ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP120

HOW EMBRACING A COMMON TECH STACK CAN IMPROVE THE LEGACY SOFTWARE MIGRATION EXPERIENCE

C. D. Burgoyne, C. R. Albiston, R. G. Beeler, M. Fedorov, J. J. Mello, E. R. Pernice, M. Shor
Lawrence Livermore National Laboratory, Livermore, USA

Abstract

Over the last several years, the National Ignition Facility (NIF), the world's largest and most energetic laser, has regularly conducted approximately 400 shots per year. Each experiment is defined by up to 48 unique pulse shapes, with each pulse shape potentially having thousands of configurable data points. The importance of accurately representing small changes in pulse shape, illustrated by the historic ignition experiment in December 2022, highlights the necessity for pulse designers at NIF to have access to robust, easy to use, and accurate design software that can integrate with the existing and future ecosystem of software at NIF. To develop and maintain this type of complex software, the Shot Data Systems (SDS) group has recently embraced leveraging a common set of recommended technologies and frameworks for software development across their suite of applications. This paper will detail SDS's experience migrating an existing legacy Java Swing-based pulse shape editor into a modern web application leveraging technologies recommended by the common tech stack, including Spring Boot, TypeScript, React and Docker with Kubernetes, as well as discuss how embracing a common set of technologies influenced the migration path, improved the developer experience, and how it will benefit the extensibility and maintainability of the application for years to come.

INTRODUCTION

On December 5, 2022, the National Ignition Facility (NIF), the world's largest and most energetic laser, made history by achieving ignition in a laboratory setting for the first time [1]. This milestone was made possible in part by various software applications which are instrumental in the NIF's ability to regularly and efficiently conduct up to 400 experiments per year. Given the speed of technological innovations in software, and with NIF now in its second decade of operations, several of these software systems require various upgrades or rewrites.

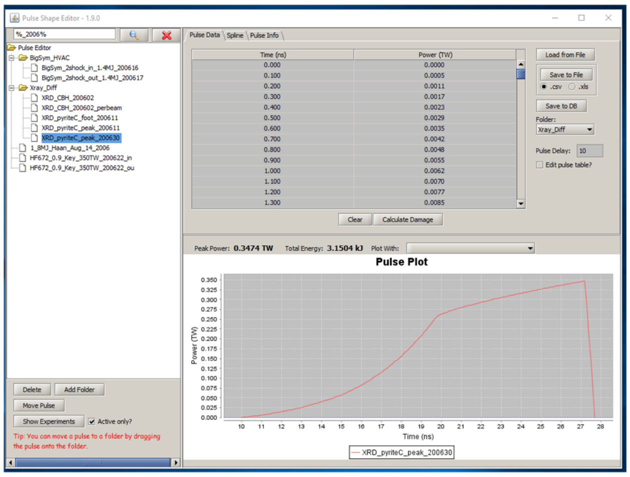
Recently, a team within the Shot Data Systems (SDS) group worked on rewriting one of these legacy tools, a Java Swing based desktop application called Pulse Shape Editor (PSE), to a new single-page, modern, web-application known as Pulse Shape Tool (PST). This application, whose purpose is to create and define the laser pulses used on NIF, is instrumental in the shot process, empowering pulse designers to configure potentially thousands of individual pulse points per experiment. The importance of PST functioning efficiently and accurately was illustrated by the successful December 2022 shot, where small adjustments to the energy and timing of laser pulses, achieved in part through manipulation of pulse point and spline point data in PST, played a role in reaching ignition [2].

With limited resources and many disparate applications to develop and maintain, when choosing a migration path for PST it was critical for SDS developers to choose a set of technologies, also known as a technology stack or tech stack, that was modern yet leveraged the knowledge already available within the team. With so many existing technologies available, and more becoming available nearly every day, choosing an ideal tech stack for a given application and development team can be challenging and time-consuming. Ultimately, it was chosen to develop PST using a tech stack that was already being utilized among a subset of other SDS applications, resulting in benefits such as the ability to leverage existing developer knowledge of technologies that were also well-supported by the developer community at large.

