ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP22

USING REACT FOR WEB-BASED GRAPHICAL USER APPLICATIONS FOR ACCELERATOR CONTROLS

R. Bacher, J. Szczesny
Deutsches Elektronen-Synchrotron DESY, Germany

Abstract

Today, control applications need to run on a variety of different operating systems, including Windows, Linux, and Mac OS, but also Android and iOS. Programming languages like Java have tried to solve this problem in the past by providing a common runtime environment. However, this approach is insufficient or even unavailable for mobile devices such as tablets and smartphones. Another problem is the different form factors of mobile and desktop devices, which makes it difficult to develop portable applications. One way out of this dilemma is to use standard web technologies (HTML5, CSS3, and JavaScript) to implement applications that run in the browser, which is available for all platforms. Modern JavaScript web application frameworks combined with JavaScript graphics libraries such as D3 are suitable for building both very simple and very complex web-based graphical user applications. This paper reports on the status and issues that we encountered in our current developments with React.

LEGACY: WEB2CTOOLKIT

At DESY, the development of web-based user applications for accelerator operation started 2005 during the conversion of the high-energy booster synchrotron PETRAII into the synchrotron light source PETRAIII. In the following years, the so-called Web2cToolkit framework [1] was further implemented and its functionalities were extended step by step.

The Web2cToolkit is a collection of web services including

(1) *Web2c Synoptic Display Viewer*: Interactive synoptic live display to visualize and control accelerator or beam line equipment,

(2) *Web2c Archive Viewer*: Web form to request data from a control system archive storage and to display the retrieved data as a chart or table,

(3) *Web2c Messenger*: Interface to E-Mail, SMS and Twitter,

(4) *Web2c Logbook*: electronic logbook with auto-reporting capability,

(5) *Web2c Manager*: administrator's interface to configure and manage the toolkit,

(6) *Web2c Editor*: graphical editor to generate and configure synoptic displays, and

(7) *Web2c Gateway*: application programmer interface (HTTP-gateway) to all implemented control system interfaces.

Web2cToolkit is a framework for Web-based Rich Client Control System Applications. It provides a user-friendly look-and-feel and its usage does not require any specific programming skills. By design, the Web2cToolkit is platform independent. Its services are accessible through the HTTP protocol from every valid network address if not otherwise restricted. A secure single-sign-on user authentication and authorization procedure with encrypted password transmission is provided. Registered and so-called privileged users have more rights compared to ordinary users (read-only permission).

The Web 2.0 paradigms and technologies used include a Web server, a Web browser, HTML (HyperText Markup Language), CSS (Cascading Style Sheets) and AJAX (Asynchronous JavaScript And XML). The interactive graphical user interface pages are running in the client's Web browser. The interface is compatible with all major browser implementations including mobile versions. The Web2cToolkit services are provided by Java servlets running in the Web server's Java container. The communication between client and server is asynchronous. All third-party libraries used by the Web2cToolkit are open-source.

The Web2cToolkit provides interfaces to major accelerator and beam line control systems including TINE [2], DOOCS [3], EPICS [4] and TANGO [5] as well as STARS [6]. The toolkit is capable of receiving and processing JPEG-type video streams.

The Web2cToolkit is a proprietary framework that is not built on widely used JavaScript toolkits such as Dojo [7]. Due to progressing standardization efforts and the impressive development speed of modern web technologies, the Web2cToolkit is outdated and hardly maintainable in terms of performance and usability. Therefore, the development has recently been discontinued and only bug fixes are being made.

Figure 1: Hybrid app architecture [8].

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2023). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP22

Figure 2: Accelerator operations schedule.

PROGRESSIVE WEB APPS

Nowadays, users are spoiled by the powerful and rich features of graphical user applications that can be downloaded from Apple's or Google's application web stores. These applications tend to be native applications implemented for a single platform and without a common code base as they most likely originated from various platform-specific software development kits.

Hybrid applications (Fig. 1) offer a way out of this dilemma. Hybrid applications are multi-platform applications with a common code base. The code is based on platform-independent web technologies and a platform-specific component (WebView) wrapped into a native container (e.g. Cordova [9]). However, an application is still deployed and installed via a platform-specific application web store.

