EPJ Web of Conferences **295**, 05006 (2024) https://doi.org/10.1051/epjconf/202429505006
CHEP 2023

Glance Search Library

Gabriel José Souza e Silva[1,*], *Carlos Henrique* Ferreira Brito Filho[1,**], and *Gloria* Corti[2,***] *Joel* Closier[2,****]

[1]Universidade Federal do Rio De Janeiro, COPPE/EE/IF, Brazil
[2]European Organization for Nuclear Research, Switzerland

Abstract. The LHCb experiment is one of the 4 large LHC experiments at CERN. With more than 1500 members and tens of thousands of assets, the Collaboration requires systems that allow the extraction of data from many databases according to some very specific criteria. In LHCb there are 4 production web applications responsible for managing members and institutes, tracking assets and their current status, presenting radiological information about the cavern, and supporting the management of cables. A common requirement shared across all these systems is to allow searching information based on logical expressions. Therefore, in order to avoid rework, the Glance Search Library was created with the goal of providing components for applications to deploy frontend search interfaces capable of generating standardized queries based on users' input, and backend utility functions that compile such queries into a SQL clause. The Glance Search Library is split into 2 smaller libraries maintained in different GitLab repositories. The first one only contains Vue components and JavaScript modules and, in LHCb, it is included as a dependency of the SPAs (Single Page Applications). The second is a PHP Object-Oriented library, mainly used by REST APIs that are required to expose large amounts of data stored in their relational databases. This separation provides greater flexibility and more agile deployments. It also enables lighter applications with no graphical interface to build command line tools solely on top of the backend classes and predefined queries.

1 Introduction

The LHCb experiment [1] is one of the particle physics detector experiments at CERN's Large Hadron Collider [2]. It is a specialized experiment initially designed to study the differences between matter and antimatter through precision measurements of particles that contain a *b* quark. The experiment has made significant contributions to the field of particle physics, including the discovery of new particles and precision measurements of CP violation, i.e asymmetric behaviour between particles and anti-particles. In order to support the organizational aspects of the collaboration and the detector operation, Web systems for asset and membership management, workflow tracking, radiological surveys are deployed.

*e-mail: gabriel.jss@cern.ch
**e-mail: carlos.brito@cern.ch
***e-mail: gloria.corti@cern.ch
****e-mail: joel.closier@cern.ch

The Glance Team is a twenty-year-old software development group initiated in 2003 at UFRJ for the ATLAS Experiment [3]. Nowadays, Glance has 12 developers affiliated with UFRJ that collaborate with ATLAS, LHCb, and ALICE with a technical coordinator at CERN and a project responsible at UFRJ. Experimental representatives from ALICE [4], ATLAS and LHCb complete the management of the project. The team also has members from Università degli Studi di Udine and LIP-Laboratório de Instrumentação e Física Experimental de Partículas.

In LHCb, Glance provides four systems. The first, **Membership**, allows to manage the collaboration members and their affiliations alongside with participations for institutes associated with the experiment. **LBEMS** is an asset management tool: it is used to monitor information such as price, location, physical details and radiation measurements of the assets that compose the LHCb detector. **Cables** has a similar purpose as LBEMS specifically for cables and requests for their installation. **RP Survey**, which is the most recent, is a system to support radiological surveys in the experimental cavern with the goal of documenting radiation levels for easy use in safety and regulatory matters.

In order to increase productivity and standardize the software development processes, the Glance team created FENCE [5]: an in-house PHP Web framework to develop systems for the experiments. FENCE takes a JSON configuration file as input and builds webpages based on the settings defined in this file, reducing the amount of code developers have to write and ultimately allowing the final users and product managers to customize the applications. One of its most powerful tools is the FENCE Super Search: a tool built in FENCE to create advanced search interfaces. It was adopted in LHCb and used to construct the first versions of the Membership and LBEMS systems. However, FENCE-based applications had limited customization options, lacked documentation and presented high coupling between the web interfaces, database connection and business logic. Due to these limitations, the team started a collective effort to refactor the applications to a more modern software stack and in this context, the Glance Search Library was created to replace FENCE's Super Search.

2 The older Fence Super Search

The FENCE Super Search is a set of classes used to create advanced search interfaces split into two web pages (views). The first is called Search Workspace [5] and allows users to compose a structured search criterion using logic operators. The second view is responsible for presenting the results in a tabular view according to a set of parameters provided in a JSON file. In the query workspace, users add circles (nodes) that are horizontally connected by **AND** operators and vertically connected by the operator **OR**. An example and its limitation are shown in Figure 1 and Figure 2 which are an attempt to **search for all cables that start in point A and end in point C plus all cables that start in point B and end in point D**. In Figure 1 the query is composed through dragging and dropping nodes, and in Figure 2 results are displayed in a table. For the search under discussion, 4 statements can be defined:

$$\begin{aligned} &s_1 : cable\_start = pointA, s_2 : cable\_end = pointC, \\ &s_3 : cable\_start = pointB, s_4 : cable\_end = pointD \end{aligned} \tag{1}$$

and an expression that would bring the correct data set is:

$$query_1 = (s_1 \wedge s_2) \vee (s_3 \wedge s_4) \tag{2}$$

However, the existing system architecture presents a constraint: $query_1$ is unfeasible due to the system's limitation to only allow horizontal connections using the AND operator and

vertical connections using the OR. This restriction points to a notable rigidity in the FENCE system's design.

In response to this constraint, users frequently adopt an alternative query, shown in Figure 1, to meet the search criteria:

$$query_2 = (s_1 \vee s_3) \wedge (s_2 \vee s_4) \tag{3}$$

Upon detailed examination of $query_2$, it becomes evident that this formulation would yield results that do not align with the intended search filter. For example, scenarios where both s_1 and s_4 are *true* would be included. The new search implementation was designed with the goal to circumvent this and other existing limitations.

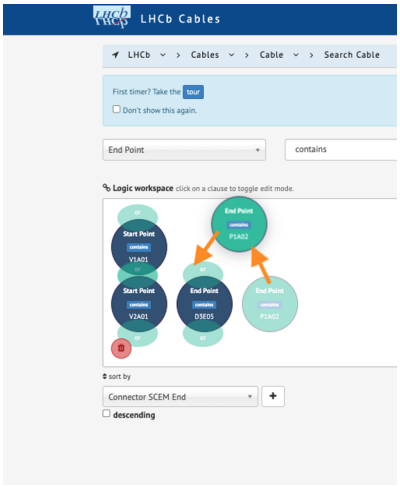

Figure 1: Query workspace

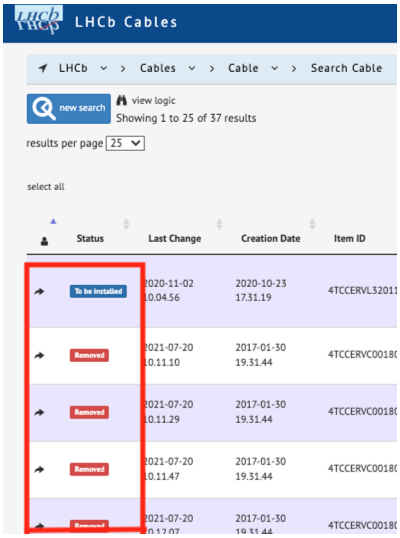

Figure 2: Fence search results

In the presentation view, the table behavior and style are also limited by the options provided in the JSON configuration file: while in most cases this is sufficient, it becomes a major issue for **dynamic content** and complex behavior. The pagination system also does not work as expected: if a result set had a total of 500 entries split into 5 pages, downloading the search results, text filtering them, and sorting results would only take the first page into consideration. Last, the save search feature, only works in the browser's local storage. FENCE's Super Search also lacked an API to expose that data in a manner that is agnostic to the data reader and did not offer any sort of preloaded/cached searches. However, even with its limitations, the FENCE Super Search was a core functionality not only in LHCb's Membership, LBEMS, and Cables systems but also for other systems in ATLAS and ALICE, having thousands of users. Hence, it had to be replaced carefully, the new solution providing the same features and addressing its shortcomings.

3 Glance Search Library

The Glance Search Library was developed to address the issues mentioned in the FENCE Super Search and provide its missing features. It is part of a broader effort to rewrite the applications, moving away from FENCE and using a more modern open-source software

stack more aligned to industry standards and with a larger community contributing to their development and maintenance. The library is composed of two code repositories: one for the backend functionality, and another for the frontend web components.

3.1 Architectural changes

The main points to be addressed in the new architectural proposal were the high coupling and limited customization in the FENCE systems. The solution chosen was a combination of the Layered and Hexagonal architectures, following a proposal from [6]. The applications were divided into 3 layers. The innermost layer is the application **Domain**, where all business rules are defined in PHP classes. The mid-layer called **Application**, exposes the system use cases and orchestrates domain classes to perform actions. The outermost **Infrastructure** layer connects the application use cases to external services such as databases and REST interfaces. The frontend is completely independent of the rest of the application, having access to the domain information through REST APIs.

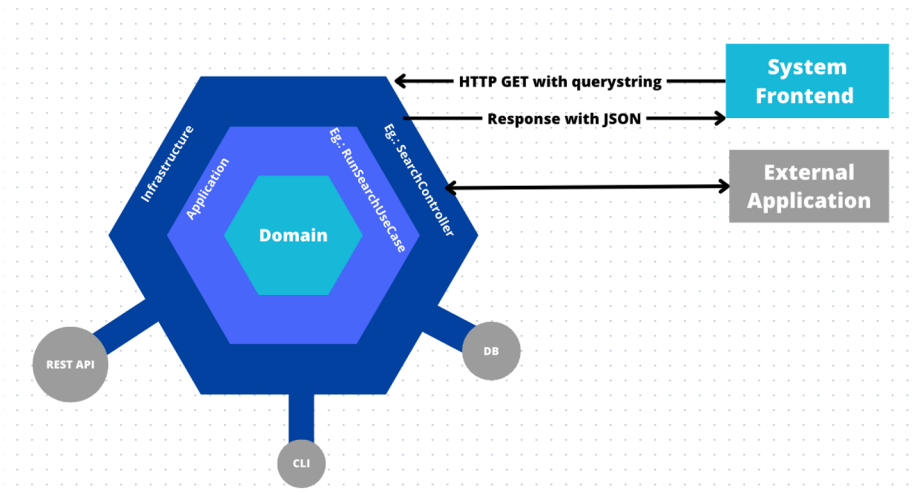

Figure 3: Application architecture organization

3.2 Glance Query Language and backend query translation

The systems' server side uses the Glance Search Library to translate a query string into a SQL filter creating a *WHERE* clause according to a set of parameters defined in a JSON configuration file. It differs from the FENCE's Super Search by limiting the scope of the JSON configuration file, being only used to map query string elements to database columns and caching settings, while the frontend interfaces do not depend on this file. Analyzing the same cables search example as in section 3 to find *all cables that start in point A and end in point C plus all cables that start in point B and end in point D*, now the option capable of generating the correct set of results ($query_1$) can be composed. This is because the query composition using the Glance Query Language (GQL) is agnostic to any user interface and its

possible limitations. The GQL elements will be introduced based on the following queries:

$$query_1 \equiv queryString_1 : \text{(start\_point = PointA } \textbf{AND} \text{ end\_point = PointC)} \\ \textbf{OR} \text{ (start\_point = PointB } \textbf{AND} \text{ end\_point = PointD)}$$

$$query_2 \equiv queryString_2 : \text{(start\_point = PointA } \textbf{OR} \text{ start\_point = PointB)} \\ \textbf{AND} \text{ (end\_point = PointC } \textbf{OR} \text{ end\_point = PointD)}$$

The elements in the queries above are categorized in Table 1.

Element	Category	Identifier
Start point	**Search Field**	f_1
End point	**Search Field**	f_2
=	**Search Operator**	o_1
AND	**Search Conjunction**	AND
OR	**Search Conjunction**	OR
Point A, B, C, D	**Search Value**	v_1, v_2, v_3, v_4
(	**Grouping Mark**	(
)	**Grouping Mark**	)

Table 1: Search elements

A **Search Statement** is a combination of a Search Field, Operator, and Value. Four statements are created from the queries above where "⌢" represents concatenation: $s_1 : f_1 \frown o_1 \frown v_1$, $s_2 : f_2 \frown o_1 \frown v_3$, $s_3 : f_1 \frown o_1 \frown v_2$, and $s_4 : f_2 \frown o_1 \frown v_4$. The query strings can now be rewritten as a function of the statements: $queryString_1$: $(s_1 \wedge s_2) \vee (s_3 \wedge s_4)$ and $queryString_2$: $(s_1 \vee s_3) \wedge (s_2 \vee s_4)$. GQL is designed to be decoupled from interface and infrastructure and their limitations, meaning that both queries above are valid, solving the aforementioned FENCE issue where $queryString_1$ cannot be created.

The categories defined in Table 1, are part of the Glance Search Library Domain PHP classes. These classes take a raw string as input and have the knowledge to convert themselves to a part of a SQL WHERE clause. They are orchestrated by the Application-layer using interfaces of the Infrastructure layer. The library's use cases are exposed by the following methods:

- `RunSearchWithAppliedFilters` - loads a list of entities filtered according to a query string based on GQL alongside with any search parameters such as pagination;
- `SaveSearch` - saves in a database a search layout, including the query and pagination;
- `GetSearchDetails` - lists all searches saved by a given user;
- `DeleteSearch` - deletes saved search.

Glance Search Library application-layer classes are used in the API controllers of LHCb Web applications and are loaded as dependencies using Composer, a Dependency Manager for PHP. A sequence of backend events to run a search includes:

1. An HTTP GET request with a "queryString" parameter reaching an API endpoint (e.g.: lbfence.cern.ch/lbems/cables/search);
2. Routing of the request to the infrastructure HTTP controller responsible for handling it;

3. Parsing incoming GET parameters and providing a configuration that maps search fields to database columns for LHCb\Search\Application\RunSearchWithFilters\RunSearchHandler.php;

4. Translation of the “queryString" into a SQL WHERE clause by the handler, using Domain classes and adding it to a base search defined in the configurations file;

5. Usage of the Infrastructure repository class to run the SQL and fetch results, exposed to the reader as a JSON.

Finally, having the REST API with the search endpoints decoupled from frontend interfaces, allows the deployment of automated jobs that run in the server executing the most common searches and caching their results for improved performance.

3.3 Frontend Web components

GQL provides a standardized way of searching and extracting information from the LHCb's Web applications. Power users can manually write it and download the search results from CLI tools, the frontend interfaces can use it to fetch data from APIs and external applications can also query LBEMS/Cables, Membership, and RP Survey for their own use. However, writing the raw query is not an option for the entire user base.

The client-side code is now completely separate from the server-side, following a Component-based Architecture with Vue.js allowing all frontend application code to be divided into independent, reusable, and easily tested components. Compared to FENCE's configuration-file-based approach, Web components provide greater flexibility for developers to choose which parts to include on their webpage interfaces. However, it could also lead to more boilerplate code being written when creating an advanced search web interface. To prevent this issue, all the key search components were wrapped in a SuperSearch.vue higher level component that is placed in a view (web page) which then passes to the wrapper a JavaScript object with all the required information to build the basic visualization elements such as a list of available Search Fields, Operators, and Values and the results columns to be displayed. A major differentiating factor compared to FENCE's Super Search is Vue's reactive two-way binding which ensures that any changes to the model are automatically reflected in the view, and conversely, any change in the view instantly updates the model. A concrete example would be a CablesSearchInterface.vue using the SuperSearch.vue component: it passes a JavaScript object to initially configure the search interface, but this object can now be mutated, and the interface will react to it. This is useful, for example, if a user triggers the insertion of a new column in the search results.

Vue.js provides a content distribution API known as *slots*, which is a powerful tool for building flexible and reusable components. *Slots* allow developers to compose components in a way that the component's content can be defined from outside the component itself. This gives developers more control over the design and structure of the component and how it can be reused. Slots can also have default content, which will be used when the parent component does not overwrite the slot. Vue's slot API provides the flexibility to handle interfaces with more complex behavior that FENCE never could.

The main search components are highlighted in Figure 4. All of them can be overwritten by the parent component in case the default behavior does not fit the application requirements. Figure 5, exemplifies a common use case of the application replacing the entire advanced search section with a select input for simpler web interfaces. The input content is then used by the frontend to generate a GQL query string that is used to fetch information from the system REST API. The sorting and download shortcomings the FENCE Super Search had

were addressed in the new search by removing the pagination parameters and re-fetching the information from the API in order to ensure the entire result set is downloaded and sorted. Saving a search is now possible using Vuex, a state management pattern library for Vue.js applications. It serves as a centralized store for all the components in an application, with rules ensuring that the state can only be mutated in a predictable way. The Vuex store state can then be persisted in the database through the SaveSearch endpoint exposed by the REST APIs.

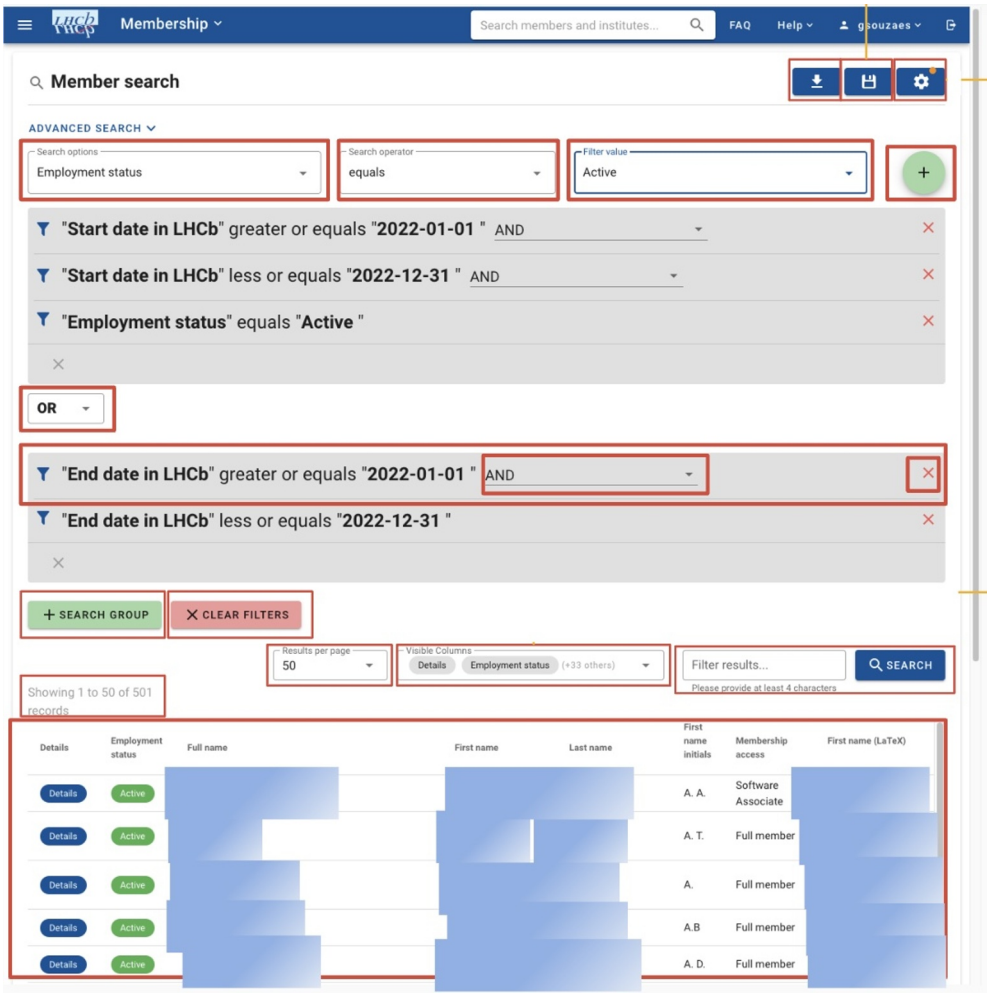

Figure 4: Super Search components highlighted

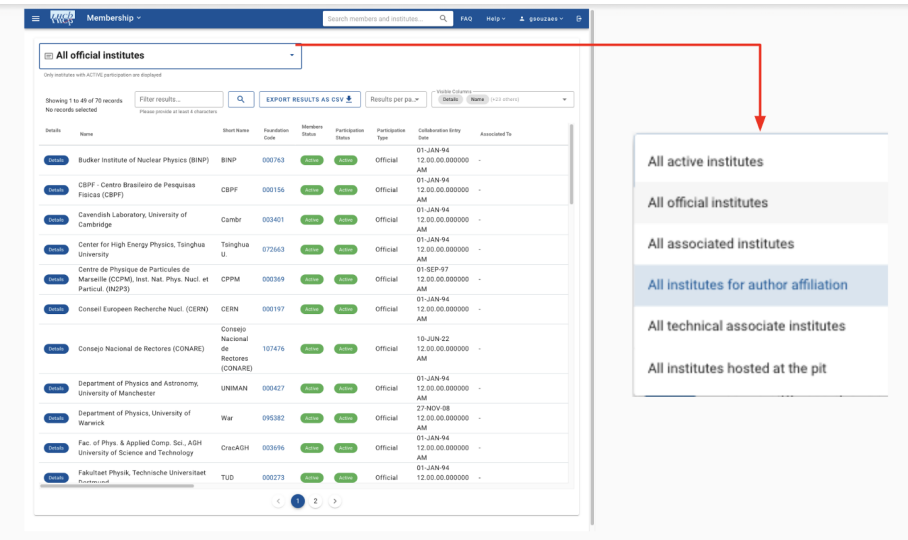

Figure 5: Simple search interface, replacing the advanced search slot by a simpler select input

4 Conclusion

The Glance Search Library is a successful and powerful replacement tool for the legacy FENCE Super Search. Currently, it powers 10 web interfaces in all LHCb systems provided by Glance. It also fixed FENCE pagination-related shortcomings and introduced new functionality such as the persistent saved searches. Another indicator of the new solution's success is its adoption by external applications that are now using it to populate their web interfaces, preventing data duplication inside LHCb. The main goal currently is to expand the usage of the library in the systems managed by Glance in ATLAS and ALICE.

References

[1] The LHCb Collaboration, A Augusto Alves Jr, L M Andrade Filho, A F Barbosa, I Bediaga, G Cernicchiaro, G Guerrer, H P Lima Jr, A A Machado, J Magnin et al. The LHCb Detector at the LHC. *J. Inst.* **3** (2008) S08005.

[2] Lyndon Evans and Philip Bryant. LHC Machine. *J. Inst.* **3** (2008) S08001.

[3] ATLAS collaboration. *The ATLAS Experiment at the CERN Large Hadron Collider* , JINST **3** (2008) S08003.

[4] ALICE collaboration. *The ALICE experiment at the CERN LHC*, JINST **3** (2008) S08002.

[5] B Lange, C Maidantchik, K Pommes, V Pavani, B Arosa, I Abreu, and on behalf of the ATLAS Collaboration. *An object-oriented approach to deploying highly configurable Web interfaces for the ATLAS experiment*. J. Phys.: Conf. Ser. **664** (2015) 062026.

[6] M Noback. *Advanced Web Application Architecture*. 2020. ISBN: 978-90-821201-6-5.

[7] C Maidantchik, F F Grael, K K Galvão, and K Pommès. *Glance project: a database retrieval mechanism for the ATLAS detector*. J. Phys.: Conf. Ser. **119** (2008) 042020.