REQUIREMENTS

The original PSE was designed as a stand-alone Java Swing application (Fig. 1). Users would download PSE onto their desktops prior to use, yet running the application still required an Internet connection to use all the features due to reliance on a remote database connection for storing and retrieving pulse data. Besides using outdated technology containing vulnerabilities, many of PSE's limitations were due to architectural decisions made by the original developers. For example, since PSE was a desktop application, it was difficult to extend it to dynamically integrate with other applications in real-time, a feature increasingly expected in modern applications. Furthermore, these same architectural decisions made it difficult to maintain in general. After thorough examination of the PSE code, it was decided performing an in-place upgrade would not provide enough added value and would be needlessly difficult and time consuming to complete. Instead, the decision was made to migrate the functionality to a new application.

Key functionalities in PSE included the ability to edit, save and plot pulse and spline points, simple data validation, and the ability to import and export pulse data to and from external files and the database. Users required these key functionalities preserved in the migration to PST. Users also requested additional modern features be integrated into the new application such as the ability for users to modify and plot multiple pulses simultaneously, support for multiple users editing the same pulses concurrently, complex data integrity checks, per pulse permission management, and complex integrations with other applications, all while ensuring the software remained easy for developers to maintain and extend as required. These requirements were to be implemented as a single-page web-application, in-line with many of SDS's other modernized applications.



Time (ns)	Power (TW)
0.000	0.0000
0.100	0.0005
0.200	0.0011
0.300	0.0017
0.400	0.0023
0.500	0.0029
0.600	0.0035
0.700	0.0042
0.800	0.0048
0.900	0.0055
1.000	0.0062
1.100	0.0070
1.200	0.0077
1.300	0.0085

(● .csv ○ .xls)
Folder: Xray_Diff
Pulse Delay: 10
☐ Edit pulse table?
☑ Active only?
Tip: You can move a pulse to a folder by dragging the pulse onto the folder.
Peak Power: 0.3474 TW Total Energy: 3.1504 kJ Plot With:

Figure 1: Legacy Pulse Shape Editor (PSE) application.

CHOOSING A TECHNOLOGY STACK

Given the current state of the development tools and technology ecosystem, the requirements for PST could have potentially been implemented in a nearly infinite number of ways. Even given the limitation that PST must be a web application, one is not short on choices for implementation of the web GUI. For example, there are many popular web frameworks alone, three of which include React, Angular and Vue.js. All three of these frameworks can be used to create modern, high-quality, single page web applications, and all are well used and supported, having tens to hundreds of millions of downloads per year as seen in Fig. 2. When also taking into consideration the choices necessary for web components such as complex tables, server frameworks, and databases, it quickly becomes clear that a strategy for finalizing choices is necessary.

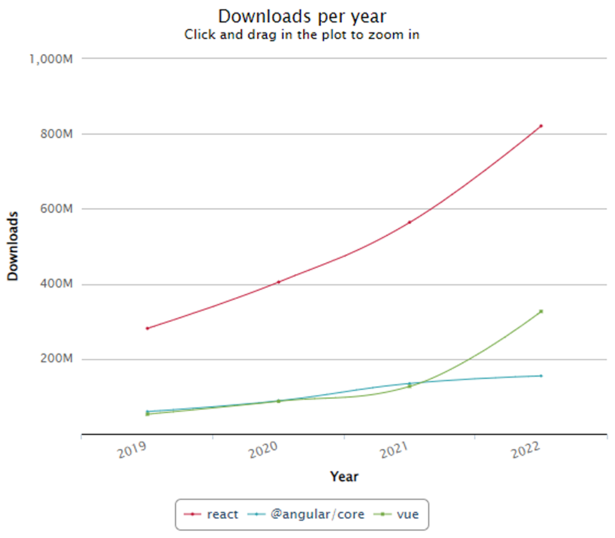


Figure 2: Chart showing npm download trends for three popular web frameworks, React, Angular, and Vue, illustrating how each has millions of downloads per year [3].

