Annex 10
The (re)Indexing System and Research portal

1. INTRODUCTION

The NaturalHeritage project aimed to reduce knowledge gap in Natural Sciences information and collection data by acting at two levels:

- Rationalize the management of collection data of each partner institution by maintaining and strengthening the existing data management systems. A technical convergence between these systems would also be searched
- Provide a common Internet portal to access and search the data of the three partners institutions (RBINS, RMCA and NBM Meise)

We observed that these two objectives correspond to a technological divide. Collection management systems (CMS) used by institutions to manage their own data are based on the well-proven relational data model, and often (but not always) rely on the SQL language. Data is therefore well-structured, easy to exchange in scientific context, but the systems themselves are not easily scalable. But as these systems are now old and mature (SQL was created in 1974 and standardized in 1986), widely used and taught in high schools and universities, it is relatively easy to find manpower to develop and maintain them.

On the other hand, search portals linking the data together rely on quite recent technologies and concepts, linked to the BigData and NoSQL worlds. Programming paradigms (e.g. Map/reduce, aggregations, faceted searches, clustering technologies) and trends evolve rapidly, as this field is less standardized and more dependent on the evolution of hardware capacities (in terms of storage and calculation speed) than relational databases. It is more difficult to appoint staff to manage the system as these are emerging technologies.

One of the indirect aims of BICS was to allow institutions to get more familiar with NoSQL systems and develop the skills and capacity to conceive in-house and maintain systems based on these technologies

Besides, workflows and development efforts differ from those prevailing with relational databases: relational databases require to define a pre-existing data structure, and to map data prior to their integration in the database systems. Exported data need to be restructured by a third party tool or backend solution (which could be web service) to exchange data with scientific networks (such as GBIF, WORMS)

NoSQL and BigData software deal with more flexible data structures that can cope with heterogeneous models, and, as they natively integrate REST APIs, have less layered architectures. This makes them actually faster to install and deploy, as they can work as standalone applications. However, and this tends to be overlooked, big-data systems are still dependent from systems storing the primary data, which are often relational databases, as they are optimized for rapid searches, and not for write and update operations that are needed by the day-to-day tasks.

The technical focus therefore shifts from data integration and modelling towards searches and data visualization and organizing the resulting data (with the concepts of facets or aggregation that are hierarchical views on the data), hence the terms *data engineering* and *feature engineering*. Search algorithms used by NoSQL solutions are also closer to natural languages (eventually with translation issues) as their aim is to configure a powerful search engine for heterogeneous data already integrated in institutions databases, rather than [1]standardize the data themselves. The two NoSQL systems envisioned at the beginning of the project, SolR and ElasticSearch, both feature an extensive set of configuration options to analyze and parse unstructured text in several languages. This implies a certain degree of lexical analysis, as these systems can recognize the stems of conjugated verbs, the singular and plural forms of nouns, the stop words and diacritic signs used in text. One of the research aims of NaturalHeritage was to identify those parameters and integrate lexical analysis inside the portal, so that their users can rapidly and smoothly identify relevant and meaningful keywords that are present in non-structured texts.

2. METHODOLOGY

2.1 Analysis of needs

The analysis of needs took the first 10 months of the project. At the beginning of the project, a big star-shaped diagram of the envisioned architecture of the NH portal was established and stored in the MARS system of the RBINS. This has been first guided by the technical

[1] *Biodiversity Information for Collections in the South*, Project of the RMCA for the 2019-2024 Framework Programme exercise of the DGD

interoperability between the portal and existing systems, linking them together by limiting the amount of additional technical developments made directly on them.
Two different approaches have been envisioned:

1. avoid all redundancy in the storage of data,-and create a fully decentralized search engine
2. consider the search engine as an harvester having a copy of the data, that could enrich them with links to related objects disseminated across all institutions and provide additional functionalities that are absent in the original systems, e.g.
    - geographical maps,
    - search engine optimisation (improvement of their referencing in search engines) ,
    - compliance with data exchange standards (like OAI-PMH for bibliographical data)

The solution that has been finally chosen follows the second type of architecture. A fully decentralized search engine would have increased the coupling between systems, both in terms of architecture, as each of them would need a dedicated data-interface to exchange with the other systems, and in terms of availability, as the whole system need each component to be available on the Internet to be functional. This would consequently increase the development effort, as these interfaces would need to be modified if the data-structure or technical specification of one of the components changed.
A centralized search engine allows exchanging data at a higher level of abstraction, improving the stability of their structure and their independence from their actual technical storage. A certain degree of standardization would be reached by using or developing solutions following, partially or totally, the paradigm of REST (Representational State Transfer) APIs. REST systems publish data through HTTP URL with meaningful GET parameters giving resulting data in a format which is easy to parse, such as JSON.
Besides, the choice of a centralized index regularly harvesting the data would ease reuse of workflows and tools that have been developed by other projects focusing on data mobilization and exchange. One of the most significant steps in this direction has been the development of a crawler harvesting data that are published in the DarwinCore archive format being used by the IPT (Internet Publishing Toolkit) of the GBIF. This would merge and rationalize the workflows required to contribute to the Belgian NaturalHeritage project and to publish data to GBIF and WoRMS. A DarwinCore mapping of the collection data stored in Darwin has been developed, under the form of a materialized PostgreSQL view. The corresponding collection metadata would simultaneously be mapped. Thomas Vandenberghe, who developed custom-made R scripts, mapped in the EML format (Ecological Markup Language) which is the second protocol used by GBIF.

The diagram revealed that one of the most important gaps existing in the model was between collection data and bibliographical data, which are poorly connected. MARS publishes the publication data of the scientists of the RBINS. We first envisioned the possibility to export the data in a static data format compliant with EndNote and/or Zotero and to re-import them in Darwin. However, this solution would shortcut the portal and create a redundancy of data. It has therefore been decided to focus first on bibliographical data.

Therefore:

- The first case-study data to be imported in the index would be bibliographical data
  A REST API would be developed by Makina Corpus and added to the MARS system (Multimedia Archaeological Research system developed in Plone, 2006-2021). A

specific crawler (in Python) would be to developed to browse, and parse these data and import them onto the index
- A focus would be put on the development of powerful lexicological analyzer able to parse data in free text formats and natural English (like abstracts of scientific articles) and extract meaningful keywords
- The portal would be used to convert these data into formats and protocols for data exchange that are relevant for bibliographical data (like OAI-PMH/ DublinCore)

3. INFRASTRUCTURE

Regarding the portal, two servers in Ubuntu or Debian have been allocated to the project, one at the RMCA (https://naturalheritage.africamuseum.be) and one at the RBINS (http://ursidae.rbins.be, proxified via https://darwin.naturalsciences.be/portal).

Several NoSQL solutions have been compared at the beginning of the project:
1. Hadoop
2. MongoDB
3. SOLR
4. ElasticSearch

It has been decided to pursue the development of the portal in ElasticSearch as:
- this system is known to be scalable: it keeps a constant level of performances as the database increases in size (which is not the case of MongoDB)
- it is easier to deploy as it doesn't require a specific formatting of the server file-system (such as Hadoop). It can therefore be installed on a non-dedicated infrastructure
- securing the server with ElasticSearch doesn't imply configuring a cluster, such as with SOLR. However, we noticed while installing that the security module of ElasticSearch is commercial, and that most of the documentation available on the web recommends using a third-party software (for instance Apache proxy and password protection of a website) to secure ElasticSearch. We followed this approach during the project
- ElasticSearch has a wide user community, and documentation is readily available on the Internet, both on the ElasticSearch website and forums such as StackOverflow.
- It is also linked to related technologies that were relevant for the project, such as Kibana (graphical dashboard for analysis) and Logstash (ETL tool for pipelines and workflows), although these technologies (and their documentation) are only partially Open-Source

Begin of 2017, a repository specific for the project has been created on the GitHub page of the RBINS, thanks to the ICT service of RBINS
https://github.com/naturalsciences/natural_heritage_search_engine/

Two ElasticSearch servers have been developed, one with the ElasticSearch index (http://ursidae.rbins.be) and one with Kibana (http://macropus.rbins.be). The portal itself (in Symfony 3) is installed on the server also storing the PHP backend of Darwin (http://diptera.rbins.be). All these servers are available on the Intranet of the RBINS, but the Symfony portal is published on the Internet via a Proxy (implemented as an Apache RewriteRule) linked to the darwin.naturalsciences.be domain: https://darwin.naturalsciences.be/portal).

We didn't configure an ElasticSearch cluster or shard, gathering several machines, and able to handle downtimes.
At the time of the writing of the documents, the NaturalHeritage index contains 2 526 535 records from the three partner institutions and uses 10Giga of hard disk storage space. The biggest part of the data probably consists of the NGrams (parts of words overlapping themselves, used to create autocomplete and fuzzy matching) that are stored with the data, duplicating the original data. NGrams are analyzed and stored when a record is created. The storage space is moderated, but while working on the server, we noticed that ElasticSearch swapped a lot of data when a record was being updated, potentially more than 200% of the original storage space. ElasticSearch is therefore optimized for data creation and search operation, but does not cope very well with the update operations that would be required to maintain primarily collection data.

The first version of the portal and ElasticSearch, up to January 2018, have been developed on a server located at the RMCA (https://naturalheritage.africamuseum.be/portal). This server has also been used to test Specify among the Collections Management Systems and SOLR.

4. RESULTS

4.1 Elastic Search

ElasticSearch, although it has a free kernel (on the Apache2 license) is not a fully open-source solution. It is supported by a commercial company, and some modules (like security and export functionalities of Kibana) and consultancy activities are commercial. Nothing guarantees that this business model will remain the same on the long term. Apache Solr, which is technically close (also based on Lucene) is closer to the open-source philosophy, however it appeared to be more complex to configure (with a lot of static XML files to edit) and its available documentation was less structured than with ElasticSearch at the time of the beginning of the project. ElasticSearch, although partly commercial, offers a REST API (though in HTTP POST format), and is linked to complete client libraries in PHP and Python, allowing rapid development of custom pipelines to import data and extraction tools to convert them.
The main criteria that decided us to adopt this solution was the easy access to documentation, its scalability and good integration with standard medium-sized hardware infrastructure, and the prospect to rapidly develop an operational prototype that could be progressively expanded.

