Initiative
Home
Explore
Download
Information

Explore data

-- Search --

NEW
X
R
Z
U
V
W
X
Y
Z
FG

C
N
O
S
F
Cl
Br
I
X

JSME Molecular Editor by Peter Ertl and Bruno Bienfait

Exact match
Substructure search
Similarity search

Substructure matching algorithm

Default substructure search (Ullmann algorithm)

-- Data Source --

(Optional) Structure search only in a taxonomic group:

Select maximal number of returned hits: 10

Display options

☐ Table
☐ Graphical

Search

Your search returned 0 hits.

FIGURE B.5: Explore page in structure-based substructure search mode

FIGURE B.6: Explore page in structure-based similarity search mode

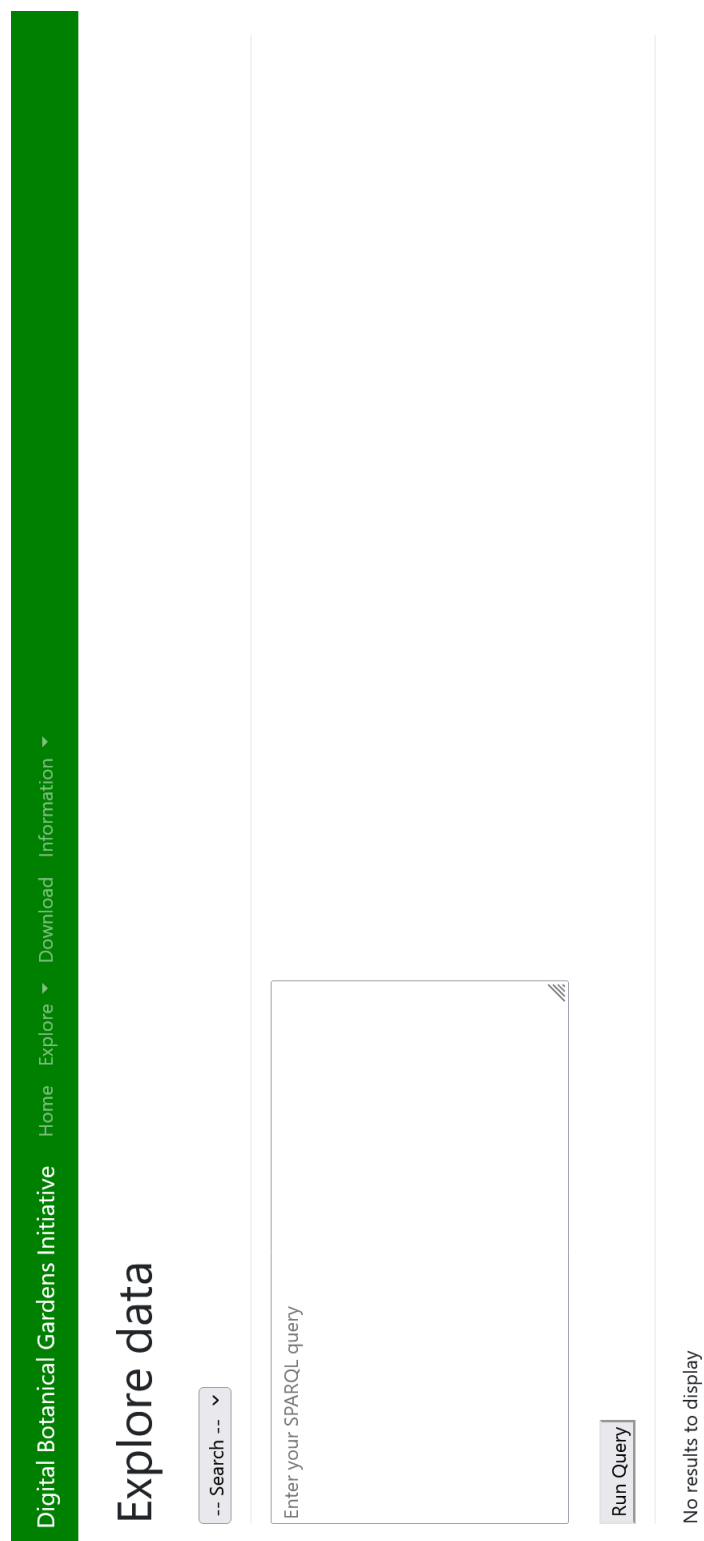


FIGURE B.7: Explore page in SPARQL-based search mode

Digital Botanical Gardens Initiative

HomeExploreDownloadInformation

Chemical compounds list

No results

Most Found:

- Beta-Sitosterol
- Stigmasterol
- Quercetin
- Limonene
- Hexadecanoate;hydron
- Palmitic Acid

- alpha-TERPINEOL
- Gallic Acid
- Kaempferol
- 4-Carvomenthenol
- Luteolin
- Camphene

FIGURE B.8: Compound list page

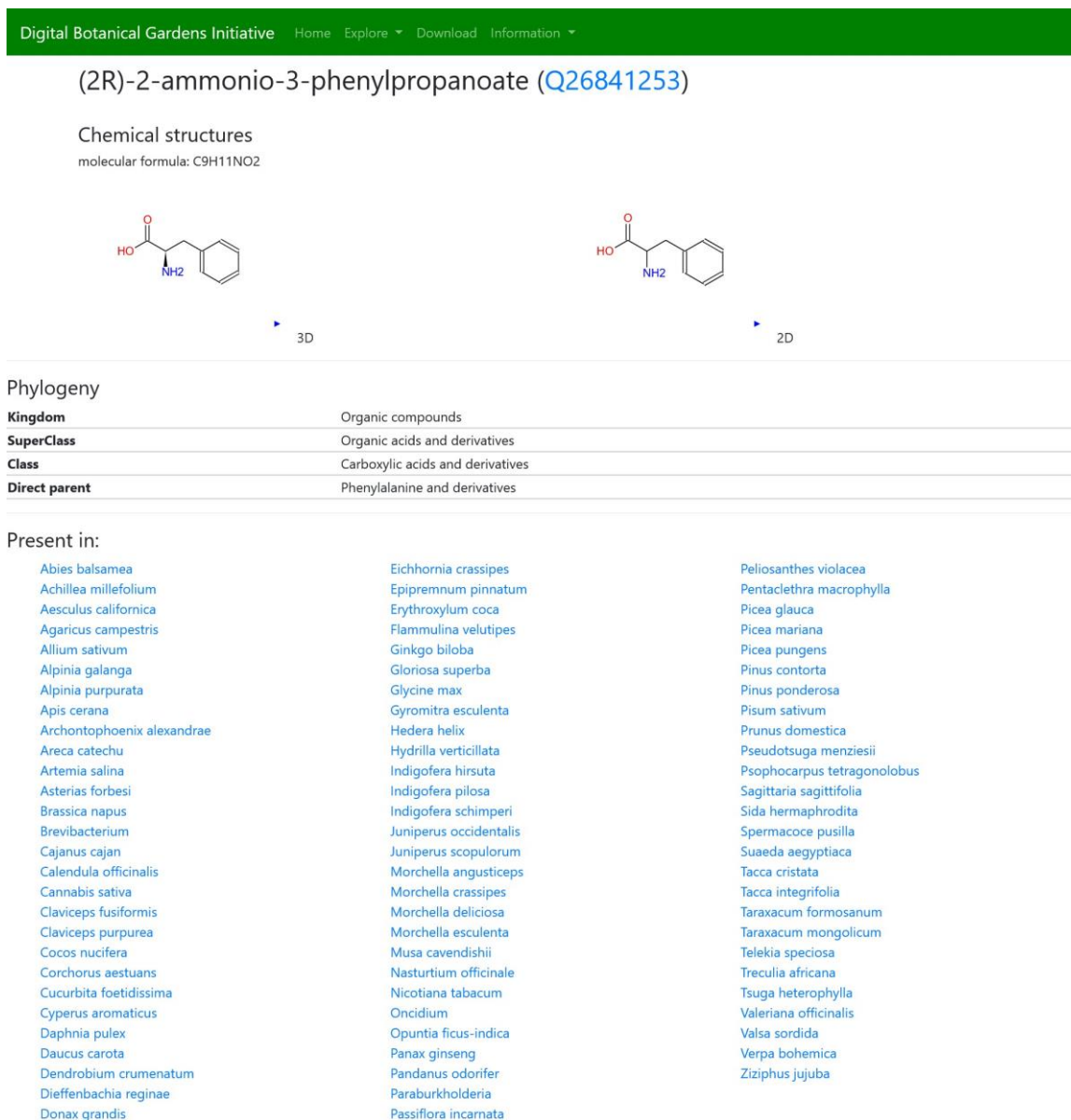


FIGURE B.9: Molecule page example



FIGURE B.10: Organisms list page

Digital Botanical Gardens Initiative Home Explore Download Information

Ajuga reptans (Q157249)



Taxonomy

Domain	Eukaryota
Kingdom	Archaeplastida
Phylum	Streptophyta
Class	Magnoliopsida
Order	Lamiales
Family	Lamiaceae
Tribe	
Genus	Ajuga
Species	Ajuga reptans
Varietas	

Chemical Compounds:

Cholesterol

Codisterol

Clerosterol

CID 15608667

22,23-Dihydrobrassicasterol

20-Hydroxyecdysone

6-Epi-8-O-acetylharpagide

Ajugareptone

6-Epiharpagide

Ajugareptansone A

Covi-ox

[(2R,3S,4S,5S,6S)-6-[(2S,3S,4S,5S,6S)-4,5-dihydroxy-6-[(2R,3S,4S,5R,6S)-2-[(2S,3S,4R,5R,6S)-2-[2-(3,4-dihydrox...

