ISBN: 978-3-95450-205-9 doi:10.18429/JACoW-Cyclotrons2019-THA03

REACT AUTOMATION STUDIO: A NEW FACE TO CONTROL LARGE SCIENTIFIC EQUIPMENT

W. D. Duckitt, J. K. Abraham, iThemba LABS, Somerset West, South Africa

Abstract

A new software platform to enable the control of large scientific equipment through EPICS has been designed. The system implements a modern tool chain with a React front-end and a PyEpics back-end as a progressive web application. This enables efficient and responsive cross platform and cross device operation. A general overview of React Automation Studio as well as the system architecture, implementation at iThemba LABS, community involvement and future plans for the system is presented.

INTRODUCTION

Mobile phone and tablet technology have driven the need for cross platform and cross device applications that deliver an instantaneous user experience.

We were eager to be a part of this technical revolution, yet the tools available to us in the EPICS [1] open source community could not enable us to develop mobile based user interfaces (UI) for EPICS.

Having upgraded many of our systems to EPICS [1–4, 6] control and having developed several state-of-the-art EPICS-EtherCAT [1–3, 6, 7] control systems with Control Systems Studio (CS-Studio) [2, 3, 5, 6] operator UIs, we were poised with the problem of converting the remaining software systems for the rest of the facility to a similar standard.

We chose however to first investigate an alternative that involved creating a progressive web application (PWA) [8] framework for EPICS. The fruits of this investigation have led to the first release of a software framework to allow real-time, cross platform and cross device responsive UI creation for EPICS. This framework has been called React Automation Studio.

A general overview of React Automation Studio as well as the system architecture, implementation at iThemba LABS, community involvement and future plans for the system is presented in the sections below.

SYSTEM REQUIREMENTS

The goal of this first release was to a develop a containerised system consisting of a back-end to serve EPICS variables to a PWA front-end that could run cross platform and cross device.

For the back-end it was critical to incorporate user authentication and authorisation, whilst ensuring that it would not be necessary to manually declare process variables that needed to be served to the client. In other words the client must request the variables needed, and the back-end server should dynamically connect and relay the process variable meta and live data to the client.

For the client, the goal was to place a data connection wrapper on freely available React components and to build in the features for macro replacement of variable and system names. This was to allow for the creation of reusable operator interfaces to implement alarm handling and to add in diagnostic ability such as a probe interface where further information about the process variable is displayed.

Finally, the system should be sufficiently documented with use cases, examples and a front-end implementation guide.

Each of the goals have been achieved and the system overview is given below.

SYSTEM OVERVIEW

React Automation Studio has been containerised with Docker [9] and version controlled as a mono-repository using Git [10].

Each of the Docker containers are deployed as micro services and environment variables can be configured to deploy the system on different ports, to enable user authentication and authorisation or to serve the application on a unique URL or on the localhost. Separate Docker commands exist to load the development and production versions. These containerised environments allow for the precise versioning of packages used and prevents deployment dependency issues.

The software stack for React Automation Studio is shown in Fig. 1 and an overview of the system components are outlined below:

pvServer

We needed to develop a back-end server to relay the process variable (PV) data to the client. We initially evaluated a JavaScript back-end using a JavaScript EPICS channel access (CA) module, but we found the support and development to be limited and it also faced stability issues.

The chosen alternative is a Python [11] back-end which uses the well supported and well maintained PyEpics [12] CA module.

The resulting micro-service is the Python process variable server (pvServer). It is layered on the Flask [13] and Flask-Socket-IO [14] web application frameworks to serve the EPICS process variables to clients.

Communication between clients and the pvServer occurs between the data connection wrapper in the client components and the pvServer as follows:

The client initially makes a Socket-IO [15] connection to the pvServer. Depending on whether or not authentication is enabled the client will first be authenticated, and then the data connection wrapper will emit Socket-IO events to the pvServer requesting access to the EPICS variable.