A Strategy for Navigating Numerous Choices

One strategy for navigating the ever-expanding ecosystem of programming languages, frameworks, libraries, and other technologies available for application development are the idea of endorsed tech stacks, which may also be known as recommended tech stacks or common tech stacks. This concept can mean different things to different groups, but in general, endorsed tech stacks aim to give developers a guarantee that their technology choices will be supported by the team or organization and that the team or organization has a pool of developer resources with at least some level of knowledge of the technologies in the stack. Some large companies, such as Meta, already put some form of this idea into practice [4]. Some of the goals of an organization adopting an endorsed tech stack may be to save developer bandwidth when beginning development on a new application by eliminating some of the research and guesswork out of the starting point, to improve maintainability with more predictable code that more developers in the organization can quickly jump into, or to simply improve the developer experience. For groups adopting recommended tech stacks, an important consideration is that the suggested tech stacks should not be too prescriptive, but more-so act as a starting point. There will be times when specific requirements may not fit the recommended technologies. In these cases, there should be a process for developers to make changes based on their discretion to develop the best product and meet mission objectives.

Applying Technology Stack Strategy to PST

When development of PST began in 2020, SDS had recently decided to leverage the concept of recommended tech stacks for future applications and modernization projects. Tech stack endorsements were based on the recommendations of an internal architectural review board. The initial tech stacks were devised by the board based on modern technologies that had already been successfully used across several applications in SDS. A non-exhaustive list of examples of initial endorsed options included (all options additionally involved using containers with Docker on Kubernetes):

- Spring Boot Application Stack: Java with Spring Boot and Oracle database
- Node Application Stack: NodeJS with TypeScript and NestJS using Oracle database
- Java Tomcat Stack: Java, Spring, and Oracle database
- Web Client Stack: TypeScript, React or Angular, and an appropriate Kendo component library

The specific stack chosen for the PSE to PST migration project was ultimately chosen based on past developer experience and familiarity with the suggested technologies. For the backend, developers chose the Spring Boot Application Stack, leveraging Java 11+ on Spring Boot 5.2 with an Oracle database. For the Web Client Stack, or frontend stack, developers chose to use React 16 with Typescript and KendoReact as a component provider, mainly for grids. As suggested, both the front and backend would utilize containerization on Docker with Kubernetes for orchestration.

MODIFYING THE TECHNOLOGY STACK

Although most of the recommendations from the endorsed tech stack were used for PST without modifications, a couple adjustments were made when it came to providers for pre-build React components.

Switching KendoReact for AG Grid React

KendoReact (Kendo) [5] was the initial solution recommended for grids by the tech stack. Kendo, created by Progress Telerik, is a commercial JavaScript library that provides a wide variety of UI components for use in React-based applications. Being a commercial library, it requires a license to create production-ready applications. The primary Kendo component that was planned for use in PST was the react-data-grid, a performant and customizable component for displaying small or large sets of data in a tabular format [6]. This component would have been used throughout the application to render tables containing data such as pulse points, spline points and data integrity summaries. However, based on experience with other SDS applications using Kendo, PST developers had concerns. Primarily, the desired customizations typical for SDS applications, customizations that would once again be required for PST, were difficult with this technology. This led developers to explore other data grids to determine if another solution may fit the specific needs of PST more appropriately.

Two other options, PrimeReact [7] and AG Grid React (AG Grid) [8] were examined. Like Kendo, PrimeReact is a set of customizable, open-source UI components for React created by PrimeTek. However, unlike Kendo, PrimeReact does not require a commercial license to develop or deploy an application to production. On the other hand, AG Grid is not a component library as the other options, but a customizable, stand-alone, high-performance grid. AG Grid has a free community edition that fits many use cases; however, it also offers an enterprise version with advanced features. In short, while Kendo and PrimeReact are libraries of components containing a grid component, AG Grid is a stand-alone, specialized, fully featured grid component.

