The migration to a standardized architecture for developing systems on the Glance project

Carlos Henrique Ferreira Brito Filho[1,*], *Gabriel Jose* Souza e Silva[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 Glance project is responsible for over 20 systems across three CERN experiments: ALICE [1], ATLAS [2] and LHCb [3]. Students, engineers, physicists and technicians have been using systems designed and managed by Glance on a daily basis for 20 years. In order to produce quality products continuously, considering internal stakeholder's ever-evolving requests, there is a need for standardization. The adoption of such a standard had to take into account not only future developments but also legacy systems of the three experiments. These systems were created using an in-house built framework, and, as they scaled, became difficult to maintain due to the framework's lack of documentation and use of technologies that were becoming obsolete. Migrating them to a new architecture would mean speeding up the development process, avoiding rework and integrating CERN systems widely. Since a lot of the core functionalities of the systems are shared between them, both on the frontend and on the backend, the architecture had to assure modularity and reusability. In this architecture, the principles behind Hexagonal Architecture are followed and the systems' codebase is split into two applications: a JavaScript client and a REST backend server. The open-source framework Vue.js was chosen for the frontend. Its versatility, approachability and extended documentation made it the ideal tool for creating components that are reused throughout Glance applications. The backend uses PHP libraries created by the team to expose information through REST APIs both internally, allowing easier integration between the systems, and externally, introducing to users outside Glance information managed by the team.

1 Introduction

Glance [4] was originally a single system created as an abstraction layer to perform CRUD[1] operations over several databases. One of its main features was recognizing the structure of the selected database, allowing the user the creation of customized views for searching and inserting data. Glance was designed to be generic and was not able to cope with new

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

[1]Create, Read, Update, Delete

requirements, which often involved more specialized features. In order to fulfill these requirements, systems were created on top of Glance, using the tool to easily retrieve and alter data. This later part evolved into FENCE [5], an acronym for *Front-ENd ENgine for glaNCE*, a Web framework created with the purpose of unifying the interfaces for these systems. Even though FENCE was designed to be used alongside Glance, it became a framework encompassing both the backend and the frontend, which resulted in FENCE becoming a framework with highly coupled concerns. Being an in-house framework with limited customization combined with a lack in documentation and the aforementioned coupling concern made it very difficult to maintain. A new architecture for systems built with FENCE was adopted, moving away from the framework and focusing on creating maintainable applications with modern programming practices using free, well documented and open-source software.

2 Glance

Introduced in 2003, Glance simplified the access and integration of technical coordination data for the ATLAS experiment at CERN stored in diverse databases. It utilized XML markup language to describe interface and seamlessly connected to different storage technologies, including Oracle, MySQL, and Microsoft SQL Server.

Glance's key feature was its automatic recognition and presentation of the internal structure of databases, providing users with insights into data organization without requiring previous knowledge of the database itself. It allowed the creation of customized search and insertion interfaces, facilitating efficient data retrieval and manipulation across multiple sources.

Glance provided a user-friendly web interface for accessing these interfaces, making it versatile and easy to use. It offered flexibility in delivering retrieved information by allowing the user to export the results in various formats such as HTML, XML, and CSV.

However, the Glance's interfaces were not able to adapt and keep pace with the requirements as they evolved. For this reason a new framework had to be developed to make use of Glance's data retrieval capabilities and address its limitations on the presentation layer.

3 FENCE

FENCE (Frontend ENgine for glaNCE) was the framework developed by the team with the goal of facilitating the generation of content for user interfaces, and constitutes the first attempt at creating a standard across the three CERN experiments (ATLAS, ALICE and LHCb) where Glance is used for creating and managing web systems for collaboration and equipment data.

In its conception, FENCE would work with Glance serving as an API (Application Programming Interface) providing the data for the frontend, which would format it accordingly and display it to the user via a friendly web interface. JSON[2] configuration files were used to efficiently store and manage the system's rules. By loading these files into the engine, it dynamically created HTML responses for users' browsers. The advantage lies in its ability to modify an application's behavior by targeting configuration files rather than making changes directly to the source code. However, the core functionalities of a system were primarily handled on the server, ensuring secure manipulation of sensitive data and access control.

FENCE was designed as a framework to be easy to adopt by requiring minimal code to get a basic application setup up and running. This approach allowed for the efficient implementation of user-requested changes and the development of over 20 web systems in the three experiments. The trade-off with this design was the difficulty and time needed to implement

[2]JavaScript Object Notation

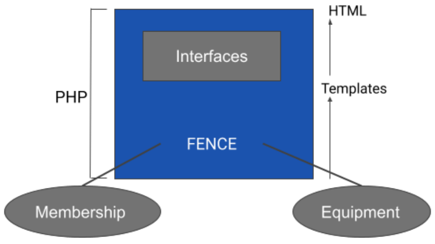

Figure 1: Structure of FENCE based applications

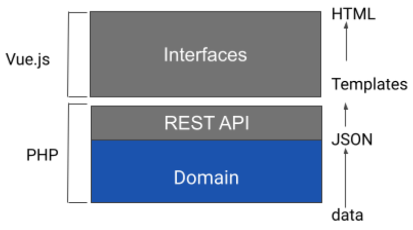

Figure 2: Structure of the new architecture applications

unique and specific features because of its rigidity and inflexibility. The lack of documentation for FENCE exacerbated the problem. Taking this into account, the development of the a newer architecture began.

4 Architecture

Systems developed using FENCE were created under a single repository with FENCE as one of its dependencies. Before the creation and adoption of the new architecture, the LHCb experiment had two systems in production: the Membership system which was taking case, among other things, for the administrative management of members and their employments, and the LHCb Equipment Management system, addressing the registration and tracing of equipment in the experimental cavern. The strucutre of the LHCb's system as implemented with FENCE is depicted on Figure 1.

The first step in developing the new architecture was defining these systems as separate applications with their own GitLab repositories. This restructuring followed the Hexagonal Architecture design alongside with Domain-Driven Design (DDD) principles [6] . Hexagonal Architecture promotes isolating the business logic from the infrastructure and technology-dependent code in a way that the core functionality of the application is not concerned by the details of how it interacts with the outside world. According to Noback [7], this separation of concerns goes in harmony with DDD principles, ensuring that all the core business logic has a single source of truth, which is the backend domain. In turn, this facilitates adapting to changing requirements and introduces flexibility since it allows adding or removing components as needed to scale the applications.

The implementation of this new architecture started by separating the applications into two parts: a JavaScript client and a REST backend server. Through a REST interface, data can be exposed to frontend clients, other CERN APIs and even external applications. Code sharing is made possible through bundles which are small Gitlab repositories that can be included as part of the application's dependencies. Since the tools developed by the team are internal open-source within CERN, these bundles allow better integration not only within the team but also CERN as a whole. The structure of the new architecture is depicted on Figure 2.

4.1 Backend

The REST backend server described utilizes the FENCE REST API (FRAPI) [8], an in-house tool that serves multiple purposes within the system. First and foremost, FRAPI handles the authorization process with CERN's services, ensuring secure and authenticated access to the server's endpoints. This integration with CERN's services enhances the server's security and allows for seamless authentication. However, FRAPI also allows the creation of public endpoints in which no authentication is required. This is an use case for accessing certain data from the collaboration that are public, such as an experiment's published papers or the list of associated institutes.

FRAPI is also responsible for building the server's endpoints. Similar to FENCE, it uses JSON configuration files for defining and managing the API configuration (such as the type of authentication and the API endpoints) allowing developers to easily create, modify, and maintain them. This streamlined approach to endpoint management simplifies the development process and promotes efficient code organization.

A logging mechanism is in place, designed to capture and record HTTP requests made to these endpoints. This logging functionality works on an user basis, saving the requests made on a file specific for each user. Logging these requests aids in debugging and allows for security audits.

In summary, FRAPI offers a comprehensive solution for handling authorization, building endpoints, and logging HTTP requests. Its use allows developers to focus on implementing the server's core functionality while benefiting from streamlined development processes.

4.2 Frontend

The choice of the JavaScript client was carefully considered. It had to fulfill specific requirements, including being open-source, well-documented, and capable of creating reusable components, since many core functionalities were shared across Glance systems. The technologies chosen were Vue.js [9] along with two key libraries: Vuetify [10] and Vuex [11].

4.2.1 Vue.js

Vue.js, an open-source JavaScript framework, provides the foundation for building the client-side applications. It has many libraries that enhance and simplify the development process. It also provides comprehensive documentation, allowing the developers to understand and implement the requested features. The framework allows the creation of components that could be easily reused throughout Glance systems, reducing code duplication, improving maintainability and familiarizing the end user with Glance interfaces.

4.2.2 Vuetify

Vuetify, an open-source material design component framework for Vue.js, offers a wide range of ready-to-use UI components. Its use further enabled the creation of the aforementioned reusable components, ensuring consistency and speeding up development.

4.2.3 Vuex

Vuex, a state management pattern library for Vue.js applications, is used in order to effectively manage state within the applications. It provides a centralized state management solution which handles complex data flows, ensuring that the state can only be mutated in a predictable fashion. The components' view is automatically synchronized, reflecting new values when the application state is mutated. Vuex's clear patterns allowed efficient organization and maintaining of the application's state.

5 Conclusion

In conclusion, the adoption of the new architecture marks a significant milestone in the evolution of the Glance project, with a clear focus on creating a standard across the experiments alongside with improving maintainability and code sharing.

In the LHCb context, the migration away from the FENCE framework is currently underway, with most of the legacy systems already migrated and efforts being made to complete the migration of the remaining one. Additionally, two new systems (Radiological Protection Survey system and Analysis Life Cycle Management system) are already in production and have been deployed using the new architecture, demonstrating its viability and independence from FENCE. The ultimate goal of this endeavor is to fully migrate away from FENCE and utilize the new architecture for all future systems.

This transition signifies a commitment to modernization and lays the foundation for enhanced performance, scalability, and maintainability in the evolution of the Glance project.

References

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

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

[3] LHCb collaboration. *The LHCb detector at the LHC*, JINST **3** (2008) S08005.

[4] 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.

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

[6] E Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. 2004. ISBN: 978-0-321-12521-7.

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

[8] *FRAPI: Product Requirements v1*. URL https://readthedocs.web.cern.ch/fence-docs/product-requirements/requirements-v1 [accessed 01-Jul-2023].

[9] *Vue.js - The Progressive JavaScript Framework*. URL https://www.vuejs.org/ [accessed 01-Jul-2023].

[10] *Vuetify - A Vue Component Framework*. URL https://vuetifyjs.com/en [accessed 01-Jul-2023].

[11] *What is Vuex?* URL https://www.vuex.vuejs.org/ [accessed 01-Jul-2023].