Depending on the clients access rights, access is either denied or the socket connection is placed in a Socket-IO room

ISBN: 978-3-95450-205-9 doi:10.18429/JACoW-Cyclotrons2019-THA03

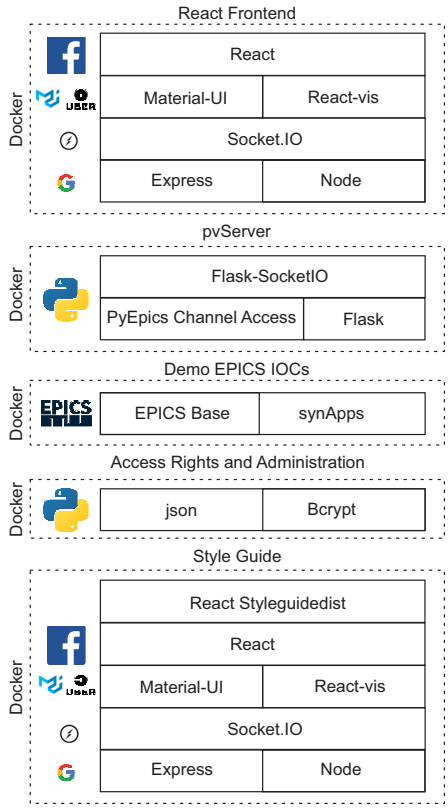

Figure 1: The React Automation Studio software stack.

with read-only or read-write privileges but with the same name as the PV. EPICS CA to the required process variables are established and the PyEpics PV is stored in a list, the connection and value change callbacks of the PyEpics CA are used to emit meta-data, connection status and value changes to the read-only and read-write rooms. The PV name is used as the event name.

In the data connection layer of the clients components, an event listener that is tied to the PV name is registered on the Socket-IO connection for each instantiation of the component. This allows efficient asynchronous updates of each listening component when the pvServer emits the PV's event update.

The only difference between the read-only and read-write rooms is that the write-access field of the meta-data has been changed to read-only based on the access rights and that for a read-write room the write access field is inherited from security rights defined by the EPICS IOC or gateway.

Similarly when writing to an EPICS variable, then depending on the access rights, the client is either granted or denied permission to write to the variable.

React Front-end

React [16] was chosen to develop the front-end for the PWA as it enabled the development of the front-end in a single language, i.e., JavaScript [17], as opposed to conventional web development in HTML, JavaScript and CSS. The UI interfaces that we have created are highly responsive and offer a real-time experience as is shown in the example of a mobile view in Fig. 2 and a desktop beam line control system in Fig. 3.

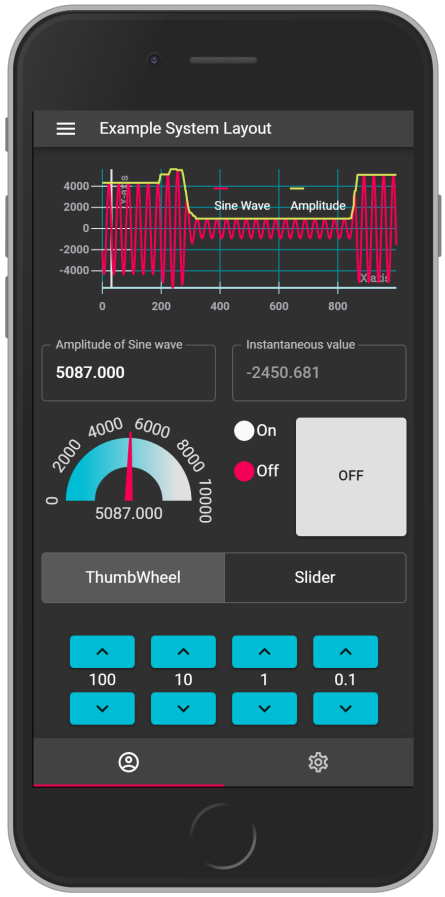

Figure 2: An example of a React Automation Studio mobile layout.