After prototyping all three options, developers ultimately decided that AG Grid would be the best option not only due to the its widespread use (Fig. 3), excellent official documentation, and ease of necessary customizations, but because through prototyping, it was discovered that AG Grid had most of the features necessary for implementation of PST requirements built-in and would therefore require minimal additional customization versus the competing packages that had been evaluated.

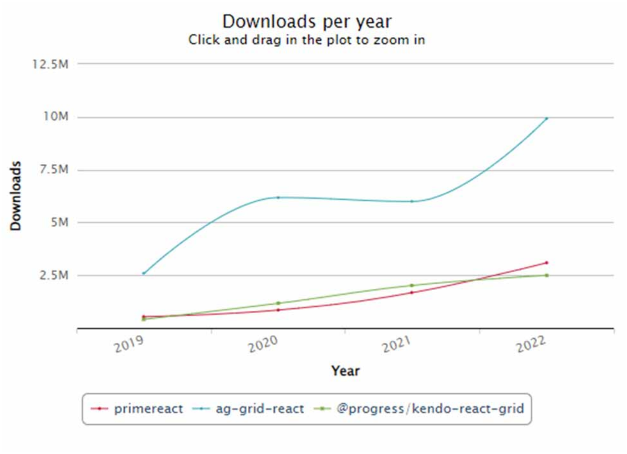


Figure 3: Chart showing npm downloads of PrimeReact, AG Grid, and Kendo's grid component, illustrating relative popularity of AG Grid compared to others [9].

Addition of ReCharts

One of the key requirements for PST was the ability to plot pulse points in real-time. Without the use of Kendo or any other component library for React, a charting library was required to be added to the tech stack. When choosing a charting package, as with the grid package selection process, the PST developers desired a solution that enjoyed widespread adoption, had excellent official documentation, required minimal customization, and was easy to customize where necessary.

Several popular React charting libraries were evaluated. These included recharts [10], a composable charting library for React built on D3.js; react-chartjs-2 [11], a React wrapper of the JavaScript-based Chart.js library; nivo [12], a set of data visualization components for React based on D3.js; and victory [13], a set of modular charting and visualization components for React. Ultimately, after an initial research period, prototypes were created using both recharts and nivo. After prototyping, it was decided that recharts best suited the needs of PST due to its relatively larger number of users (Fig. 4), the minimal customization required, and ease of necessary customizations.

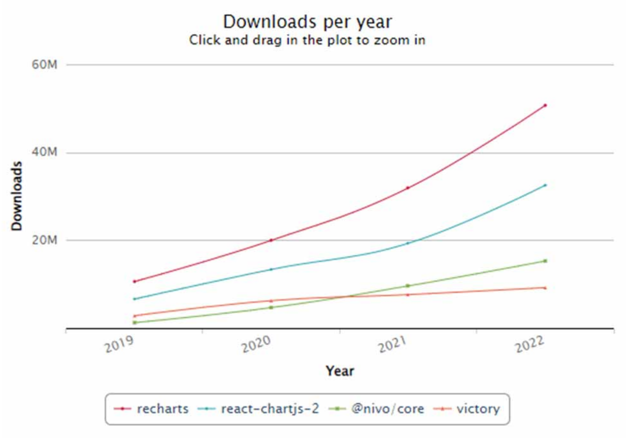
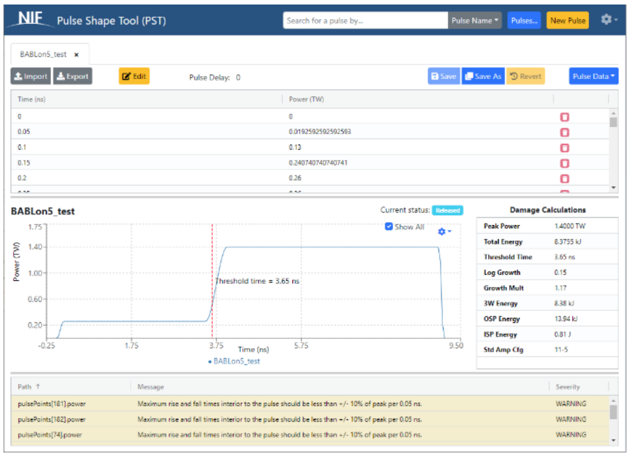