Regarding the backend for the web portal, we decided to use the Symfony portal as:

- This should ease the interoperability with Darwin which is developed in Symfony 1. Staff was already familiar with Symfony as they were contributing to Darwin
- The Symfony framework is still actively developed and has a wide user community, although versions are not backward compatible. At the time of the project beginning, a technical convergence was also reached between Symfony 3 and Drupal 9
- Recent versions of Symfony use the Twig templating system which is convenient and easy to handle. It also doesn't depend from JavaScript, which should help the referencing of data by Google (which was a requirement of the project)
- Symfony has a well-documented ElasticSearch library, that also helped us a lot to get familiar with ElasticSearch concepts:

ongr (https://github.com/ongr-io/ElasticsearchBundle)

- Since version 2, Symfony uses a modular approach with the so-called “bundle”, which makes expanding an existing minimalistic prototype with additional services easier. It also allows reuse of external components (that can be based on an object-oriented interface) in a relatively smooth way. We benefited from this architecture when adding an OAI-PMH layer to the portal. This makes the development of the initial prototype longer, but is very relevant when it comes to expand it and manage the scalability of the application.
  The first version of the backend portal has been developed in Symfony 2. It was converted into Symfony 3 mid-2018, when this new release of the package was made available. Conversion was easy as these two versions of the library are mutually compliant (which is not the case with Symfony 1 used by Darwin).

4.2 Data model

It has been decided to base ElasticSearch on an explicit (i.e constraining) data-model in JSON format. The following principle would be established:

- The main resource provided by the index would be the URL of the resource describing the object on the Internet. This URL would be both the landing page to access the resource, and the identifier of the data in the index. This choice had several implications for the data to be indexed.
- Indexed data must be made available on the Internet prior to their indexing by the portal. Each partner institution must have a functional institutional database storing their collection object.
- The portal would therefore not deal with collection metadata, but directly to the individual object, specimens or documents
- Their URL are supposed to be stable and unique, as they are both locators and identifiers
- The indexed data are meant to be human-readable, in HTML format, rather than raw data (in XML or JSON format).A content negotiation mechanism (like the one developed by CETAF for stable URL) could be implemented by the provider of data to serve both versions of data at the same URL

Regarding the data model, we decided to base ourselves on the **Balat** portal developed by the KIK-IRPA (http://balat.kikirpa.be/intro.php), which uses a free text field and 4 main criterias to index and search data:

- *Who*
- *What*
- *Where*
- *When*

We decided to

- associate the fields to autocomplete lists
- Add a second hierarchical level to these fields, specifying the role of the object. e.g
  - *Who*
    - *Collector,*
    - *author,*
    - *Identifier,*
    - *Creator of the data*
    - *Curator of the collection*
    - *etc...*

- *What*
  - *Type of object (document, collection specimen, publications, etc…)*
  - *Biological type*
  - *Electronic format / Mime type*
  - *etc...*
- *Where*
  - *Collecting country*
  - *Collecting locality*
  - *Regions*
  - *Ecoregion*
  - *Maritime area*

- *When*
  - *Collecting date*
  - *Publishing data*
  - *Creation date of the dat*
  - *...*

This second level would be optional: the user could fill data at the more generic first level or use these fields to increase the granularity of the research. These fields would be hidden during the initial load of the portal, but a button would toggle their display.

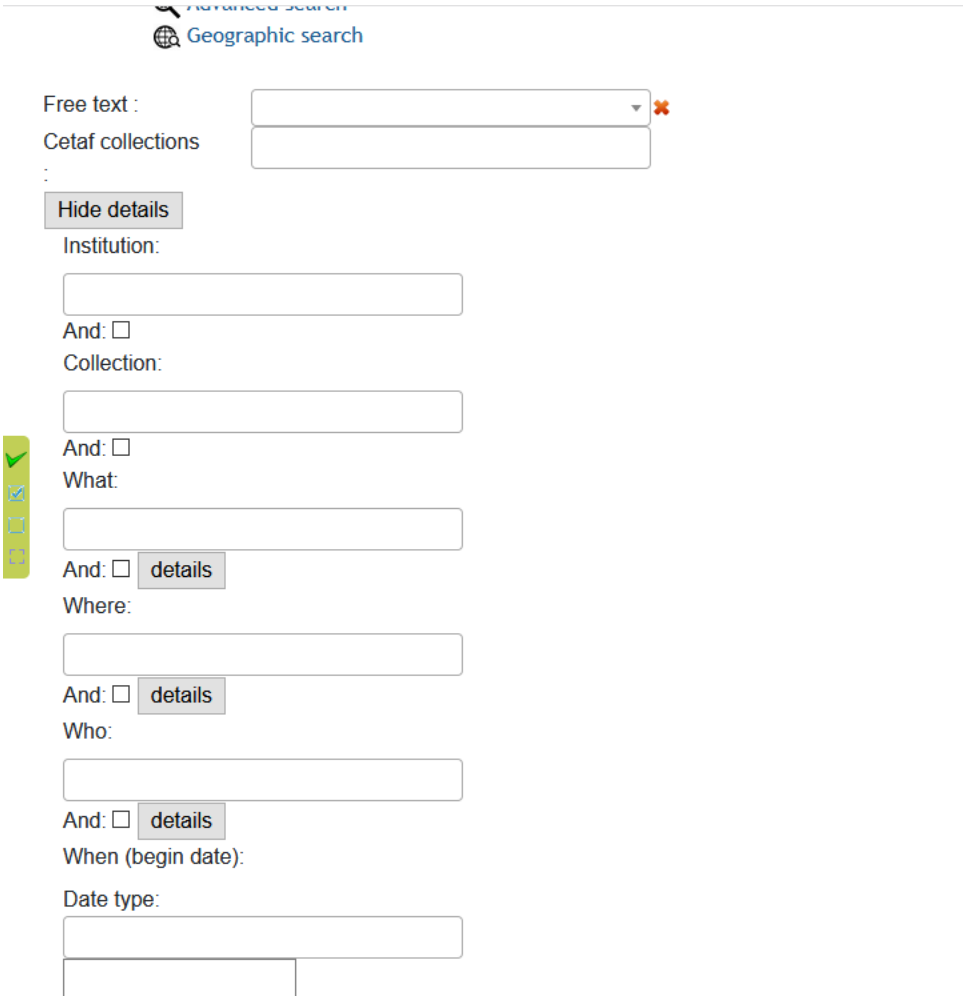


Figure 1. First level of keywords (unexpanded)

This second level of data would be dynamical, and not constrained: keywords used for this second level of classification being themselves a variable of the data record.

The search interface (for this part of the portal) would therefore be built dynamically from a faceted search query, triggered when the user loads this part of the interface.

Figure 2. Second level of search criteria available for “who”

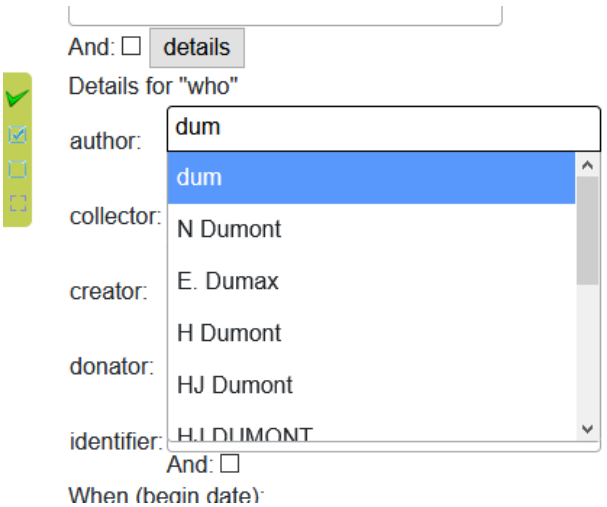


Figure 3. Autocomplete for the second level keywords

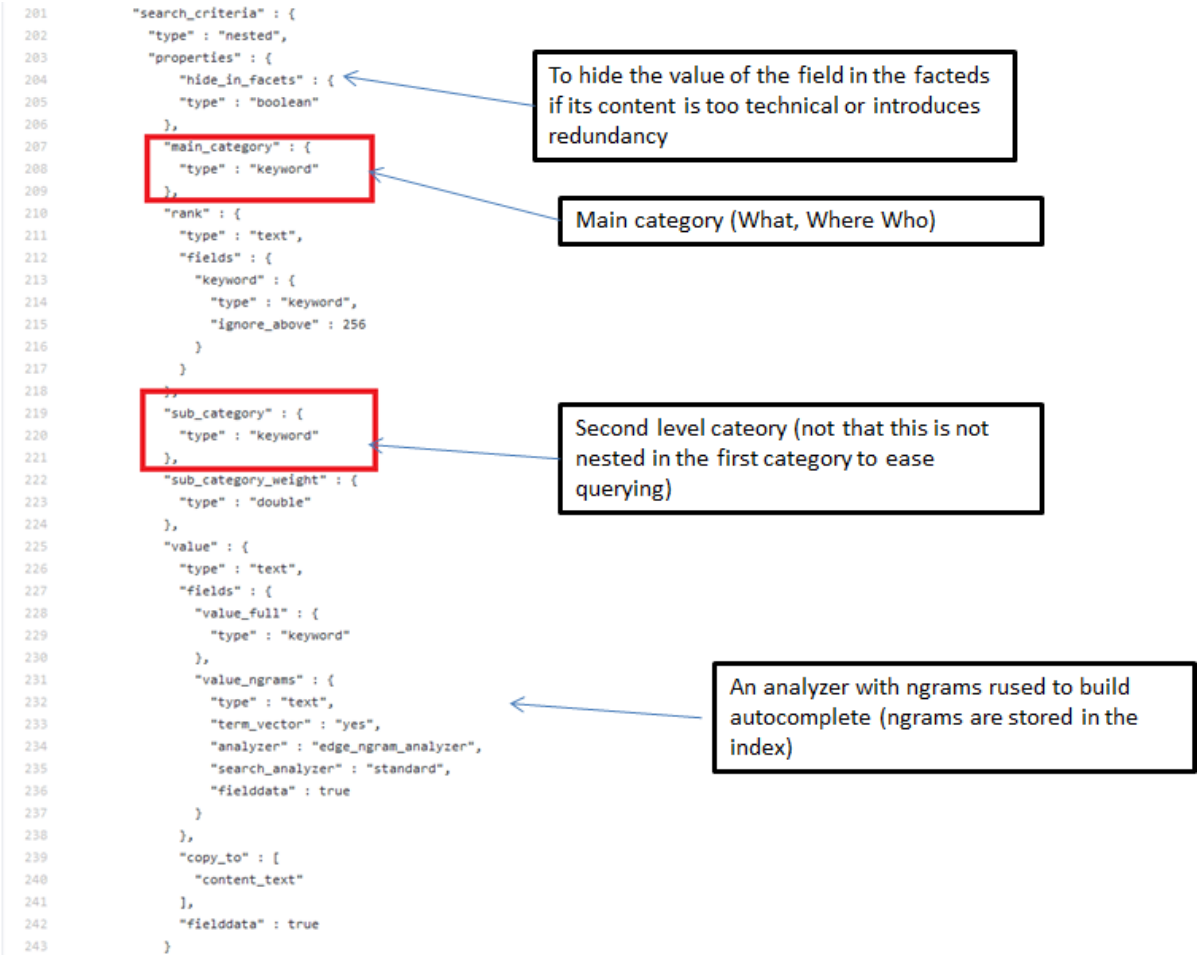


Figure 4. Definition of the two level keywords in the ElasticSearch JSON model

This solution is scalable but has two drawbacks:

- It increases traffic and latency when using the portal (although ElasticSearch is reasonably fast when building aggregations).
- The effort to have a standardized and cleaned list of keywords for search categories is transferred to the crawlers aggregating data, and the values cannot be deduced and documented from the data model. It is still possible to clean these keywords by using third-party scripts controlling and aligning their spelling and typography, that could be run on a scheduled basis, but this would require a certain degree of manual intervention, when defining a controlled reference vocabulary and comparing the value of the keywords present in the index.

Extra fields have also been appended to access data:

- The main identifier of the object in the original collection
- Its conservation institutions
- The department managing the collection (e.G. Collection Management, library)
- The name of the main collection (e.g. Vertebrates)
- The name of its subcollections (.eg subcollections)

There are therefore 4 nested hierarchical levels to classify the position of an object inside of one institution. These hierarchical structures have been used to build a second-faceted search engine being used in the interface, allowing the user to navigate and filter by successive refinement inside of the result of a first search

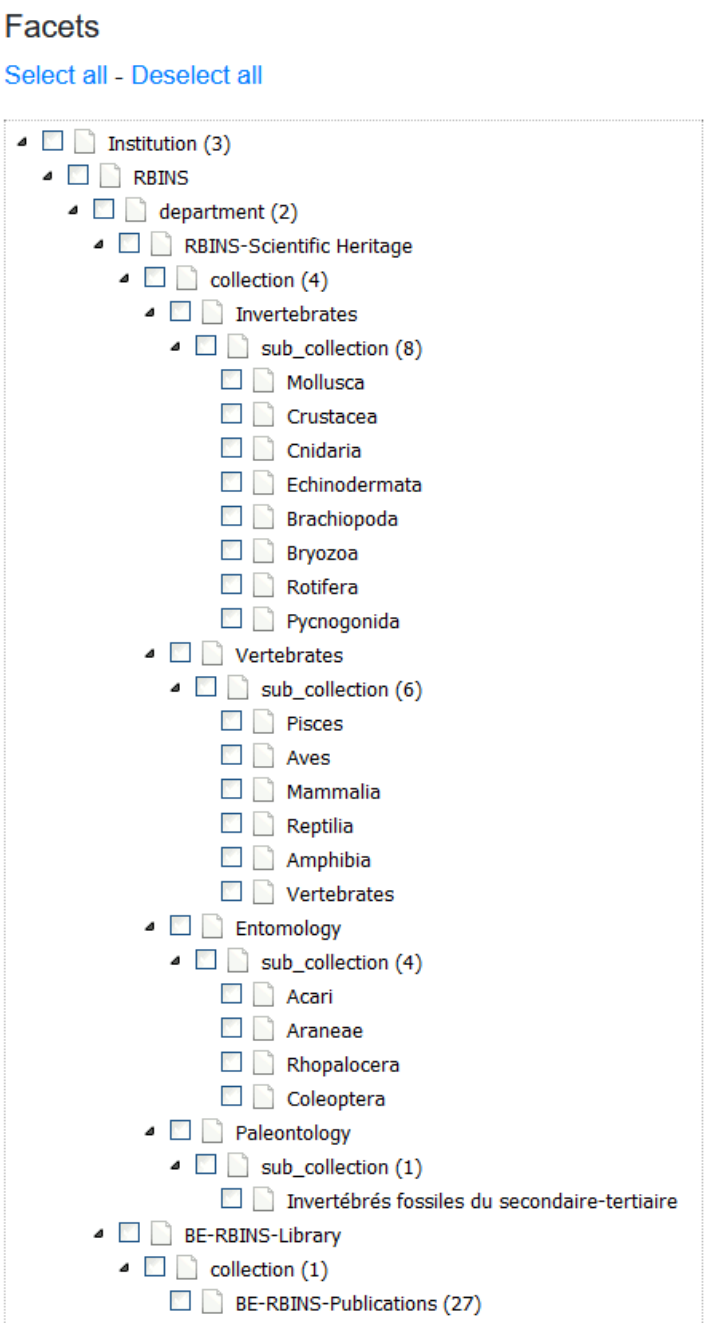


Figure 5. Hierarchical faceted results: institutions and collections

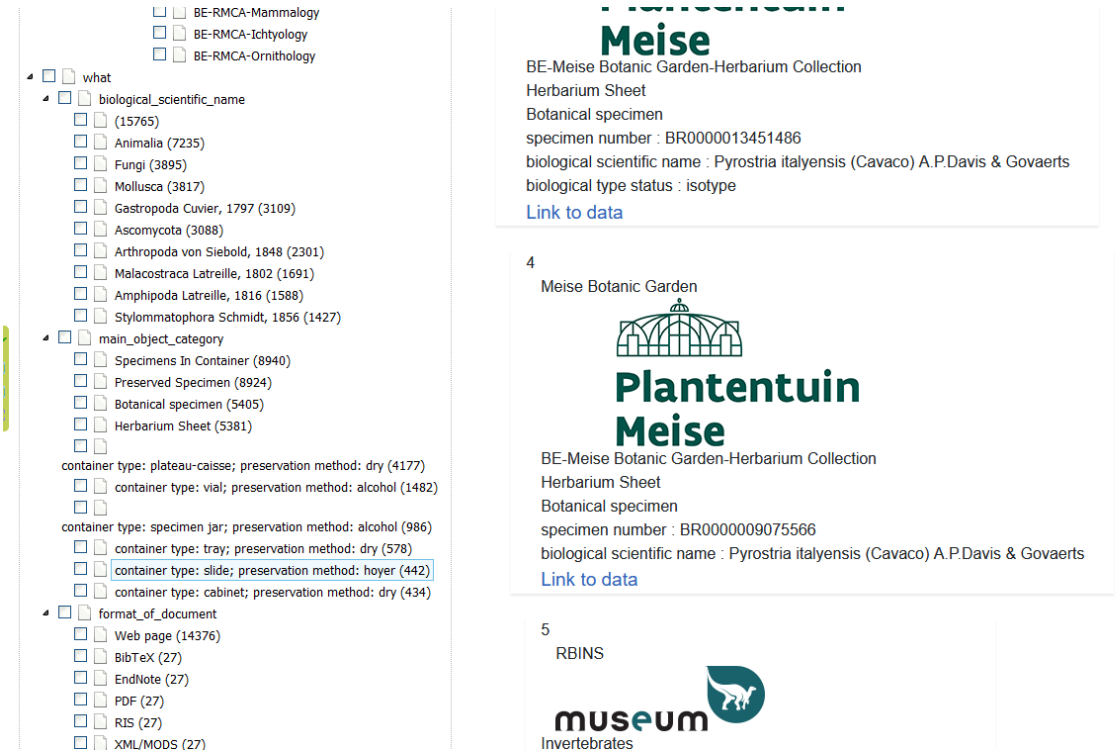


Figure 6. Hierarchical faceted results: “what”

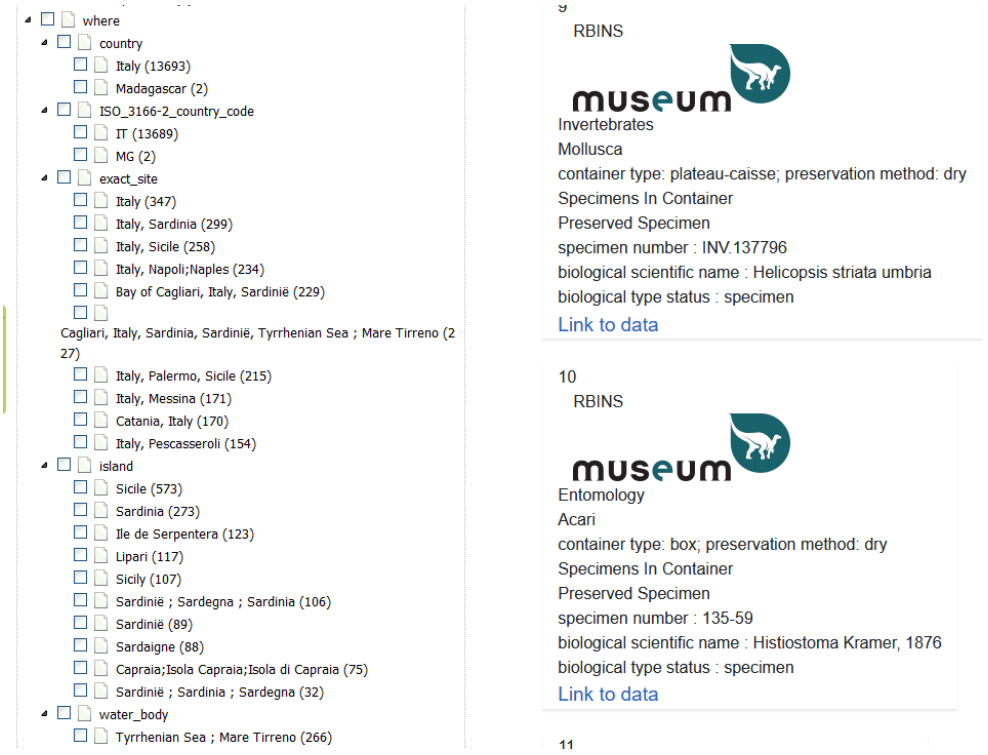


Figure 7. Hierarchical faceted results: “where” (from the search keyword “Italy”)

This faceted search engine has been developed by using the “FancyTree” JavaScript library that can be linked to Ajax data source (in this case: a JSON mapping of the ElasticSearch result provided by the backend).

Some more additional fields have been added for:

- the geographical point coordinates

- the additional links. Initially we didn't forecasted to handle images in the portal, to avoid the issues related to their storage and copy, but we implemented in November 2020 an IIIF link

Please note that these links are not dynamically categorized, and statically defined in the model. This is related to the fact that they are used to uniquely identify the provided document, and to structure the graphical display of the HTML page.

We also added a transversal free text field. This field copies the values of all other search fields (except dates and geographical coordinates) and can also store a free text in natural English (like the abstract of a scientific article). This field is the first field being displayed in the search portal. It is linked to the most sophisticated autocomplete mechanism and text analyzer of the index. Figure 8 a and 8 b list the parameters used to configure the text analyzer.

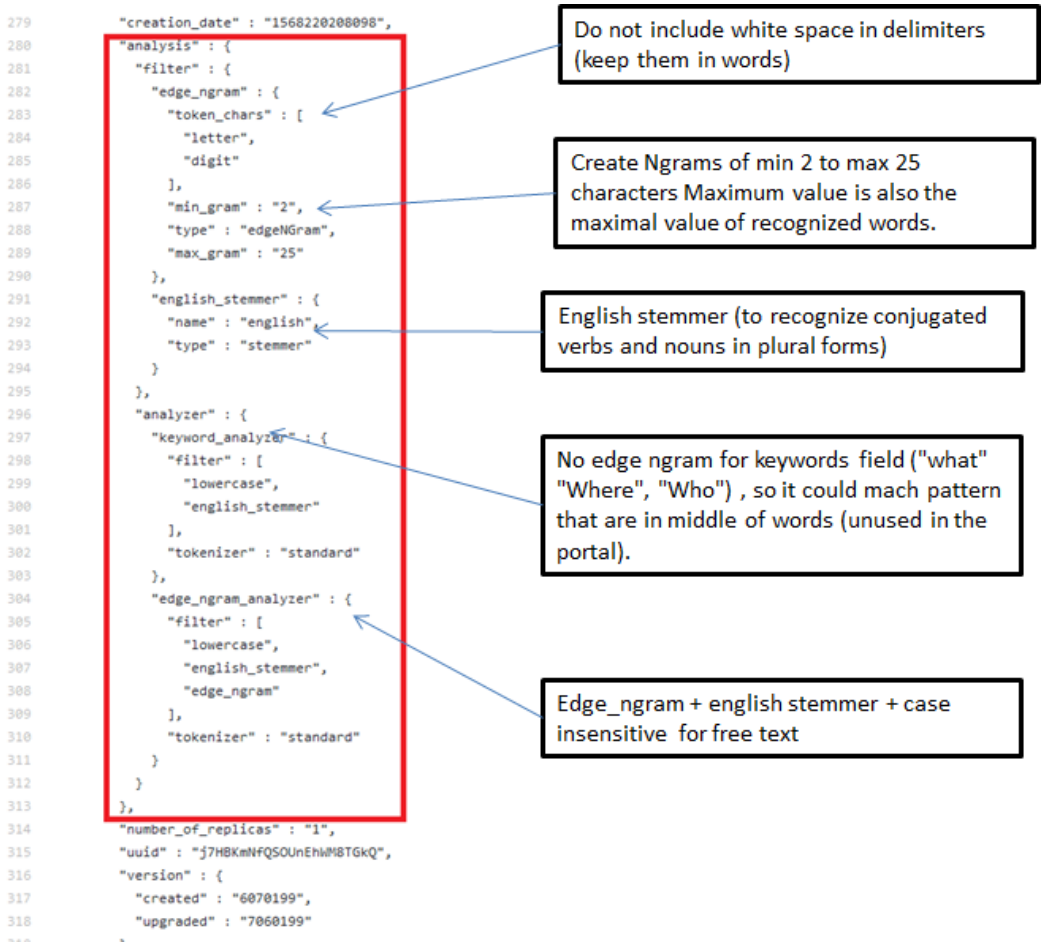


Figure 8.a Definition of string Tokenized and text analyzer

Figure 8.b Definition of string Tokenized and text analyzer (association to field)

The Figure 9 shows that white spaces, diacritics and punctuation are included in the results of the search engine). The resulting search engine is quite fast and can efficiently highlight text patterns both in keyword lists and text written in natural English. However, as ElasticSearch is a Java system, the first run after a period of latency can be quite slow, as it may involve a recompilation of the platform (the so called JIT “Just-in-time” compilation).

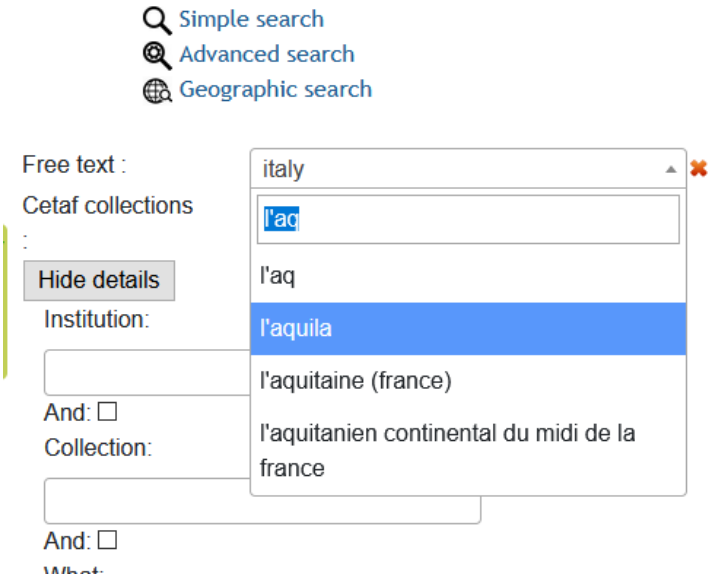


Figure 9. String tokenization in the free text autocomplete

4.3 Data ingestion

Insertion of data in indexes are made by Python scripts, acting as crawlers, connecting the APIs or interface for data extraction of the source system, and building ElasticSearch queries.

Their code is available in a specific folder of the GitHub repository:
https://github.com/naturalsciences/natural_heritage_search_engine/tree/VERSION_2020/crawlers_elasticsearch

They can be ran on scheduled basis, e.g via automation by Cron jobs

Three crawlers have been developed, handling different data formats:

1. one crawler connects the REST API for publications of MARS (access point is:http://biblio.naturalsciences.be/@nh-search?). It has been used to create the documents that have been mapped afterwards into the OAI-PMH and DublinCore standards (see portal description).
2. One crawler connects the REST/JSON representation of Darwin data, that has been added in 2017 to this database. It browses the public collections in Darwin, and features a paging mechanism to access a second page in JSON format. It has been used to import the data from RMCA
3. A third crawler can upload files in DarwinCore Archive formats, which is being used by the GBIF IPT. It reads the specimen CSV file of the archive package and exports it as ElasticSearch queries . Source code is available on https://github.com/naturalsciences/natural_heritage_search_engine/tree/VERSION_2020/crawlers_elasticsearch/ipt_darwin_rbins
   This script has been used to import data from NPM Meise Botanical Garden and the Darwin data from RBINS. This is the fastest solution, as it doesn't require any HTTP connection to connect each data individually, the file being unzipped on the server itself and data are read by streambuffer I/O operation. The zipped files are unzipped in RAM memory (though possibly swapped if they have a big size), the uncompressed version is not directly saved on the server hard drive. This parser is very interesting as it could map to the index structure any other data sources than Darwin or MARS API, as long as they are published through the GBIF IPT. As GBIF associates a DOI to published datasets, it can also link the index to citable data.

These scripts have been developed as Python classes, in order to make them extendable to other data sources. The connection parameters (IPs of service, file path, connection credentials) are not hardcoded, but stored in ancillary configuration files. It is important to note that the efforts to include other data source in the future is linked to the development of these crawlers, while the data model of the index should remain stable

4.4 Statistics and data visualization

The indexes have been linked to a Kibana tool that helps data visualization, with statistics and graphs. Kibana can also be seen as a graphical query interface helping to build ElasticSearch queries (which can be verbose and tedious to write) The tool is operational. One of the limiting factor is that Kibana is only partially free, features like exportation of logs data to files are commercial. Besides, Kibana doesn't handle hierarchical JSON structures very well (the data schema of Natural Heritage features a lot of hierarchical fields) and requires a "flattened" 2-dimensional copy of the index to work. This can be done in realistic times (circa 10 minutes to replicate the 10 Goge index) but increases the storage space usage on the server. This is again a factor enabling us to say that solutions like ElasticSearch are useful to build fast and strongly indexed search engines, but not optimized for the storage and management of primary source data, as the data model can be more constrained by technical factors than for relational databases.

4.5 The NaturalHeritage Search portal

The portal views were envisioned as the graphical user interface for the ElasticSearch indexes. This would represent our first attempt to create a graphical search engine on top of NoSQL and BigData technologies. Its usage should be intuitive. One of its key concepts was that it should not store the reference (or primary) data, but provide an URL link to the original resource. By design, the portal would only give access to data that are already published by their institution, via an unique and unambiguous URL. It would complement the existing web portal and collection management systems of partner institutions.
It should be an application easily allowing the update of data, as the original resource may be modified or deleted in the source institutions.
It should also deal with heterogeneous content, not only description of specimens, but also abstract of scientific paper and metadata of collections that are not yet digitized. It follows the data schema of the ElasticSearch indexes (with the WHO, WHAT, WHERE, WHEN search criterias).
The portal would not only publish the data in HTML, but could also publish raw data in REST JSON format and even other data protocols. It was required to use OAI-PMH (Open Archives Initiative Protocol for Metadata Harvesting) and DublinCore, two XML-over-HTTP schemas for bibliographical references.