Progressive Web Apps (PWA) are platform-agnostic web applications developed using a set of specific technologies (e.g., HTML, CSS, JavaScript, WebAssembly, Service Worker) and standard patterns to take advantage of web and native application features. A PWA runs in the browser engine, either embedded in the browser window or as a fast-loading, network-independent standalone application added to the user's home screen. It looks and feels like a native app, has a responsive design, can be used on any device and includes offline storage and access to native features. A PWA can be discovered through a simple web search and is downloadable from a web server.

TECHNOLOGY EVALUATION

Modern web development platforms like Angular [10], React [11] or Vue.js [12] are well suited for the implementation of Progressive Web Apps. Recently, DESY has started to explore the possibilities of cross-platform graphical user apps based on React for the control of accelerators and beamlines. So far, 3 evaluation projects using the React framework have been carried out.

Web Data Display App

The Web Data Display app follows the design principles of the Web2c Synoptic Display Viewer application. It provides generic, customizable GUI widgets (e.g. Web2dPage, Web2dChart, Web2dTable, Web2dLabel-Value, …).

A specific graphical user application consists of a predefined set of graphical components whose configuration parameters are stored in a configuration file that is read and processed by the Web Data Display app at startup.

The graphical widgets are derived from the UI component set of the Ionic framework [13]. This open source framework is both platform-agnostic by using the cross-platform app runtime Capacitor [14] and SDK-agnostic by providing an interface to Angular, React and Vue.js. The Ionic framework provides ready-to-use build workflows for native, hybrid as well as Progressive Web Apps

Dashboard Apps

Two dashboard apps (accelerator operations schedule (Fig. 2), PETRAIII status display (Fig. 3)) based on React, JavaScript, HTML and CSS have been implemented to visualize the status and the performance of the operation of the DESY accelerators. These apps are created using the standard React project creation workflow. In most cases, style information is separated from the code and contained in CSS-files. The responsive behaviour is provided by the Material UI component library [15]. Routing within the apps is performed by the React Router library [16] which is not part of the React framework.

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2023). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP22

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2023). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