Figure 4: Chart showing npm downloads of recharts, react-chartsjs-2, @nivo/core (nivo), and victory, illustrating relative popularity of Recharts compared to others [14].

LESSONS LEARNED

The bootstrapping, development and subsequent maintenance periods for PST have been an overall positive experience. Making use of a tech stack that shared many aspects with other applications in the SDS software library allowed for efficient bootstrapping and development periods with the project beginning in late 2020 and being delivered to users by early 2022 (Fig. 5). Thus far, PST has enjoyed a relatively low effort maintenance period with few bugs compared to what developers had experienced in past applications. However, even though there were plenty of benefits with using this approach, there are still some potential drawbacks to consider before adopting a recommended or endorsed tech stack for application development.



Pulse Delay: 0

Time (ns)	Power (TW)
0	0
0.05	0.0192592592592593
0.1	0.13
0.15	0.240740740740741
0.2	0.26

BABLon5_test

Damage Calculations

Peak Power	1.4000 TW
Total Energy	8.3755 kJ
Threshold Time	3.65 ns
Log Growth	0.15
Growth Mult	1.17
3W Energy	8.38 kJ
OSP Energy	13.94 kJ
ISP Energy	0.81 J
Std Amp Cfg	11-5

Path ↑	Message	Severity
pulsePoints[181].power	Maximum rise and fall times interior to the pulse should be less than +/- 10% of peak per 0.05 ns.	WARNING
pulsePoints[182].power	Maximum rise and fall times interior to the pulse should be less than +/- 10% of peak per 0.05 ns.	WARNING
pulsePoints[74].power	Maximum rise and fall times interior to the pulse should be less than +/- 10% of peak per 0.05 ns.	WARNING

Figure 5: PSE was successfully migrated to PST, pictured here, and delivered to users in early 2022.

Benefits

There are many benefits that may be enjoyed by a team or organization from adopting recommended tech stacks. Some potential benefits are outlined in the following.

Reduced project startup times Developers found that the time to get the project up and running was greatly reduced due to not requiring an extensive research and prototyping period. Although this phase was in no way eliminated entirely, the process was found to be more efficient and allowed for more focus on the details, as an outline for the big picture was already provided by the tech stack.

Simplified development of internal tools for future use Adopting a recommended tech stack has allowed SDS developers to create internal component libraries and starter projects to further save future development time. For example, while developing PST, developers were able to create a front-end starter project, or seed project, that can be cloned and modified as needed when bootstrapping future projects.

Simple to create a common look-and-feel across various applications Having similar tech stacks, especially on the front end, and the ability to have common code in shared, downloadable packages, makes it trivial to set up a new application with the same look and feel as others. This provides users with a predictable, cohesive experience across applications, improving the overall user experience.

Streamlined update process When using similar technologies and versions across projects, as with a common tech stack, developers found that keeping projects up to date can be simpler. For example, if a library has a breaking change between versions, adapting to this change only needs to be learned once and the knowledge can then be repeatedly applied across all effected applications versus needing to research and apply fixes for unrelated breaking changes across a set of differing technologies.

Developer experience may be more enjoyable, resulting in more satisfied developers Developer experience may be improved for several reasons. For example, if the tech stack encourages the use of technologies with widespread adoption, developers can spend less time researching answers to development issues and spend more time coding due to a large online user community. In addition, developer's flexibility is increased as they are empowered to easily move between projects as they will already be familiar with the technologies used. This same familiarity may also result in fewer bugs due to greater knowledge resulting in a better experience for developers as well as stakeholders.

Drawbacks

Although there are many benefits to using recommended tech stacks, there are still potential pitfalls to consider. A couple potential issues that may arise are outlined as follows.

Updating for breaking changes can be time consuming Although a common tech stack can make maintaining a set of projects easier, it can result in updates becoming a more time-consuming process that affects many applications simultaneously.

Developers need to be proactive at research Use of a recommended tech stack still requires research and thought from developers. It is easy when pressed for time to accept a tech stack at face value and make it work with project requirements even if the tech stack is not ideal which may result in a sub-par result. Instead, developers still need to do research on technologies versus project requirements and make adaptations and changes where necessary to best meet mission objectives.

CURRENT STATUS AND FUTURE WORK

With the use of recommended tech stacks, the development experience can be improved to include shorter project bootstrapping times and a shorter development period while maintaining high quality code, as was seen through the migration process from PSE to PST. Currently into the maintenance phase, the benefits of PST's tech stack choices are still evident with few bug reports and easy to implement fixes. Developers have worked to consistently keep PST technologies up to date and in-line with other applications in the SDS portfolio.

The development team for PST is currently in the process of updating to React 18 and Java 17 (soon to be Java 21), an effort that is being performed in parallel to these updates in several other SDS applications. Future plans include updating Spring Boot to the latest version and updating authentication to use OAuth 2.0. As technologies continue to update and release new versions, developers will continue to update PST regularly to keep everything up to date and reduce the chance of PST being exposed to critical vulnerabilities in the future.

Currently, SDS is continuing to migrate additional legacy applications either to new applications or using in-place migrations as necessary. We are continuing to use the idea of a recommended tech stack, taking advantage of seed projects and internal component libraries to bootstrap migrations, which has continued to produce a positive experience and quick turnaround time for developers and stakeholders alike.

Beginning in fall of 2023, SDS established a revised architectural review board. The review board will advise other SDS developers on revised tech stack recommendations and their benefits, as well as inform and influence on other up-to-date development best practices.

ACKNOWLEDGEMENT

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. IM Release #LLNL-CONF-854378.

REFERENCES

[1] Celebrating the Milestone of Ignition, `https://www.llnl.gov/news/ignition`

[2] Star Power: Blazing the path to fusion ignition, `https://www.llnl.gov/article/25706/star-power-blazing-path-fusion-ignition`

[3] npm-stat: Download statistics for packages react, vue, @angular/core, `https://npm-stat.com/charts.html?package=react&package=vue&package=%40angular%2Fcore&from=2019-01-01&to=2022-12-31`

[4] Programming languages endorsed for server-side use at Meta, `https://engineering.fb.com/2022/07/27/developer-tools/programming-languages-endorsed-for-server-side-use-at-meta/`

[5] KendoReact, `https://www.telerik.com/kendo-react-ui`

[6] KendoReact: React Data Grid (Table), `https://www.telerik.com/kendo-react-ui/grid`

[7] PrimeReact, `https://primereact.org/`

[8] AG Grid React Documentation, `https://www.ag-grid.com/react-data-grid/`

[9] npm-stat: Download statistics for packages @progress/kendo-react-grid, ag-grid-react, primereact, `https://npm-stat.com/charts.html?package=%40progress%2Fkendo-react-grid&package=ag-grid-react&package=primereact&from=2019-01-01&to=2022-12-31`

[10] Recharts, `https://recharts.org/en-US/`

[11] react-chartjs-2, `https://react-chartjs-2.js.org/`

[12] nivo, `https://nivo.rocks/`

[13] Victory, `https://formidable.com/open-source/victory/`

[14] npm-stat: Download statistics for packages recharts, @nivo/core, react-chartjs-2, victory, `https://npm-stat.com/charts.html?package=recharts&package=%40nivo%2Fcore&package=react-chartjs-2&package=victory&from=2019-01-01&to=2022-12-31`