4-Hydroxycinnamic acid

Vitamin E

(2R,3S,4R,5R,6S)-2-[(2S,3S,4R,5R,6S)-2-[2-(3,4-dihydrox...

Kuromanin

Cyanidin 3-sophoroside 5-glucoside

Cyanidin 3-sophoroside-5-glucoside

Cianidanol

(2S,3R,4S,5S,6R)-2-[(2S,3R,4S,5S,6S)-4,5-dihydroxy-6-(h...

3-[[[(2R,3R,4S,5R,6S)-6-[3-[(2S,3R,4S,5R,6R)-4,5-dihydro...

3-[[[(2S,3R,4S,5S,6S)-6-[3-[(2S,3R,4S,5R,6S)-4,5-dihydrox...

Polypodineb

29-Norsengosterone

29-Norcysterone

Ajugasteron B

Ajugareptansin

Harpagide

[(1R,2S,3R,4aR,5S,6R,8S,8aR)-5-[(3aR,5R,6aS)-3a,4,5,6a-...

[(1R,3S,4R,4aR,5S,7R,8S,8aR)-8-[(3aR,5S,6aS)-2,3,3a,4,5...

Ajugachin A

[4a-Hydroxy-7-methyl-1-[3,4,5-trihydroxy-6-(hydroxy...

Polypodine b

Ajugalactone

(1S,4aR,5R,7S,7aR)-7-methyl-1-[(2R,3S,4R,5S,6S)-3,4,5-t...

Ajubractin D

[(1R,2S,3R,4S,4aR,5S,6R,8S,8aR)-5-[(3aR,5S,6aS)-2,3,3a...

Areptin B

[(1R,3S,4R,4aR,5S,7R,8S,8aR)-8-[(3aS,5S,6aR)-3a,4,5,6a-...

[(1S,4aR,5R,7S,7aS)-4a,5-dihydroxy-7-methyl-1-[(2R,3S...

Ajugorientin

(3S,4S,5R)-4-[(2R,3R)-2,3-dihydroxy-3-[(2S,3R,5S,9R,10...

Reptoside

[(4aS,7S,7aR)-4a,5-dihydroxy-7-methyl-1-[3,4,5-trihydr...

(2S,3R,4S,5S,6R)-2-[[[(1S,3R,5R,7S,7aS)-5,7-dihydroxy-3-...

20-Hydroxyecdysone 3-acetate

(1S)-1alpha-(beta-D-Glucopyranosyloxy)-3beta-metho...

(15S)-12,19-Bis(beta-D-glucopyranosyloxy)abieta-8,11,...

(3R,5R,9R,10R,13R,14R,17S)-17-[(1R)-1-[(2R)-4-ethyl-5-...

[(1S,4aS,5S,7S,7aS)-4a,5-dihydroxy-7-methyl-1-[(2S,3R...

[(1S,4aS,5R,7S,7aS)-4a,5-dihydroxy-7-methyl-1-[(2S,3R...

(3S,4S)-4-[(2R,3R)-2,3-dihydroxy-3-[(2S,3R,5R,9R,10R,1...

4-[2,3-dihydroxy-3-(2,3,14-trihydroxy-10,13-dimethyl-...

28-epi-Sengosterone

(3R,4S,6R)-4-(2-hydroxyethyl)-6-[(1R)-1-hydroxy-1-[(2S...

(2S,5R,9R,10R,13R,14R,17S)-17-[(1R)-1-[(2R)-4-ethyl-5-...

Cyasterone

(2S,3S,5R,9R,10R,13R,14R,17S)-17-[(1R)-1-[(2R)-4-ethyl...

3alpha-Hydroxyecdysone

(3S,8S,9S,10R,13R,14S,17R)-17-[(2S,3E,5S)-5-ethyl-6-m...

(3S,4S)-4-[(2R,3R)-2,3-dihydroxy-3-[(2S,3S,5S,9R,10R,1...

(3S,4S)-4-[(3R)-3-hydroxy-2-oxo-3-[(2S,3R,5R,9R,10R,1...

20-Hydroxyecdysone 22-acetate

Viticosterone E

(2R,3R,5S,9R,10R,13R,14S,17S)-2,3,5,14-tetrahydroxy-1...

(3S,4S)-4-[(2R,3R)-2,3-dihydroxy-3-[(2S,3S,5R,9R,10R,1...

(3S,4S,5R)-4-[(2R,3R)-2,3-dihydroxy-3-[(2R,3S,5R,9R,10...

(3S,4S,5R)-4-[(3R)-3-hydroxy-2-oxo-3-[(2S,3R,5R,9R,10...

(3S,4S)-4-[(3R)-3-hydroxy-2-oxo-3-[(2S,3R,5S,9R,10R,1...

(3S,4S,5R)-4-[(3R)-3-hydroxy-2-oxo-3-[(2S,3R,5S,9R,10...

Ecdysone

[(1R,3S,4R,4aR,5S,7R,8S,8aR)-8-[(3aS,5R,6aS)-3a,4,5,6a-...

[(1S)-2-[(1S,2R,4S,4aR,5R,6R,8aR)-4-acetyloxy-4a-(acety...

[(1R,3S,4R,4aR,5S,7R,8S,8aR)-8-[(3aR,5R,6aS)-2,3,3a,4,5...

[(1R,3S,4R,4aR,5S,7R,8S,8aR)-8-[(3aS,5S,6aR)-3a,4,5,6a-...

3-[[[6-[3-[4,5-Dihydroxy-6-[3-(4-hydroxyphenyl)prop-2-...

(3S,4S,5R)-5-[(1R,2R)-1,2-dihydroxy-2-[(2S,3R,5R,9R,10...

Breviflorasterone

Delphinidin 3-sophoroside-5-glucoside

3-[[[6-[3-[4,5-Dihydroxy-6-[3-(4-hydroxyphenyl)prop-2-...

(2S,3R,5R,9R,10R,13R,14S,17S)-17-[(1R)-1-[(2R,4R,5S,6S...

(3S,4R,5S)-5-[(1R,2R)-1,2-dihydroxy-2-[(2S,3R,5R,9R,10...

(2R)-4-[(1R)-1-hydroxyethyl]-2-[(1R)-1-hydroxy-1-[(2S...

2-Hydroxyecdysone

Isocyasterone

Ajugareptansone B

[(1R,2R,4aR,5S,6R,8S,8aR)-8-acetyloxy-8a-(acetyloxyme...

[(1S,4R,4aR,5S,7R,8S,8aR)-8-[(3aR,5S,6aS)-2,3,3a,4,5,6a-...

[(1S,3S,4R,4aR,5S,7R,8S,8aR)-8-[(3aS,5S,6aR)-2,3,3a,4,5...

FIGURE B.11: Organism page example



FIGURE B.12: Download page

Bibliography

- [1] The Digital Botanical Gardens Initiative Consortium. *The Digital Botanical Gardens Initiative*. en-US. Tech. rep. Publication Title: Manubot. Manubot, Dec. 2022. URL: <https://digital-botanical-gardens-initiative.github.io/dbgi-green-paper/> (visited on 07/03/2023).
- [2] Fraser D. M. Smith et al. “How much do we know about the current extinction rate?” In: *Trends in Ecology & Evolution* 8.10 (Oct. 1993), pp. 375–378. ISSN: 0169-5347. DOI: [10.1016/0169-5347\(93\)90223-C](https://doi.org/10.1016/0169-5347(93)90223-C). URL: <https://www.sciencedirect.com/science/article/pii/016953479390223C> (visited on 08/31/2023).
- [3] M. Bakkenes et al. “Assessing effects of forecasted climate change on the diversity and distribution of European higher plants for 2050”. en. In: *Global Change Biology* 8.4 (2002). _eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1046/j.1354-1013.2001.00467.x>, pp. 390–407. ISSN: 1365-2486. DOI: [10.1046/j.1354-1013.2001.00467.x](https://doi.org/10.1046/j.1354-1013.2001.00467.x). URL: <https://onlinelibrary.wiley.com/doi/abs/10.1046/j.1354-1013.2001.00467.x> (visited on 08/05/2023).
- [4] Céline Bellard et al. “Impacts of climate change on the future of biodiversity”. en. In: *Ecology Letters* 15.4 (2012). _eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1461-0248.2011.01736.x>, pp. 365–377. ISSN: 1461-0248. DOI: [10.1111/j.1461-0248.2011.01736.x](https://doi.org/10.1111/j.1461-0248.2011.01736.x). URL: <https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1461-0248.2011.01736.x> (visited on 08/05/2023).
- [5] *Metabolome*. en. Page Version ID: 1160225240. June 2023. URL: <https://en.wikipedia.org/w/index.php?title=Metabolome&oldid=1160225240> (visited on 08/02/2023).
- [6] Christian Marchese. “Biodiversity hotspots: A shortcut for a more complicated concept”. en. In: *Global Ecology and Conservation* 3 (Jan. 2015), pp. 297–309. ISSN: 2351-9894. DOI: [10.1016/j.gecco.2014.12.008](https://doi.org/10.1016/j.gecco.2014.12.008). URL: <https://www.sciencedirect.com/science/article/pii/S235198941400095X> (visited on 08/09/2023).
- [7] Mike Maunder et al. “Plant Conservation in the Caribbean Island Biodiversity Hotspot”. en. In: *The Botanical Review* 74.1 (Mar. 2008), pp. 197–207. ISSN: 1874-9372. DOI: [10.1007/s12229-008-9007-7](https://doi.org/10.1007/s12229-008-9007-7). URL: <https://doi.org/10.1007/s12229-008-9007-7> (visited on 08/09/2023).
- [8] *A propos | Jardin botanique de l'Université de Fribourg |*. URL: <https://www.unifr.ch/jardin-botanique/fr/about/> (visited on 08/09/2023).
- [9] Adriano Rutz et al. “The LOTUS initiative for open knowledge management in natural products research”. In: *eLife* 11 (May 2022). Ed. by David A Donoso, Anna Akhmanova, and Charles Tapley Hoyt. Publisher: eLife Sciences Publications, Ltd, e70780. ISSN: 2050-084X. DOI: [10.7554/eLife.70780](https://doi.org/10.7554/eLife.70780). URL: <https://doi.org/10.7554/eLife.70780> (visited on 08/03/2023).
- [10] *Zakodium*. URL: <https://www.zakodium.com/zakodium> (visited on 08/24/2023).

- [11] Daniel Petras et al. "GNPS Dashboard: collaborative exploration of mass spectrometry data in the web browser". en. In: *Nature Methods* 19.2 (Feb. 2022). Number: 2 Publisher: Nature Publishing Group, pp. 134–136. ISSN: 1548-7105. DOI: [10.1038/s41592-021-01339-5](https://doi.org/10.1038/s41592-021-01339-5). URL: <https://www.nature.com/articles/s41592-021-01339-5> (visited on 07/25/2023).
- [12] Hannes L. Röst et al. "OpenMS: a flexible open-source software platform for mass spectrometry data analysis". eng. In: *Nature Methods* 13.9 (Aug. 2016), pp. 741–748. ISSN: 1548-7105. DOI: [10.1038/nmeth.3959](https://doi.org/10.1038/nmeth.3959).
- [13] Kenneth Haug et al. "MetaboLights: a resource evolving in response to the needs of its scientific community". eng. In: *Nucleic Acids Research* 48.D1 (Jan. 2020), pp. D440–D444. ISSN: 1362-4962. DOI: [10.1093/nar/gkz1019](https://doi.org/10.1093/nar/gkz1019).
- [14] Juan A. Vizcaíno et al. "ProteomeXchange provides globally coordinated proteomics data submission and dissemination". eng. In: *Nature Biotechnology* 32.3 (Mar. 2014), pp. 223–226. ISSN: 1546-1696. DOI: [10.1038/nbt.2839](https://doi.org/10.1038/nbt.2839).
- [15] Manish Sud et al. "Metabolomics Workbench: An international repository for metabolomics data and metadata, metabolite standards, protocols, tutorials and training, and analysis tools". eng. In: *Nucleic Acids Research* 44.D1 (Jan. 2016), pp. D463–470. ISSN: 1362-4962. DOI: [10.1093/nar/gkv1042](https://doi.org/10.1093/nar/gkv1042).
- [16] Plotly Technologies Inc. *Collaborative data science*. Place: Montreal, QC Publisher: Plotly Technologies Inc. 2015. URL: <https://plot.ly>.
- [17] *Wikidata:Introduction - Wikidata*. URL: <https://www.wikidata.org/wiki/Wikidata:Introduction> (visited on 08/02/2023).
- [18] *Resource Description Framework*. en. Page Version ID: 1161223823. June 2023. URL: https://en.wikipedia.org/w/index.php?title=Resource_Description_Framework&oldid=1161223823 (visited on 08/02/2023).
- [19] *Sugar Removal Service*. URL: <https://sugar.naturalproducts.net/> (visited on 08/01/2023).
- [20] *COCONUT: Natural Products Online*. URL: <https://coconut.naturalproducts.net/> (visited on 08/01/2023).
- [21] Peter Ertl, Silvio Roggo, and Ansgar Schuffenhauer. "Natural Product-likeness Score and Its Application for Prioritization of Compound Libraries". In: *Journal of Chemical Information and Modeling* 48.1 (Jan. 2008). Publisher: American Chemical Society, pp. 68–74. ISSN: 1549-9596. DOI: [10.1021/ci700286x](https://doi.org/10.1021/ci700286x). URL: <https://doi.org/10.1021/ci700286x> (visited on 08/24/2023).
- [22] *Kotlin Introduction*. en-US. URL: https://www.w3schools.com/kotlin/kotlin_intro.php (visited on 08/02/2023).
- [23] *Programming languages: Kotlin rises fastest but JavaScript lures millions more developers*. en. URL: <https://www.zdnet.com/article/programming-languages-javascript-now-used-by-12-million-developers-but-kotlin-rises-fastest/> (visited on 08/02/2023).
- [24] *LOTUS*. URL: <https://search.nprod.net/> (visited on 08/03/2023).
- [25] *Streamlit Docs*. URL: <https://docs.streamlit.io/> (visited on 08/03/2023).
- [26] *Directus: The Modern Data Stack, Democratized*. en. URL: <https://directus.io/> (visited on 08/03/2023).
- [27] *methodology*. Feb. 2022. URL: <https://digital-botanical-gardens-initiative.github.io/dendron-dbg/notes/mpboa8wpvhpygf5fw4o9v11/> (visited on 08/03/2023).

- [28] Pierre-Marie Allard. *Earth Metabolome Initiative semantic model workshop intro - What is the stuff we are talking about ?* eng. July 2023. DOI: [10.5281/zenodo.8137605](https://doi.org/10.5281/zenodo.8137605). URL: <https://zenodo.org/record/8137605> (visited on 07/14/2023).
- [29] Michael Heron, Vicki L. Hanson, and Ian Ricketts. "Open source and accessibility: advantages and limitations". In: *Journal of Interaction Science* 1.1 (May 2013), p. 2. ISSN: 2194-0827. DOI: [10.1186/2194-0827-1-2](https://doi.org/10.1186/2194-0827-1-2). URL: <https://doi.org/10.1186/2194-0827-1-2> (visited on 07/21/2023).
- [30] Janith Gamage. *Single-Threaded and Asynchronous — How Does Node Do It?* en. Dec. 2020. URL: <https://betterprogramming.pub/single-threaded-and-asynchronous-how-does-node-do-it-d964100766a> (visited on 08/28/2023).
- [31] *Node.js Introduction*. en-US. URL: https://www.w3schools.com/nodejs/nodejs_intro.asp (visited on 08/24/2023).
- [32] *Express JS Tutorial: What is Express in Node JS?* en-US. URL: <https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-express-js> (visited on 08/03/2023).
- [33] *Building and structuring a Node.js MVC application - LogRocket Blog*. URL: <https://blog.logrocket.com/building-structuring-node-js-mvc-application/> (visited on 08/03/2023).
- [34] Bruno Bienfait and Peter Ertl. "JSME: a free molecule editor in JavaScript". In: *Journal of Cheminformatics* 5.1 (May 2013), p. 24. ISSN: 1758-2946. DOI: [10.1186/1758-2946-5-24](https://doi.org/10.1186/1758-2946-5-24). URL: <https://doi.org/10.1186/1758-2946-5-24> (visited on 08/03/2023).
- [35] *d3*. URL: <https://github.com/d3/d3> (visited on 08/03/2023).
- [36] *stackgl*. en. URL: <https://github.com/stackgl> (visited on 08/03/2023).
- [37] *Introduction to HTML*. en-US. URL: https://www.w3schools.com/html/html_intro.asp (visited on 08/03/2023).
- [38] *CSS Introduction*. en-US. URL: https://www.w3schools.com/Css/css_intro.asp (visited on 07/17/2023).
- [39] Brian C. *node-postgres*. original-date: 2010-10-15T23:05:50Z. Aug. 2023. URL: <https://github.com/brianc/node-postgres> (visited on 08/03/2023).
- [40] *SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems / DigitalOcean*. en. URL: <https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems> (visited on 07/17/2023).
- [41] *SQL Introduction*. en-US. URL: https://www.w3schools.com/sql/sql_intro.asp (visited on 07/21/2023).
- [42] *About GraphDB — GraphDB 10.2.3 documentation*. URL: <https://graphdb.ontotext.com/documentation/10.2/about-graphdb.html> (visited on 07/21/2023).
- [43] Mohit Mayank. *A guide to the Knowledge Graphs*. en. Sept. 2021. URL: <https://towardsdatascience.com/a-guide-to-the-knowledge-graphs-bfb5c40272f1> (visited on 07/13/2023).
- [44] *What is a NoSQL Graph Database?* en-US. URL: <https://www.ontotext.com/knowledgehub/fundamentals/nosql-graph-database/> (visited on 08/03/2023).
- [45] *What is SPARQL?* en-US. URL: <https://www.ontotext.com/knowledgehub/fundamentals/what-is-sparql/> (visited on 07/17/2023).
- [46] Greg Landrum et al. *rdkit/rdkit: 2023_03_1 (Q1 2023) Release*. Apr. 2023. DOI: [10.5281/zenodo.7880616](https://doi.org/10.5281/zenodo.7880616). URL: <https://doi.org/10.5281/zenodo.7880616>.

- [47] Olaf Bánki et al. *Catalogue of Life Checklist*. ISSN: 2405-8858 Place: Leiden, Netherlands Version Number: 2023-07-18. July 2023. DOI: [10.48580/dfsy](https://doi.org/10.48580/dfsy). URL: <https://www.checklistbank.org/dataset/9916>.
- [48] *Biota / COL*. ISSN: 2405-8858. URL: <https://www.catalogueoflife.org/data/taxon/5T6MX> (visited on 07/31/2023).
- [49] *5.8: Line Notation (SMILES and InChI)*. en. Aug. 2020. URL: [https://chem.libretexts.org/Courses/Fordham_University/Chem1102%3A_Drug_Discovery_-_From_the_Laboratory_to_the_Clinic/05%3A_Organic_Molecules/5.08%3A_Line_Notation_\(SMILES_and_InChI\)](https://chem.libretexts.org/Courses/Fordham_University/Chem1102%3A_Drug_Discovery_-_From_the_Laboratory_to_the_Clinic/05%3A_Organic_Molecules/5.08%3A_Line_Notation_(SMILES_and_InChI)) (visited on 08/05/2023).
- [50] *Comparison of InChI to other chemical formats*. en-US. URL: http://inchi.info/inchi_comparison_en.html (visited on 08/05/2023).
- [51] Hans-Christian Ehrlich and Matthias Rarey. "Systematic benchmark of substructure search in molecular graphs - From Ullmann to VF2". In: *Journal of Cheminformatics* 4.1 (July 2012), p. 13. ISSN: 1758-2946. DOI: [10.1186/1758-2946-4-13](https://doi.org/10.1186/1758-2946-4-13). URL: <https://doi.org/10.1186/1758-2946-4-13> (visited on 07/29/2023).
- [52] Miroslav Kratochvíl, Jiří Vondrášek, and Jakub Galgonek. "Sachem: a chemical cartridge for high-performance substructure search". In: *Journal of Cheminformatics* 10.1 (May 2018), p. 27. ISSN: 1758-2946. DOI: [10.1186/s13321-018-0282-y](https://doi.org/10.1186/s13321-018-0282-y). URL: <https://doi.org/10.1186/s13321-018-0282-y> (visited on 08/05/2023).
- [53] *Explaining the Ullmann Algorithm: A Simple Example | Saturn Cloud Blog*. en. Section: blog. July 2023. URL: <https://saturncloud.io/blog/explaining-the-ullmann-algorithm-a-simple-example/> (visited on 07/29/2023).
- [54] Roberto Todeschini et al. "Similarity Coefficients for Binary Chemoinformatics Data: Overview and Extended Comparison Using Simulated and Real Data Sets". In: *Journal of Chemical Information and Modeling* 52.11 (Nov. 2012). Publisher: American Chemical Society, pp. 2884–2901. ISSN: 1549-9596. DOI: [10.1021/ci300261r](https://doi.org/10.1021/ci300261r). URL: <https://doi.org/10.1021/ci300261r> (visited on 07/24/2023).
- [55] Dávid Bajusz, Anita Rácz, and Károly Héberger. "Why is Tanimoto index an appropriate choice for fingerprint-based similarity calculations?" en. In: *Journal of Cheminformatics* 7.1 (Dec. 2015). Number: 1 Publisher: BioMed Central, pp. 1–13. ISSN: 1758-2946. DOI: [10.1186/s13321-015-0069-3](https://doi.org/10.1186/s13321-015-0069-3). URL: <https://jcheminf.biomedcentral.com/articles/10.1186/s13321-015-0069-3> (visited on 08/28/2023).
- [56] *Jaccard index*. en. Page Version ID: 1166842542. July 2023. URL: https://en.wikipedia.org/w/index.php?title=Jaccard_index&oldid=1166842542 (visited on 08/05/2023).
- [57] *Theory: Descriptors, Similarity Measures, and Clustering Schemes*. en-US. URL: <https://girke-lab.github.io/chemminetools-docs/docs/theory/> (visited on 08/05/2023).
- [58] Hyunwoo Kim et al. *NPClassifier: A Deep Neural Network-Based Structural Classification Tool for Natural Products*. en. Aug. 2020. DOI: [10.26434/chemrxiv.12885494.v1](https://doi.org/10.26434/chemrxiv.12885494.v1). URL: <https://chemrxiv.org/engage/chemrxiv/article-details/60c74f58702a9ba8dc18bb6b> (visited on 08/28/2023).
- [59] Yannick Djoumbou Feunang et al. "ClassyFire: automated chemical classification with a comprehensive, computable taxonomy". In: *Journal of Cheminformatics* 8.1 (Nov. 2016), p. 61. ISSN: 1758-2946. DOI: [10.1186/s13321-016-0174-y](https://doi.org/10.1186/s13321-016-0174-y). URL: <https://doi.org/10.1186/s13321-016-0174-y> (visited on 08/28/2023).
- [60] and Bootstrap Mark Otto contributors Jacob Thornton. *Bootstrap*. en. URL: <https://getbootstrap.com/> (visited on 08/03/2023).

- [61] *JavaScript performance optimization - Learn web development / MDN*. en-US. Aug. 2023. URL: <https://developer.mozilla.org/en-US/docs/Learn/Performance/JavaScript> (visited on 08/07/2023).
- [62] *How to Optimize HTML to Boost Web Performance*. en. URL: <https://bluetriangle.com/blog/how-to-optimize-html-to-boost-web-performance> (visited on 08/07/2023).
- [63] *Frontend Optimization - 9 Tips to Improve Web Performance*. en. URL: <https://www.keycdn.com/blog/frontend-optimization> (visited on 08/07/2023).
- [64] *Introduction to the DOM - Web APIs / MDN*. en-US. May 2023. URL: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction (visited on 08/07/2023).
- [65] *Node.js*. URL: <https://nodejs.org/en> (visited on 08/03/2023).
- [66] *ENAPSO Graph Database Client*. original-date: 2018-06-19T09:39:30Z. Nov. 2022. URL: <https://github.com/innotrade/enapso-graphdb-client> (visited on 08/03/2023).
- [67] *axios/axios at main*. URL: <https://github.com/axios/axios/tree/main> (visited on 08/03/2023).
- [68] *Mot. dotenv*. original-date: 2013-07-05T18:25:05Z. Aug. 2023. URL: <https://github.com/motdotla/dotenv> (visited on 08/03/2023).
- [69] Matthew Eernisse. *Embedded JavaScript templates*. original-date: 2014-12-31T17:49:35Z. Aug. 2023. URL: <https://github.com/mde/ejs> (visited on 08/03/2023).
- [70] *Favicons*. original-date: 2014-05-16T03:00:08Z. Aug. 2023. URL: <https://github.com/itgalaxy/favicons> (visited on 08/03/2023).
- [71] *graphdb.js*. original-date: 2019-02-26T14:07:05Z. July 2023. URL: <https://github.com/Ontotext-AD/graphdb.js> (visited on 08/03/2023).
- [72] *json2csv*. URL: <https://mircozeiss.com/json2csv/> (visited on 08/03/2023).
- [73] J. P. Richardson. *Node.js - jsonfile*. original-date: 2012-09-10T19:02:27Z. July 2023. URL: <https://github.com/jprichardson/node-jsonfile> (visited on 08/03/2023).
- [74] *Simple and fast NodeJS internal caching*. original-date: 2011-10-18T14:30:09Z. Aug. 2023. URL: <https://github.com/node-cache/node-cache> (visited on 08/29/2023).
- [75] *node-fetch/node-fetch*. original-date: 2015-01-26T07:29:26Z. Aug. 2023. URL: <https://github.com/node-fetch/node-fetch> (visited on 08/03/2023).
- [76] *nodemon*. URL: <https://nodemon.io/> (visited on 08/03/2023).
- [77] *svg*. original-date: 2012-08-26T03:38:25Z. Jan. 2022. URL: <https://github.com/npm-dom/svg> (visited on 08/03/2023).