Jiskefet, a bookkeeping application for ALICE

Marten Teitsma[1,*], *Vasco Chibante* Barosso[2], *Pascal* Boeschoten[1],
and *Patrick* Hendriks[1]

[1]Amsterdam University of Applied Sciences
[2]CERN

Abstract. A new bookkeeping system called Jiskefet is being developed for
A Large Ion Collider Experiment (ALICE) during Long Shutdown 2, to be in
production until the end of LHC Run 4 (2029). Jiskefet unifies two function-
alities: a) gathering, storing and presenting metadata associated with the oper-
ations of the ALICE experiment and b) tracking the asynchronous processing
of the physics data. It will replace the existing ALICE Electronic Logbook and
AliMonitor, allowing for a technology refresh and the inclusion of new fea-
tures based on the experience collected during Run 1 and Run 2. The front
end leverages web technologies much in use nowadays such as TypeScript and
NodeJS and is adaptive to various clients such as tablets, mobile devices and
other screens. The back end includes an OpenAPI specification based REST
API and a relational database. This paper will describe the organization of the
work done by various student teams who work on Jiskefet in sequential and
parallel semesters and how continuity is guaranteed by using guidelines on cod-
ing, documentation and development. It will also describe the current status of
the development, the initial experience in detector stand-alone commissioning
setups and the future plans.

1 Introduction

During the LHC Long Shutdown 2 a renewal of the bookkeeping systems in place for AL-
ICE is envisioned. For this, a unified experience for shifters active in detector operations,
physicists searching the run catalogue to find appropriate runs and overall management of
the ALICE operation should be provided. Users access all metadata related to operational
activities, keep historical records of configurations used to operate the detector, execute per-
formance studies of detector operations, monitor the quality of the collected data, and produce
reports for operational teams and experiment management in a single place.

ALICE [1] is one of the four main detectors collecting collision data at the CERN LHC.
ALICE focuses on the study of quark–gluon plasma in heavy-ion collisions but also carries
out substantial measurements of proton–proton interactions. ALICE produces a lot of data of
interest to physicists who need to know the circumstances under which the data are gathered.
For this, metadata is produced and stored by the bookkeeping system. Secondly, the goal of
the bookkeeping system is to keep track of the data stored on the Grid and used for analysis
by end-users.

At the start of the ALICE operations in 2007 a bookkeeping system was developed and it
evolved with time during the LHC data-taking periods Run 1 (2009-2013) and Run 2 (2015-
2018) following the users needs. This system was based on the LAMP (Linux, Apache,

*e-mail: m.teitsma@hva.nl,

MySQL and PHP) software stack with a relational database, a web-based Graphical User Interface for the members of the ALICE collaboration, and bindings for machine to machine access [2]. The electronic logbook system thus far gathered informations from about 5,000 LHC fills, 280,000 data-taking runs, and comprises about 37 GB data from 195,000 log entries and 20,000 file attachments. For monitoring the data on the Grid a system called AliEn, later known as AliMonitor, was developed to present detailed information such as CPU and memory usage, open files and network traffic of each site offering services [3]. The Data Preparation Group, Physics Working Groups and ALICE physicists are the main users of AliMonitor. Since its deployment it has stored 12.9 million jobs, i.e. specified activities on data, amounting to 97 GB data.

In this paper we first present the methods for developing the application and the tools we used. Then the requirements of the system are described and the architecture of the system is shown. We finish with the current status, discussion, future work and conclusions.

2 Methods and tools

To gather the requirements for the new bookkeeping system we started with a series of interviews at CERN. The requirements are described in Section 3. As a method of development we used Scrum [4].

The development of Jiskefet is performed by several student teams. The main effort is done in Amsterdam at the Amsterdam University of Applied Sciences (AUAS). Several teams and interns involved in the Software for Science initiative work on the application consecutively. To work effectively it is important to transfer knowledge and skills in a transparent way. Documentation is very important together with the continuous involvement of staff. After the first release, student teams from a course of Web Development and the Moscow Polytechnic University were involved. The first team worked on a design for the GUI. The team from Moscow worked on the reporting functionality.

To organise the communication among the several teams we used regular video conferences, Slack, Telegram and the odd email. Visits to the different Institutes contributing to the project were also organised.

Git is used for software version control. The hosting of the code is done on GitHub [5] as well as a private repository on a GitLab instance. The formal documentation is edited in LaTeX. Less formal documentation is included in the repository as Markdown files. Coordination of the project is performed with the Jira online tool [6].

Testing is part of the continuous integration process for which we use Travis CI [7] and Jenkins [8]. The testing itself is done by Mocha [9] and Jest [10]. The linting is done by TSLint [11]. We use CodeCov [12] to determine how much of the code is being tested.

3 Requirements

To retrieve the requirements we interviewed several stakeholders of the system at the start of the project. We then categorised the findings in a System Requirements Specification according to IEEE Std 830-1998 [13]. In this specification the business goals, vision and scope are described. Several interfaces, functions and users are mentioned. Furthermore, the functional and non-functional requirements are summed up.

The functional requirements are categorised according to the large number of various users of the system. To name just a few: shifter, (sub system) run coordinator, manager, member of the physics board. Each user has its own specific requirements. Besides the existing functionalities of the electronic logbook system in place, several new requirements

were mentioned during the interviews such as: users should be able to make a log entry using a template to avoid incomparability, it should be possible to have a flexible tagging system for runs, a smart editor should be available, a search functionality should be in place.

The bookkeeping system shall function as part of a large information technology system. Several system attributes, which should lead to a friction-less embedding, were mentioned during the requirement engineering phase. The choice of tools and the software stack was dependent on several desiderata such as:

- familiarity with the software at CERN;

- minimisation of dependencies;

- maximisation of development community.

It was also important to use modern tools and software expected to remain in use for the entire time of the project.

Jiskefet has to be in production until 2029. Since during this period several students and members of the staff will independently be involved in the development and maintenance, sustainability of the application is of importance. An application can be characterised by its intrinsic sustainability, i.e. level of documentation, testing, readability, usage of third party libraries, usefulness and scalability. Its extrinsic sustainability is characterised by its availability, resourcefulness, level of community actions and relations, independence from infrastructure [14].

The availability of the system should be optimal and not be a reason for the detector to stop taking data. Each entry, i.e. input of data, should be logged and although the GUI and REST API are not essential the database should be highly available. Once in the repository, no data should be lost.

4 Architecture

The bookkeeping system used in ALICE during Run 1 and 2 used a web technology and the same requirement is valid for the new system. The web has evolved fast and so did the technology behind it. Users now have more seamless and responsive applications in the browser with the rising popularity of Single Page Application (SPA) frameworks and real time web apps developed in JavaScript. These techniques require strict separation of data and markup. The data is delivered and gathered by a RESTful back-end as can be seen in Figure 1. The environment of Jiskefet consists of a database, the detector, human users who directly interact with the system and users who interact not directly but use other applications to retrieve data or send requests.

The JavaScript technology stack is notorious for its many dependencies. To mitigate the number of dependencies, we carefully assessed common frameworks and libraries for developing Jiskefet. These frameworks where selected based on a popularity ranking and on the features they packed. The ALICE team was working on a collection of frameworks with a CERN wrapper around it called WebUI [15]. Whilst WebUI is merely a collection of relatively lightweight nodeJS [16] modules i.e. express.js [17], mySql, the volatility of WebUI at that time made us choose for more popular frameworks, i.e. Nest.js [18] and Mithril [19].

Mithril is a modern client-side Javascript framework for building Single Page Applications. It's small, fast and provides routing and several utilities out of the box. The benefit of choosing Mithril over for example React is that the ALICE people are familiar with it because it is included in WebUI.

One of the main non-functional requirements is to develop a REST-API as a back-end. RESTful Web services are constrained to be a client-server architecture, stateless, cacheable,
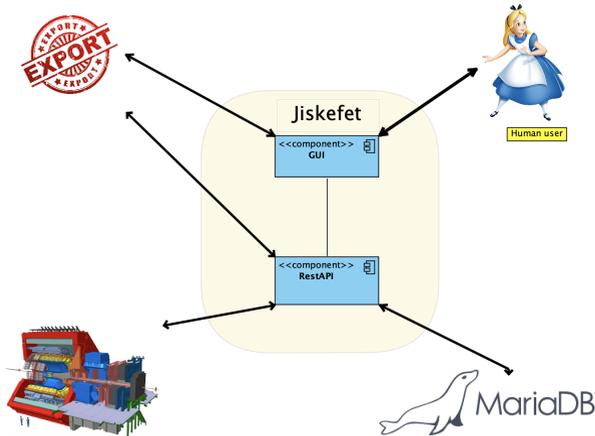
Figure 1. Environment of Jiskefet

layered and have a uniform interface. Advantages are good performance, scalability, simplicity, modifiability, etc [20]. The back-end is implemented with the Nest.js web framework. One of the main advantages is the Swagger [21] functionality built in. Swagger helps us and other developers to model, document and test the API.

In the data model several entities are discerned. Entities concerning the work done at ALICE are an 'Activity', i.e. a set of well-defined tasks during a finite time period. An example is a 'Run'. An 'Activity' aggregates configurations, selected tasks, global statistics etc. A 'Run' can be of different types such as 'Detector Calibration Run' or 'Global Run'. A 'Run' is formally defined as 'a synchronous data readout and processing Activity in the O^2 farm with a specific and well-defined configuration'. It normally ranges from a few minutes to tens of hours and provides a unique identifier for the data set generated at the end of the synchronous data flow. Another 'Activity' is a 'Reconstruction Pass' which is defined as 'asynchronous data processing Activity in the O^2 farm or the Grid with a specific and well-defined configuration'. Runs identify the data generated at the end of the synchronous data flow. Reconstruction Passes will have as input the data generated in either a Run or a previous Reconstruction Pass. Runs can be executed either during an LHC Fill (as for most global physics data taking cases) or in between LHC Fills (for cosmics data taking, calibration and/or tests). An LHC Fill can have many Runs but a Run only belongs to an LHC Fill. Reconstruction Passes are indirectly connected to LHC Fills via the Run.

The database is implemented using MariaDB [27]. Familiarity with its predecessor MySql and ease of use were the main reasons for this choice. For the users the central entity is the log entry. Log Entries are generic and can be connected to any of the other entities.

Jiskefet consists of several packages as shown in Figure 2. On a CentOS platform a Nginx Web Server [22], consisting of two Node.JS applications, is deployed. For the User Interface we use Mithril as a framework, Webpack [23] to bundle the modules and decrease the dependencies and TypeScript [24] as a way to make JavaScript more consistent. On the front-end we use HTML5, CSS and JavaScript. For the back-end we use NestJs and Typeorm [25] as an interface with the database. For loadbalancing PM2 [26] is used. As a database we use MariaDB. To streamline the work flow and ease testability we use Ansible [28] as automated deployment framework.
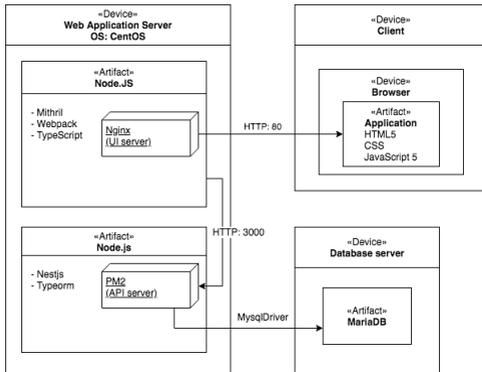
Figure 2. Deployment diagram

5 Current Status and Discussion

A prototype has been developed. The prototype was to be distributed among the teams performing detector commissioning of the ALICE subdetectors, but it wasn't because of a number of mismatches in the Ansible playbook. In web development the use of external libraries is normal practice. This creates an important drawback making it obligatory to do maintenance continuously. Particularly NodeJS has a great number of dependencies. Together with other packages it makes the current version too much dependent on insecure sources. Although thoroughly selected packages, there is always the risk of depending on outdated or obsolete packages in the long run. A plan is needed to cover those risks.

Communication between the several teams and stakeholders in this project is of great importance. Insufficient feedback on what has been produced, shifting requirements or misunderstandings, among other problems make a project of this size and timespan complex.

Important decisions about the use of specific tools were made when at ALICE the development of a web framework was not yet matured. The asynchronous development and use of different frameworks is causing now some disturbance in the working together of the systems. This and other problems make it seemingly unavoidable that the current prototype has to be refactored.

6 Future Developments

To comply with the non-functional requirements of the user we will refactor Jiskefet to include WebUI. Furthermore, we are now going to work on the second goal which is the management of the computation of the data on the Grid.

7 Conclusions

The first goal of our project was to create a log functionality for ALICE. We now have a first version which has to be refactored. The development of such a fairly complex system for a remote client by several student teams from different countries is not a trivial project. A lot of time has to be spent on communication to make sure everybody involved knows of what the projectpartner is talking about.

References

[1] K. Aamodt, A.A. Quintana, R. Achenbach, S. Acounis, D. Adamová, C. Adler, M. Aggarwal, F. Agnese, G.A. Rinella, Z. Ahammed et al., *The ALICE experiment at the CERN LHC* (IOP Publishing, 2008), Vol. 3, p. S08002

[2] V. Altini, F. Carena, W. Carena, S. Chapeland, V.C. Barroso, F. Costa, R. Divià, U. Fuchs, I. Makhlyueva, F. Roukoutakis et al., *The ALICE electronic logbook*, in *Journal of Physics: Conference Series* (IOP Publishing, 2010), Vol. 219, p. 022027

[3] S. Bagnasco, L. Betev, P. Buncic, F. Carminati, C. Cirstoiu, C. Grigoras, A. Hayrapetyan, A. Harutyunyan, A. Peters, P. Saiz, *AliEn: ALICE environment on the GRID*, in *Journal of Physics: Conference Series* (IOP Publishing, 2008), Vol. 119, p. 062012

[4] *Scrum alliance*, https://www.scrumalliance.org/about-scrum/overview, accessed: March 2020

[5] *Jiskefet*, https://github.com/SoftwareForScience, accessed: March 2020

[6] *Jira software*, https://www.atlassian.com/software/jira, accessed: March 2020

[7] *Travis ci*, https://travis-ci.org, accessed: March 2020

[8] *Jenkins*, https://jenkins.io, accessed: March 2020

[9] *Mocha*, https://mochajs.org, accessed: March 2020

[10] *Jest*, https://jestjs.io, accessed: March 2020

[11] *Tslint*, https://github.com/palantir/tslint, accessed: March 2020

[12] *Codecov*, https://codecov.io, accessed: March 2020

[13] IEEE, *830-1998 - ieee recommended practice for software requirements specifications* (2009), `"https://standards.ieee.org/standard/830-1998.html"`

[14] M.R. de Souza, R. Haines, C. Jay, *Defining sustainability through developers' eyes: Recommendations from an interview study*, in *Proceedings of the 2nd Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE 2014)* (2014)

[15] *Webui*, `https://github.com/AliceO2Group/WebUi`, accessed: March 2020

[16] *Node.js*, `https://nodejs.org/en/`, accessed: March 2020

[17] *Express.js*, `https://expressjs.com`, accessed: March 2020

[18] *Nest.js*, `https://nestjs.com`, accessed: March 2020

[19] *Mithril*, `https://mithril.js.org`, accessed: March 2020

[20] R.T. Fielding, R.N. Taylor, *Architectural styles and the design of network-based software architectures*, Vol. 7 (University of California, Irvine Irvine, USA, 2000)

[21] *Swagger*, `https://swagger.io`, accessed: March 2020

[22] *Nginx*, `https://nginx.org`, accessed: March 2020

[23] *Webpack*, `https://webpack.js.org`, accessed: March 2020

[24] *Typescript*, `https://www.typescriptlang.org`, accessed: March 2020

[25] *Typeorm*, `https://opencollective.com/typeorm`, accessed: March 2020

[26] *Pm2*, `https://www.npmjs.com/package/pm2`, accessed: March 2020

[27] *Mariadb*, `https://mariadb.org`, accessed: March 2020

[28] *Ansible*, `https://www.ansible.com`, accessed: March 2020