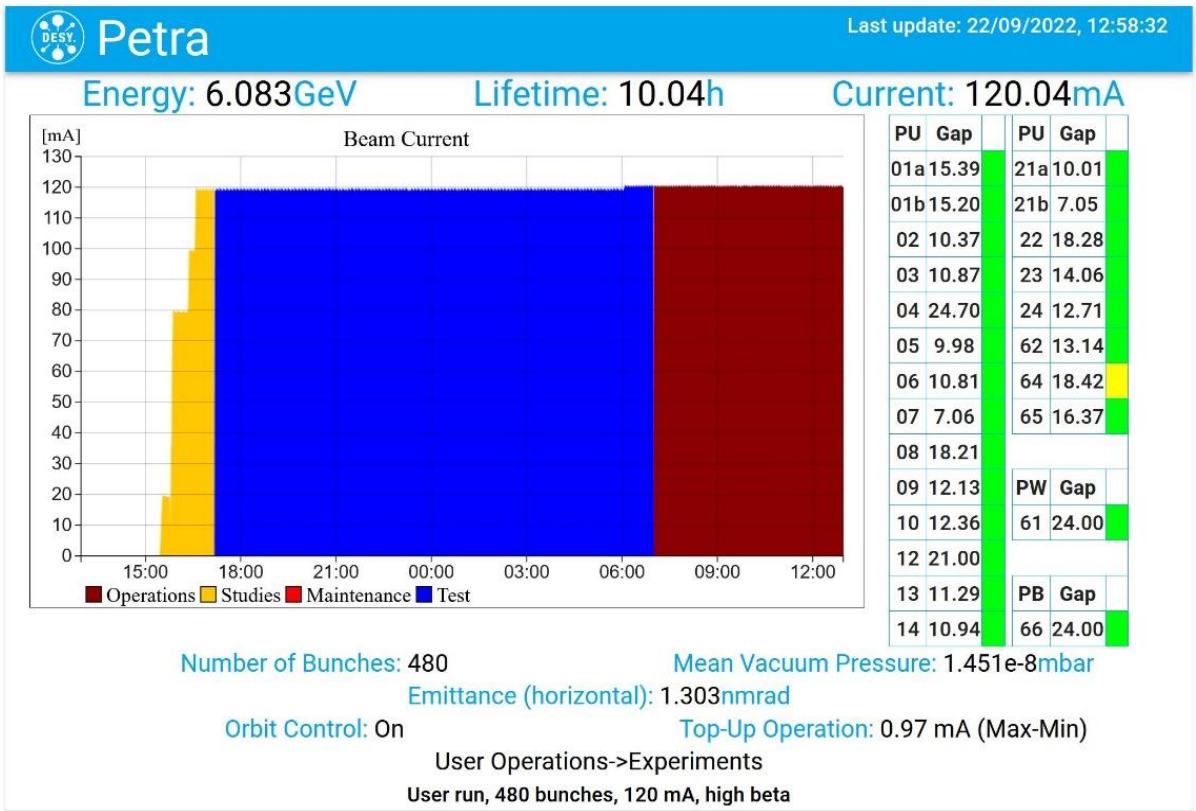

Figure 3: PETRAIII status display.

The apps communicate with the accelerator control system and receive their data asynchronously using the promise based axios [17] HTTP client from a RESTful web server which acts as a gateway to the accelerator control system. The data is encapsulated in a JSON structure that seamlessly integrates with JavaScript structures. Table 1 shows an example JSON data structure.

Table 1: Example JSON Data Structure

context	PETRA
server	VAC.ION_PUMP
device	SEK.ALL
property	P.MEAN
stsCode	0
timestamp	1663774206566
systemStamp	1873269314
fmtCode	5
numElements	1
format	FLOAT
status	success
data	1.3870215E-8

WebACOP Chart Library

The JavaScript WebACOP chart library is a novel variant of the so-called ACOP component family [18]. It provides a graphical component for displaying indexed data such as timeseries or histogram data. WebACOP wraps the D3.js [19] library in a JavaScript library. The D3.js library is suitable for creating dynamic, interactive data visualizations in web browsers and makes use of SVG, HTML5, and CSS standards. WebACOP dynamically configures the D3.js-based chart and injects data read asynchronously from the control system gateway server into the HTML/Canvas element of the drawing area. The WebACOP chart library is used to visualize live data from the operation of the PETRAIII accelerator as shown in Fig. 3. Its development is not yet completed.

FINDINGS AND CONCLUSIONS

The React framework is a powerful tool for rapid prototyping. Once implemented, components can be easily reused in other projects. While the implementation of simple components has proven to be relatively easy, the development of complex components can be quite time-consuming, cumbersome and error-prone.

The rapid release cycle of React versions often poses challenges for the developer, as backward compatibility is limited and incompatibilities with other third-part libraries can arise.

It turns out that the responsive design approach works well for dashboard applications, for example, but may not work for complex, well-designed control room applications with a variety of graphical widgets.

Our exploration of the possibilities of cross-platform graphical user apps based on React for the control of accelerators and beamlines is far from complete. For example, our experience in terms of debugging is still very limited and we also haven't explored how to migrate a React app into an Electron [20] desktop application.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP22

REFERENCES

[1] R. Bacher, "Light-Weight Web-based Control Applications with the Web2cToolkit", in *Proc. ICALEPCS'09*, Kobe, Japan, Oct. 2009, paper THP110, pp. 889-891.

[2] TINE, https://tine.desy.de

[3] DOOCS, https://doocs.desy.de

[4] EPICS, http://www.aps.anl.gov/epics

[5] TANGO, http://www.tango-controls.org

[6] STARS, http://stars.kek.jp

[7] Dojo Toolkit, https://dojotoolkit.org/

[8] Hybrid vs. Native, eBook, https://ionicframework.com/

[9] Apache Cordova, https://cordova.apache.org/

[10] Angular, https://www.angular.io/

[11] React, https://reactjs.org/

[12] Vue.js, https://vuejs.org/

[13] Ionic framework, https://ionicframework.com/

[14] Capacitor, https://capacitor.ionicframework.com/

[15] Material UI, https://mui.com/

[16] React Router, https://github.com/remix-run/react-router

[17] axios, https://github.com/axios/axios

[18] P. Duval, M. Lomperski, J. Szczesny, H. Wu, T. Kosuge, J. Bobnar, "ACOP. NET : Not Just Another GUI Builder", in *Proc. PCaPAC'18*, Hsinchu, Taiwan, Oct. 2018, pp. 139-143. doi:10.18429/JACoW-PCaPAC2018-THCB1

[19] D3.js, https://d3js.org/

[20] Electron, https://www.electronjs.org/

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2023). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI