Abstract—The advantages of web applications have already got attention in major physical research facilities like Canadian Light Source (CLS). It is the accessibility of web applications that makes them preferred to native desktop application in some experimental control scenarios. This short paper presents two web applications that were mainly developed at CLS — Science Studio for remote access and collaboration of instruments and computation resources, and Logit for beamline experiment information management. These two applications represent two typical web applications. Science Studio is heavy-weight and provides a large spectrum of functionalities and has been developed by distributed teams for years. Logit is light-weight, and provides a very limited set of features and was delivered in a very short time. The architectural designs are discussed for both sides, and the lessons learned from them are discussed.

I. INTRODUCTION

Web technologies might seem not suited for experimental control tasks in accelerator research facilities like Canadian Light Source (CLS) because the latency of web applications compared to native real-time control applications. On the other side, web technologies do show strengths that cannot be found in native real-time applications such as:

- An application can be developed once, and then be easily deployed and updated on various operating systems for the sake of the ubiquitousness of web browsers.
- HTTP enables an application to work across network boundaries with firewall protection.

These advantages make web applications the top design option for the following scenarios:

- remote monitoring and control of physical instruments across network boundaries for a large number of users with different roles,
- near-real-time sharing of fresh data and on-demand utilization of computation resources, and
- easy access of experimental information on different platforms including mobile devices.

The web application solutions for such scenarios have been made even simpler with continuous development of web technologies like server push, HTML/CSS and numerous high quality open-source JavaScript libraries. However, this does not mean that a web application can be definitely successful in a research facility environment. We want to discuss and share the lessons we have learned from two web application projects within CLS.

II. SCIENCE STUDIO

Science Studio is a joint software project by Canadian Light Source (CLS), University of West Ontario (UWO) and IBM funded by CANARIE under the NEP program [1], [2]. The project delivered web applications for secure and role-based access to distributed experimental instruments and data. As well, Science Studio provides on-demand access to remote computing resources that are able to perform high performance processing of large data sets transferred on high-speed network. Science Studio has been used by scientists at CLS, UWO, Advanced Light Source (ALS) and Brazilian Synchrotron Light Laboratory (LNLS) [3]. The first phase of the Science Studio project, Remote Beamline Access (RBA) started in 2006 [4], and the third phase, Active Network Interchange for Scientific Experimentation (ANISE) [5], was officially closed in early 2012.

Science Studio was first targeted to support remote access of instruments on a couple of beamlines within CLS in the first phase. Support was added for access to experimental data and on-line collaboration in the second phase, as well as support for more diverse instruments running on different platforms. In the third phase, it supported remote computational resources for high performance processing of experimental data, and was able to exchange experimental meta information across multiple deployments across networks. Changing requirements have been common for the project from the beginning, which resulted in a continuously evolving architecture and implementation. When Science Studio had just one application for a set of instruments, the change was not difficult. However, at the end of the project, any changes touching the core data models or libraries caused pains for the distributed developing teams.

The architecture of Science Studio is shown in Figure 1. The components on the bottom of the diagram including AD, NFS, Jetty service, DB, EPICS and ActiveMQ are within the protection of an intranet. Science Studio is a typical Java-based multitire web application. Besides the frequent changes of data model, other significant changes during the development include:

- to replace direct AD/LDAP authentication with CAS (Centralized Authentication Service), and
- to provide RESTful interfaces to manipulate SS experiment meta-data across multiple SS deployments.

The first change was made to support SSO (single sign on) for the applications deployed on multiple SS deployments. The second change was made to decouple the dependency of a common data model among multiple SS deployment. The RESTful interface make it possible for a single SS deployment to update its data model, and allow other unchanged
deployments to still exchange data with it.

III. LOGIT

Logit is an internal project delivered by the CLS CID group for the BioMedical Imaging and Therapy (BMIT) beamline. The major goal is to provide a web-based application that can capture diverse experimental information on the beamline including instrument state and control parameters available as EPICS PV’s, image acquisition information hidden in image headers, and other information only available from the scientists and users on site by manual record. The application can serve two scenarios: 1) to collect all the information for user experiments for later reference and review, and 2) to record the state of the machine and beamline related to faults for diagnosis.

The project was delivered with an effort of about 10 developer-weeks. This time also includes initial testing and modifying some features. The application was designed in a service-oriented way such that the major functionalities are developed as services. Because each service was independent from others, the developer was able to release a service and its web interface and let the scientists test it while developing other services. This also makes the later integration of services quite straightforward. When the whole application was released for testing, most issues had already been fixed during the previous service testing cycles. The first release of Logit supported the capture of more than 100 PV’s, parsing 5 customized TIFF headers, and a form of more than 140 control elements.

The architecture of Logit was shown in Figure 2. Logit was a very light-weight web application with all services developed using node.js\(^1\) and its modules. Naturally, all the objects and messages are represented in JSON\(^2\). The developer originally wanted to use a no-SQL document-oriented database like CouchDB or MongoDB. The choice of database was changed to MySQL because the beamline staff had more experience with the use of a traditional SQL database. The object-relation mapping (ORM) was done with the help of Sequelize\(^3\). The image metadata parser was based on imageMagick\(^4\). The login service was implemented by Passport\(^5\). The user interface was developed with jQuery\(^6\) and Bootstrap\(^7\).

IV. LESSONS FROM SCIENCE STUDIO AND LOGIT

Science Studio and Logit can represent two very different web applications in a scientific experimental facility setting like CLS. Science Studio aimed to address almost every

---

\(^1\)See http://nodejs.org/
\(^2\)See http://www.json.org/
\(^3\)See http://www.sequelizejs.com/
\(^4\)See http://www.imagemagick.org/
\(^5\)See http://passportjs.org/
\(^6\)See http://jquery.com/
\(^7\)See http://twitter.github.com/bootstrap/
aspects that a web application can do. It was designed to be heavy-weight. It had been developed for several years. On the other side, Logit was developed for just one beamline, though it can be very easily extended for other beamlines or facilities. Its scope was small and the project spans only a short period. It is difficult to tell which project or application is more successful than the other. This paper aims to summarize the lessons that we learn from these two projects. These lessons are common for web applications and cover both heavy-weight and light-weight varieties.

A. Browser compatibility

Browser support might be the most tricky aspect of web applications. Even many mature commercial web applications have issues with browser compatibility. Science Studio removed the burden of browser compatibility by supporting only Firefox from the start of the project mainly because canvas was not supported by early versions of Internet Explorer (IE). That decision has saved lots of effort for developers to tune the web pages and JavaScript. In fact, there were still requirements for Science Studio to support other popular browsers like IE and Chrome.

When Logit was first developed, Firefox and Chrome are the browsers to be tested. However, the beamline staff indicated that IE should be supported because IE is the default browser on many Windows workstations on the beamline. Testing Logit against IE and making it IE-compatible used about 2 weeks of development time out of the total 10 weeks.

Browser compatibility will continue to be an issue for web applications from now on with the development of HTML5, CSS3 and growth of mobile devices. A general rule to achieve the best browser compatibility is to use the most popular web front-end frameworks and JavaScript libraries, and to avoid the testing features such frameworks or libraries provide. For example, we found BootStrap was much better for browser compatibility than jQuery UI components.

B. Version control

Source code version control is straightforward with so many mature tools’ support. The source version control tool used in Science Studio development was CVS at the beginning, and later was changed to Git with repository hosting and issue tracking provided by SourceForge. The tool used in Logit was Git. We also track production versions by MKS in CLS intranet.

The deployment version control is not that easy as source version control. Deployment of right versions of compiled libraries and third-party libraries on development, testing, and production servers took lots of efforts during the development of Science Studio. It got even more difficult when there were multiple SS deployments in various organizations. The deployment version control was made much easier in Logit. Because the JavaScript source does not need to be compiled before run by node.js virtual machine, the deployment version control was made exactly the same as source version control. The developer can easily update and roll back the versions in development, testing and production environments with Git. Even the version control of third-party libraries could be done the same with Node Packaged Modules (NPM)8. The versions of dependent node.js modules are explicitly specified in a JSON configuration file that is also version controlled by Git.

C. Development methodology

In both Science Studio and Logit projects, we had formally collected and documented requirements. However, sometime the requirements did not cover all the potential users when the application would be used by a large community like Science Studio. Such missing requirements resulted in missing features or features that were not useful for the user group. When the development team learned the negative feedbacks, it was too late because the project deadline approached. This

8See https://npmjs.org/
problem can be avoided by making the development process agile and fast. It very naturally suggests to adapt feature-driven or service-oriented design. Tools like MKS can help track emerging requirements and corresponding source and configuration changes. CLS is moving towards using MKS for an integrated management of development process from requirements to test cases.

The other common pitfall of Science Studio and Logit is the emphasis of data model. Because both applications have a database back-end, the development started with a comprehensive data model from the elicited requirements. We always assumed that the data model was designed so well that it would hardly be changed during the development. Lots of effort therefore had been spent on the discussion and refinement of those models. However, it turned out some part of the model was either never implemented in the application and most part of the model had been revised continuously. The use of advanced ORM tools can save developers’ effort in such cases, but it cannot help them to avoid designing models that are not suitable for either business requirements or integration requirements.

A thorough solution might be test-driven development. For web applications, that means the web user interfaces or programming interfaces should be developed and tested before the model was implemented and finalized. The development of test cases, either prototype web forms or cURL\textsuperscript{9} scripts, can potentially help the developer to better understand the use cases, which will eventually bring better data model that can stay unchanged longer.

V. Conclusion

We presented the design of two web applications — Science Studio and Logit — in this paper. Although web applications are currently not the major players in experimental physics facilities, we expect that more attention will be drawn to such applications with the development of more powerful web technologies and increased access to mobile devices. The lessons that we learned from the projects are discussed, and they will help us to avoid these pitfalls in future projects and to deliver better web applications for the scientific research community.

Acknowledgment

The research work described in this paper was performed at the Canadian Light Source, which is supported by CANARIE, NSERC, NRC, CIHR, and the University of Saskatchewan.

References


\textsuperscript{9}See http://curl.haxx.se/