We decided to use a Symfony/PHP framework, as their modular approach could allow us to create one module for each specific content type and output format. We used Symfony 2, and later Symfony 3, for four reasons:

1. DaRWIN was already the first version of the framework, while also being part of the NaturalHeritage project
2. At the time of the project, Symfony2 was becoming the technical basis of the Drupal (version 8) content management system, which is used by RBINS and RMCA for their main website. This technical convergence would help to allocate manpower for developing and maintaining the portal. This would also ease its technical integration in the web infrastructure of partner institutions.
3. Symfony 3 uses the Twig system for HTML templating, which is particularly easy to use and allows a good integration with JavaScript libraries and Bootstrap, a CSS/JavaScript library for responsive HTML layout
4. As Symfony is widely used, there are several freely available and well-documented libraries provided by user for the integration of ElasticSearch and OAI-PMH

However, there is no backward and forward compatibility between Symfony 1 and ulterior versions, meaning that the development of the two systems would remain separated.

Other frameworks have also been tested, such as AngularJS, but were rejected as JavaScript solution would perform poorly in terms of Search Engine Optimisation (SEO), as crawlers of web search engines like Google were only able to read and parse raw HTML informations in web page. This would prevent the integration of solutions for a better referencing of data based on Microdata that were emerging technical paradigms at the time of project inception.

Finally, the search views can be integrated in another CMS used in the institution. We used the Plone 4.3 CMS at RBINS as the front-end portal of NaturalHeritage. The Symfony/PHP views are embedded as Iframes in the Plone html. This allows to manage the users, the access rights, the menus and the new DaRWIN interface with a versatile, WYSIWYG and customisable solution.

4.5.1 Elastic search and Symfony

An appropriate ElasticSearch driver library for Symfony has been first identified. We choose ONGR as it is free and well-documented. It's structured and clear documentation enabled us to get more familiar with concept and design patterns that are specific to whole ElasticSearch indexes. However, it maps ElasticSearch queries into fully Object-Oriented structures. As developers working on the project were getting more familiar with ElasticSearch, they gradually wrote ElasticSearch statements directly under the form of multi-dimensional PHP arrays, which are less verbose and more portable (easier to convert in JSON structures).

A prototype of the portal was developed at the end of 2017 and beginning of 2018. It would need to be dynamically adaptive, as the second-level search keywords (e.g. WHO/collector, WHO/author, WHO/donator) are not defined in the model but correspond to variables with an unlimited range of values.

The main technical challenges were therefore:

1. For the backend part: create dynamic ElasticSearch search statements (as the structure of the search interface is not constrained and non-deterministic)
2. For the graphical front-end: build a semi-structured interface whose first-level keywords are statically defined while the second-level keyword are dynamically created

These two components needed to be synchronized together. The usage of the Symfony framework which isolated the controller in a specific PHP class, with a standardized "Request class" to access HTTP POST or GET parameter was there very relevant and helped us to develop the code writing conditional conditional ElasticSearch statement, with a reasonable level of complexity.
This also means that the web interface has a specific state as the current structure of the interface has to be passed to the controller. A first version of the portal used server-side session variables to store the state of the interface. It has been progressively replaced by JavaScript storage containers and HTTP GET variables, as server side session variables worked poorly in the context of HTTP frames, as they depend on security-settings for cookie variables that are browser-specific and more and more strict.

Another key concept of the portal was that no data would be serialized or stored in a database, except the ElasticSearch Index data. In other words, the portal would perform read-only search operations. This would ease the definition of a dynamic and self-adaptive search interface, as no static object mapping of the index structure in PHP classes would be needed, like this is usually the case when working with server-side frameworks and relational databases. This implied to write most of logic of the graphical interface in JavaScript, by using appropriate libraries:

1. JQuery to group together and structure the search criterias, that are passed to the server-side controller as HTTP GET statements
2. Select2, a library that allows to create complex autocomplete statements (allowing multiple selections). These autocomplete fields can be dynamically created, and are filled by HTTP GET statements linked to a generic single access point of the controller.
3. FancyTree would be used to build the hierarchical faceted view of the result

These facets would be provided in JSON format. Therefore the main HTTP controller of the portal would need to provide two data outputs in JSON formats: the main reply with the URL link to the original resource, and the structure with the aggregation to build the facet trees.

4. OpenLayers would also be used to create the maps to search data and displayed georeferenced data

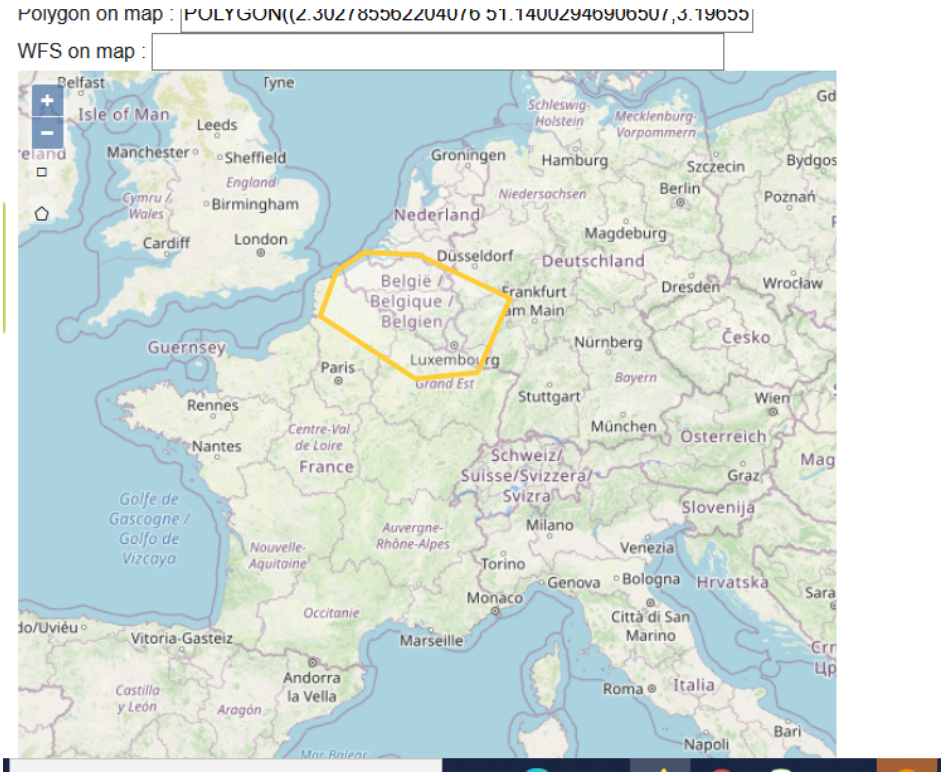


Figure 10. Geographical search with OpenLayers, Free polygon drawing

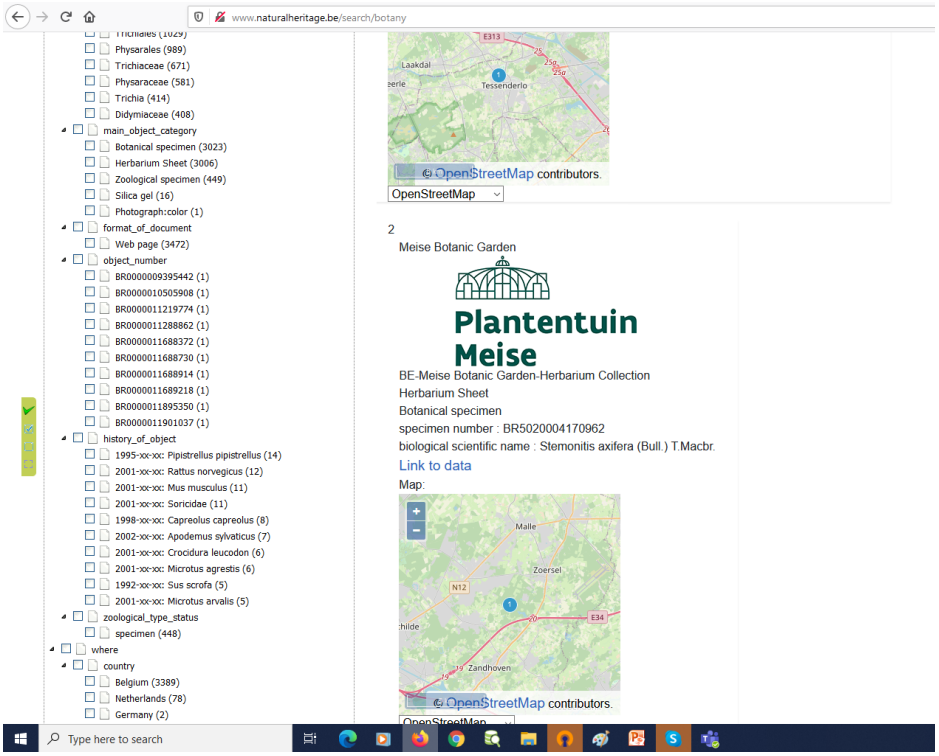


Figure 11. Result of a geographical search (maps + faceted results)

The geographical search allows the user to draw bounding box and free-hand polygons on the map, like for DaRWIN. It is also possible to connect the WMS defined for DaRWIN to select administrative and natural areas by simple clic.

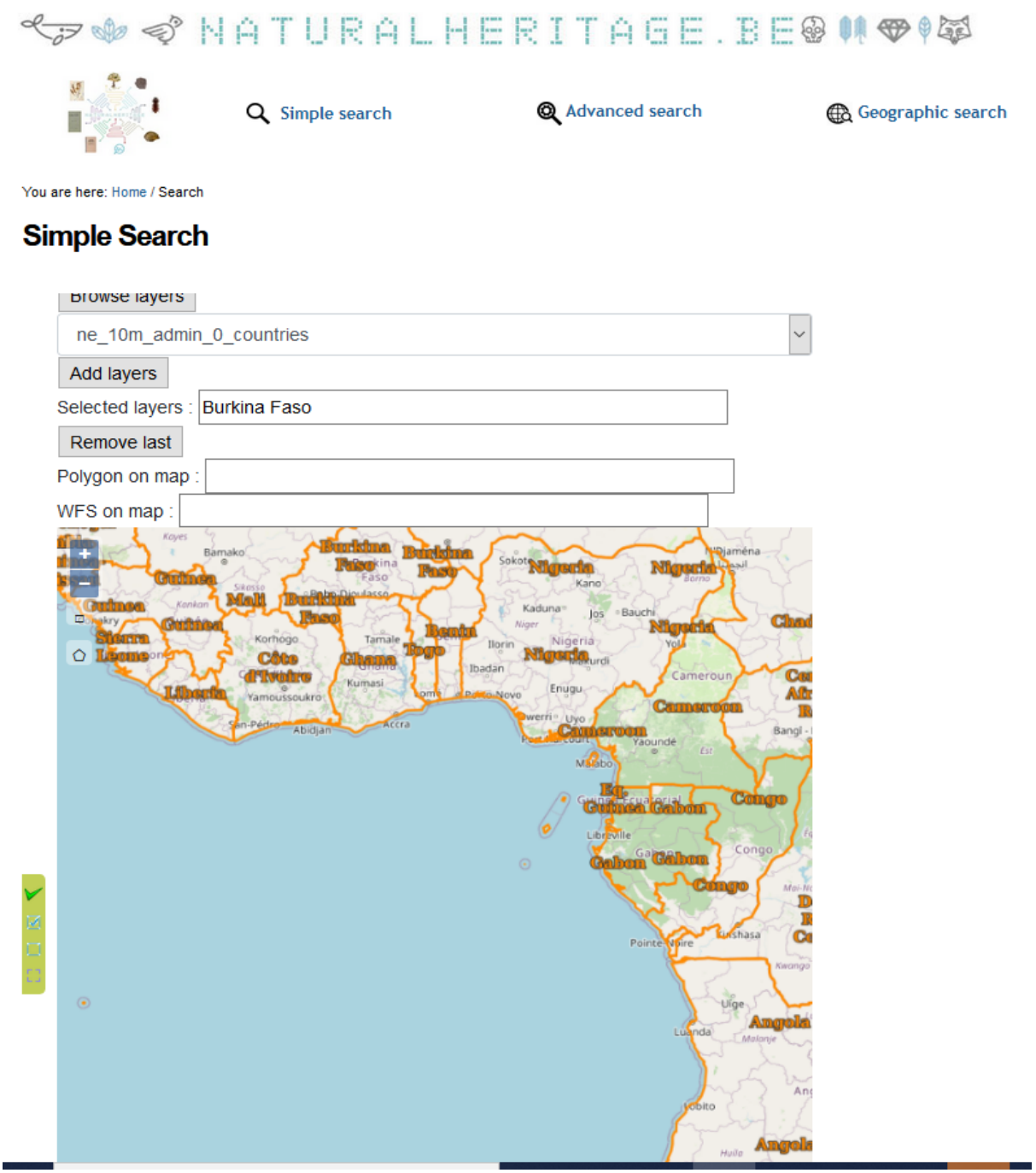


Figure 12. Selection of Burkina Faso as search criteria from WMS layers

4.5.2 Portal Workflow

The figure 4 shows the overall query workflow of the portal, and the different types of queries handled by the controller:

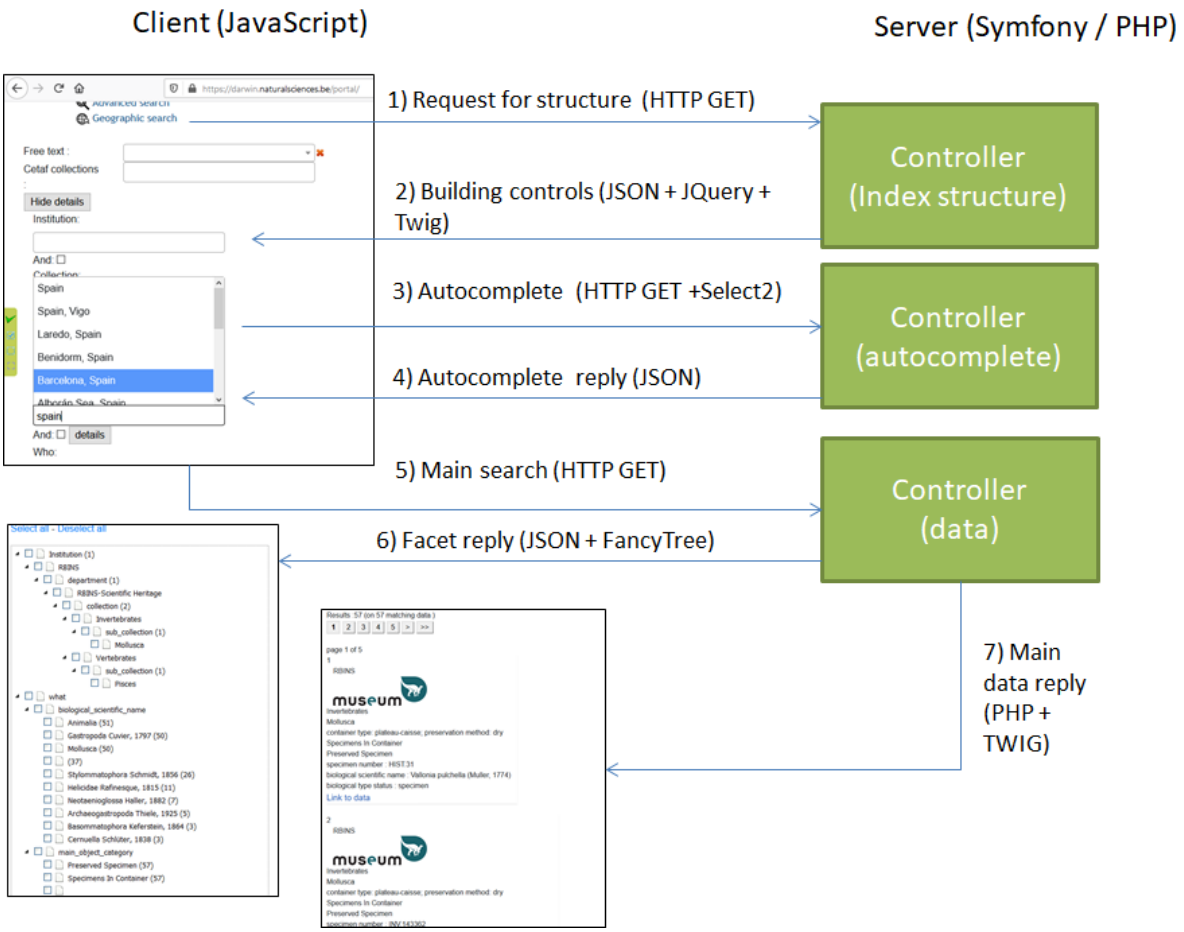


Figure 13. HTTP workflow of the portal

Bootstrap has also been appended to the portal, to make the layout responsible and compliant with the small touch-screens of tablets and mobile telephones. The integration of Bootstrap in Twig was quite straightforward.

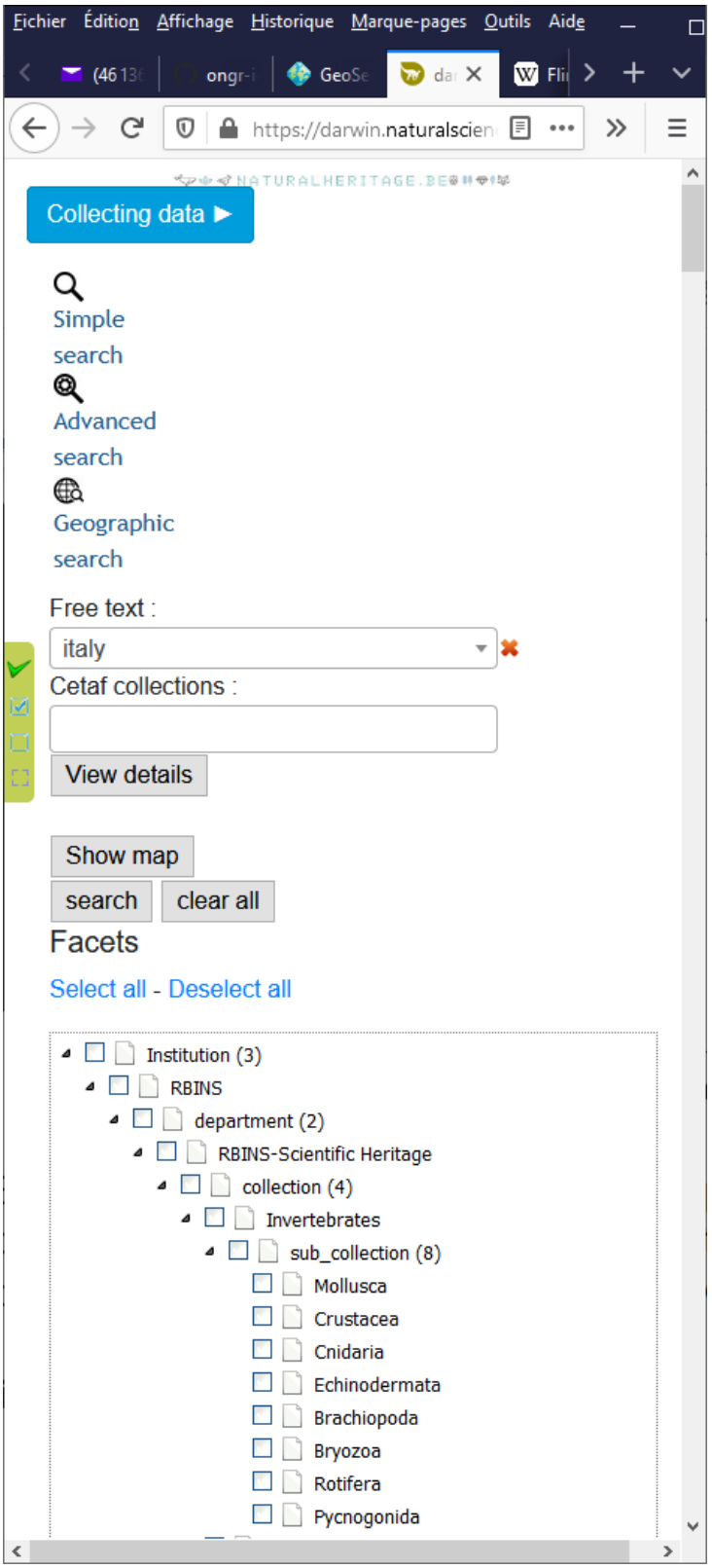


Figure 14. Usage of Bootstrap on a narrow screen

4.5.3 OAI-PMH Server

An OAI-PMH server has also been appended to the server, so that it can serve bibliographical data in DublinCore format. OAI-PMH is an XML-over-HTTP query protocol which is widely used by networks of academic and scientific libraries. This would allow the portal to be harvested by external aggregators. The OAI-PMH server is based on code published on GitHub by the company Naoned, based in Nantes, France, which kindly allowed us to reuse (and event fork) this component. The original code was made of abstract

classes and interfaces. The consistent structure of the code allowed a relatively easy adaptation to ElasticSearch while the original code was intended for SQL relational databases. The most technically challenging part of this development was the implementation of the resumptionToken, a mechanism specific to OAI-PMH, based on a hashed server-side cookie, enabling (forward only) pagination of query results. The OAI-PMH server has been implemented in a specific Symfony bundle. The usage of a PHP framework actually eased merging custom code developed in-house by project developers with external components made available on the web, this is a typical situation where server-side frameworks can improve the scalability of a web application.

The access points of the OAI-PMH service are:

1. Self description of the service:
   https://darwin.naturalsciences.be/portal/app_dev.php/oaipmh/?verb=Identify

2. List of catalogues:
   https://darwin.naturalsciences.be/portal/app_dev.php/oaipmh/?verb=ListSets

3. List of published resources (identifier level):
   https://darwin.naturalsciences.be/portal/app_dev.php/oaipmh/?verb=ListIdentifiers&metadataPrefix=oai_dc

4. List of published resources (complete records) ;
   https://darwin.naturalsciences.be/portal/app_dev.php/oaipmh/?verb=ListRecords&metadataPrefix=oai_dc

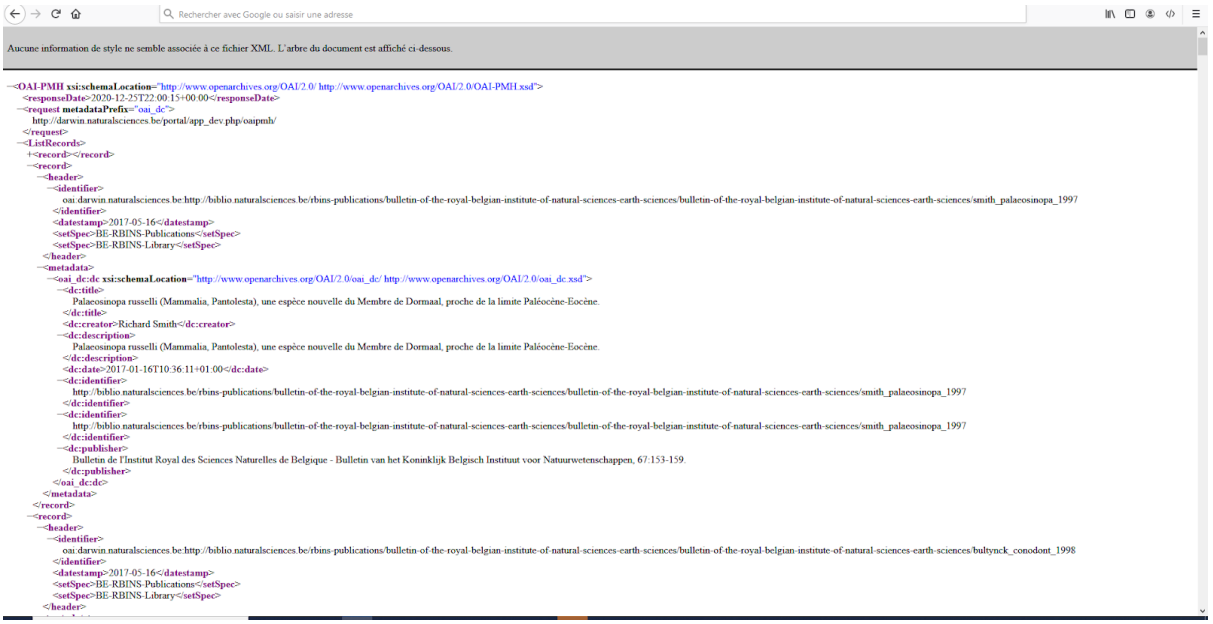


Figure 15. Overview of an OAI-PMH “get records query”

The OAI-PMH of the NaturalHeritage portal currently stores the scientific publication of RBINS staff that are currently stored in MARS. Data are harvested by the means of a Python script mapping the data from MARS (in RESt JSON format) to the JSON structure of the ElasticSearch index

4.5.4 The final user interface

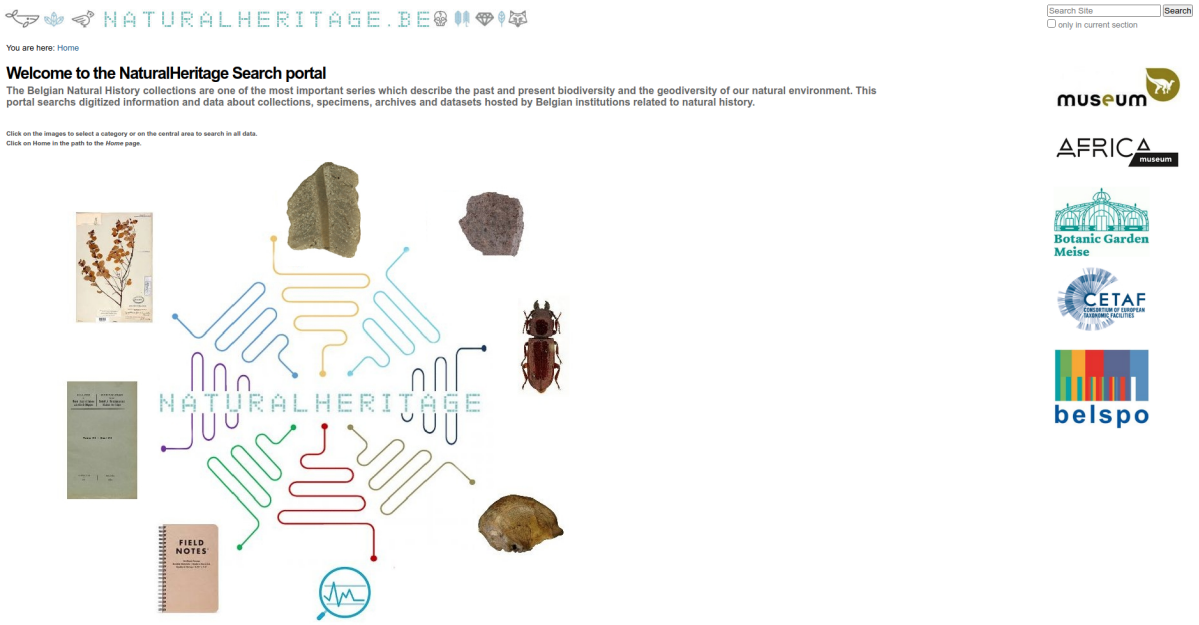
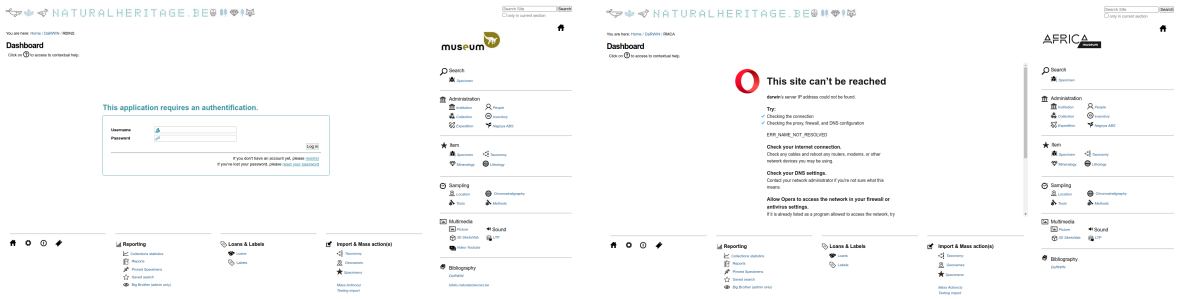


Figure 16. The NaturalHeritage Home page

The user interface was developed in Plone with all out of the box functionalities. The NaturalHeritage portal allows searching in the RBINS, RMCA and APM digitized collections and in the bibliographic references hosted by the Plone RBINS servers.
The chosen architecture allows the inclusion of new data sets in the portal.

The icons situated on the right menu link to the Collection Management system of the institutions. For RBINS and RMCA, it also provides the new DaRWIN interface as described in the Annexe 6.
We tested the internet access with DaRWIN RBINS server and intranet access with RMCA DaRWIN server.



RBINS DaRWIN with internet

RMCA DaRWIN with intranet

Figure 17. The NaturalHeritage DaRWIN Home pages

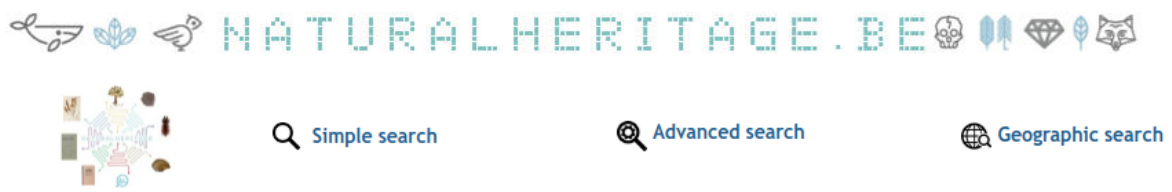


Figure 18. The NaturalHeritage search menu

The details of the search options are described at 4.2 Data Model.
Clicking on one of the image in the central menu provide access to a preset search e.g. in the paleontology collections

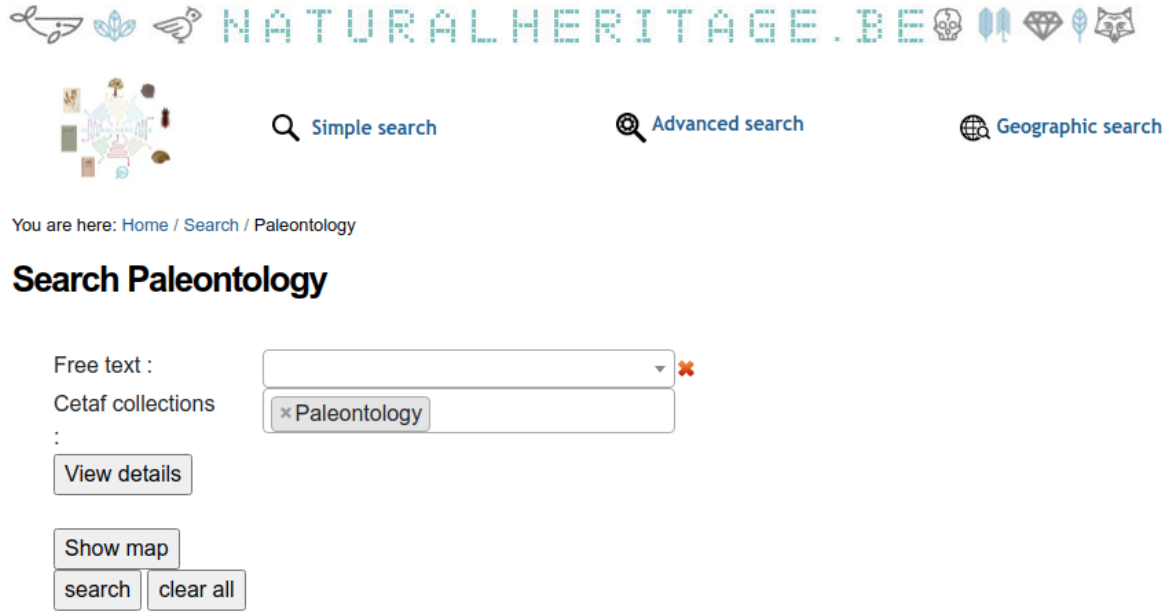


Figure 19. The NaturalHeritage search menu with preset collection

5. RECOMMENDATIONS

The portal and indexes are up and functional. Their development has represented a very valuable opportunity for scientists and IT engineers to get more familiar with the emerging NoSQL and Big-Data technologies and programming paradigms. The portal played the role of a direct inciting factor enabling each partner institution to gradually adapt their existing collection management systems to open them up and expose publicly available data on the Internet, by using sound technical standards, such as REST API. It greatly improves scalability of these existing systems, and forms the technical backbone of a platform valorizing their data on the Internet, and introduces a level of technical convergence between the contributing systems.

One of the technical lessons learned is that such an infrastructure can deal with a large amount of data in a reasonable amount of time. It took 6 hours to load in the portal the 1 475 982 records published by Meise Botanical Garden, with a server having 16 GB of RAM. However, the fastest solution was not the REST API, but the upload of static Darwin Core Archive files. A certain balance had to be found between the interoperability between data interfaces and performance. The index allowed us to successfully determine it, but in an empirical way, after having first tried a solution with the API.

Similarly, the project helped to determine a good balance between data cleaning operation occurring before the publishing of data, and the development of powerful free text analyzer working efficiently on less standardized and/or less controlled data, if the graphical search interface is well-designed and provide a single access-point to browse, compare and sort the date produced by search operations. The development effort shifts then from data cleaning operation to the development and parametrization of the search engine itself. The search engine itself can be enriched by external authoritative resources (for instance, boundaries of countries and ecological areas coming from Open Street Map, Natural Earth of GeoNames that are made available in the search interface). This implies a consequent development effort, but which can be specific, while data cleaning data standardization tasks have to be routinely repeated. Besides, it leaves the original data unaltered and eases the tracking of their historic evolution.
These two approaches are not mutually exclusive, the search engine can be considered as a preparatory step for later data-cleaning tasks. The definition of a clean and neat graphical interface displaying faceted (or aggregated) results becomes then a key factor. We defined in the project a fully hierarchical tree structure allowing a quick analysis of data where outliers values are immediately displayed.
The limiting factor is that the NoSQL technologies currently offer less possibility to export file data and reports than SQL systems. The free version of Kibana has limited functionality when it comes to export data. A sensible approach would be to build custom-made scripts, in Python or PHP, converting data from indexes into flat tab-delimited data, that can be re-imported into third-party data-cleaning and visualization tools. We noticed that the Python API to connect ElasticSearch is often simpler and terser than the native JSON query language of ElasticSearch.

A third result is the fact that powerful and fast search engines, associated to a well-designed interface (with fast autocomplete and an hierarchical representation of facets), able to parse and classify information in natural language, allow a bottom-up approach that can significantly reduce the cleaning and standardization efforts of values and keywords upstream. In certain cases, it can help avoid the introduction of costly and time-consuming controlled vocabularies, while keeping the data in their original form. The fact that ElasticSearch includes a stemmer for English words, handling conjugated verbs and plural forms, is very interesting in this respect. However, this approach is possible if we consider the index as a centralized aggregator with a dedicated search interface, and not as a part of linked data federated networks.

One of the possible future developments on this infrastructure could be to expand the English stemmers with different languages (e.g. English and the three national languages of Belgium), while minimizing the storage space used on disk. The French stemmer of ElasticSearch can recognize the masculine and feminine forms of nouns. ElasticSearch can deal with several stemmers simultaneously. It is worth mentioning that the effort to align and use a consistent classification (for instance in classification keywords) is not directly visible in the structure of the index itself, but in the parameters of the scripts that crawl data sources and convert their data into JSON. The structure and constraints on the data are therefore less self-documented than with relational databases, and these scripts have to be documented and modified cautiously.

Another interesting aspect of the index, and its related portal, is that their search interface is partially dynamic, able to adapt itself to the structure (especially the WHERE/WHO/WHAT subcategories) if they are modified and expanded, while still offering powerful autocomplete

lists to guide the user and tools to make complex queries (such as switches between Boolean OR and AND associations). The whole system can actually be considered as a successful attempt to bridge technical query languages with concepts and reasoning that are closer to natural language.

As we decided that the relation between the index and the original description of the scientific object was an URL linking back to the original Internet resource, the index indirectly made us more accustomed with key-concepts of semantic web, like stable and permanent URIs, and enhanced the level of integration of the partner institutions inside of European data networks having a semantic-web aspect (like CETAF stable IDs, DISSCO architecture and Elvis). The index is also compliant with several standards for scientific data, such as IIIF (as a client) and WKT (*Well Known Text*) for geographical coordinates, that are natively handled by ElasticSearch. This allows linking it to other GIS systems, though this link consists of the data rather than the query API.

Finally, as the portal has been developed in a recent version of Symfony (version 3), it also represented an opportunity to get more familiar with modern layered IT architectures, and put into practice key object oriented principles (interfaces and heritage) that enabled it to integrate external components quite rapidly. The OAI-PMH interface for the bibliographical part of the index reused code made open-source by the company Naoned, but which was originally intended for relational databases. The usage of object-oriented interfaces rather than directly implemented classes allowed the developers to fastly adapt the existing code to NoSQL technologies. This is not really a research result, but a sound practice that increases the level of integration and convergence of the contributing institutions with the good engineering practices and *de facto* standards of the scientific community, when developing their internal and external data architecture. Another indirect effect of the project is that technical staff working on the project become more familiar with GitHub and concepts related to versioning, which is a widely-used system inside of the scientific community when it comes to the maintenance, documentation and publishing of code. This also guarantees the conservation of the code in the longer run.

This was a situation justifying the usage of a PHP framework. Frameworks have a steep learning curve that may slow down the development of the project at earlier phases. But their standardized and layered architecture in three parts (Model, View and controller) allows, on the medium-run, to easily extend the application with external and reusable components, especially if they are documented and published on repositories such as GitHub. It should therefore be interesting to publish the scientific applications developed within the framework of nationally-funded projects on GitHub, to improve their visibility, and to keep track of developed code and its documentation. This platform provides standardised standards and tools to document and handle IT medium to large scale projects. However, GitHub has become de facto the main (if not only) platform to publish Open-Source projects, and this monopolistic situation could undermine the viability of Open-Source projects in the longer run, as many scientific projects are maybe too dependent on it. We would recommend using both GitHub and in-house project management service on the Intranet scientific institutions to guarantee that the code and technical resources developed within BELSPO-funded projects are preserved on the longer-run.

Symfony, as a PHP framework, was therefore a sensible choice when it comes to building a scalable application, even if the strict layered architecture can be perceived as an obstacle making the development cycle longer. This layered architecture ultimately helps to integrate

external components, and improves the reliability of the system by separating the code in smaller modules, with a limited length, communicating between themselves by higher-level interfaces. They also enable several developers to work on the same code, easing the translation of the staff changes. This is typically an Object-Oriented approach.
On an engineering point of view, we would advise to organize framework bundles according to business domain (i.e from the perspective of humans specialized in a specific task or domain, like collections, publications etc…) instead of too technical functional subdivisions that are internal to these domains. While detailed functional subdivision makes the code more compact and may speed-up development in the earlier stage of the project, more abstract and generic modules based on real-world tasks ease documentation and integration of external components and API on the longer run.

6. DISSEMINATION AND VALORISATION

The codebase of the index and portal have been published on the GitHub repository of the RBINS , giving a public visibility to the project, and allowing the reuse of the code and architecture in future projects. It has also been linked to the naturalheritage.be portal developed at RBINS, in Zope format.

The project has also been presented in a poster during the TDWG/BiodiverityNext conference in Leiden in October 2019, whose abstract has been published in Pensoft.

Finally, we also used the lessons and technical knowledge gained to contribute the website to the development of the CETAF website and the European ELVIS project, to build a registry of natural history institutions and collections. ElasticSearch is used as an intermediate broker, bridging several data sources (the MARS collection passport and the CETAF website developed in WordPress) and exposing their data to the public through a REST API. The code of this development is also published in the GitHub repository of the project.

7. PUBLICATIONS

Theeten F, Adam M, Vandenberghe T, Dillen M, Semal P, Scory S, Herpers J, Van den Spiegel D, Mergen P, Smirnova L, Engledow H, Casino A, Gödderz K (2019) *NaturalHeritage: Bridging Belgian natural history collections. Biodiversity Information Science and Standards 3: e37854.* https://doi.org/10.3897/biss.3.37854 (Poster and conference abstract)

The code of the portal has been published on the GitHub of RBINS, at the following address: https://github.com/naturalsciences/natural_heritage_search_engine

8. ACKNOWLEDGEMENTS

A substantial part of the concepts used to develop the data model of the portal have been inspired from the Balat Portal developed at KIK-IRPA.
We would like to thank the Nanoed company, in Nantes (France), who allowed us to reuse their OAI-PMH Symfony library in the project, and even supported the possibility to fork it.

Authors : Franck Theeten & Patrick Semal