We have integrated selected components from the Material-UI [18] React component framework and the React-vis [19] graphing framework with our system to create user interfaces with the same features that we use in our current CS-Studio operator interfaces. These components have been integrated with a data connection layer which handles input and output, meta-data for labels, limits, precision, alarm sensitivity and initialisation from the pvServer.

Some components can handle multiple PVs such as graphs or single PVs i.e. text inputs. For each of the components, the PV's name can be declared using macros. The macros are replaced at component instantiation. This allows the design of complex user interfaces that can be reused by simply grouping the components and changing the global macro to point to another system.

The data connection layer can currently support EPICS process variables by adding "pva://" and local variables by adding "loc://" as prefixes to the component pv names. In the future, data connections to process variables of various protocols will be added, as well as new prefixes to indicate connection to these protocols.

ISBN: 978-3-95450-205-9 doi:10.18429/JACoW-Cyclotrons2019-THA03

Figure 3: An example of a wide screen operator display for the control of a beam line.

Demo EPICS IOC

The framework comes with a containerised demonstration IOC that enables the front-end demos to connect in real-time to a live system. The default EPICS port has been changed for the demo IOC to prevent multiple instances on different machines from influencing one another.

Access Rights and Administration

The URL, protocol selection for HTTPS or HTTP , authentication and server ports are controlled through the environment variables.

If React Automation Studio is installed on the localhost then there is no need to enable authentication as the host authentication system will protect access.

In this release, and with authentication enabled, the user name and password are managed through an administrator Docker environment through the command line. Passwords are stored on the server in encrypted format using Bcrypt [20]. In future releases this may be replaced by a web based administration page. The default authentication procedure can easily be modified to suit a different environment and point to an authentication server. The client is kept authenticated using an encrypted Jason Web Token (JWT) [21]. This JWT is used to check authorisation and access rights for every PV request and write. If the JWT is invalidated by the server then the user will be required to login.

Access rights can be controlled though a JSON file which contains user access groups and rules for defining PV access using regular expressions in the same way that the EPICS Gateway [22] access is defined. All of the components in React Automation studio currently indicate access rights to the PV.

Style Guide

A lot of effort was put into the documentation and a style guide based on React Styleguidedist [23] is used as the help function and to document the use of all the components from the source files. The current style guide is also interactive with a demo IOC. The properties of each of the components are documented and examples of their usage are shown.

IMPLEMENTATION AT iThemba LABS

React Automation Studio is currently being used as the control system front-end and overview screens for the Low Energy Radioactive Ion Beam (LERIB) [24] demonstrator which is part of the new South African Isotope Facility (SAIF) project. For LERIB it is used to control the motion control, vacuum systems, actuator systems and to display the status of the machine safety interlocking system. In total the various UIs currently connect up to 648 unique PVs from the same pvServer from multiple IOCs.

At the main separated sector cyclotron (SSC) complex, we have upgraded the harp beam diagnostics and Faraday cup front-end with a React Automation Studio front-end that controls all the harps and Faraday cups. For this system a total of 1683 unique process variables are connected to the UI from the same pvServer to multiple IOCs located on site.

We have also developed various engineering and mobile displays for other projects on site.

ISBN: 978-3-95450-205-9 doi:10.18429/JACoW-Cyclotrons2019-THA03

FUTURE PLANS AND COMMUNITY INVOLVEMENT

We are in the process of open-sourcing React Automation Studio and we encourage the community to get involved and collaborate with us on this project. The main release of React Automation Studio will contain all examples, source code and a style guide for implementation. We will also release boiler-plate example repositories that will contain simplified examples and staging areas that can be used by other laboratories to develop their own front-ends. We encourage interested persons to contact us via rasadmin@tlabs.ac.za.

In future releases we also hope to add in an interface to archived data, a dedicated alarm handler and a database abstraction layer to access saved settings.

CONCLUSION

iThemba LABS has successfully designed a new software platform to enable the control of large scientific equipment through EPICS via PWAs. The system is cross device and cross platform compatible. The operational readiness and stability of this software has been demonstrated and we encourage the EPICS community to test, evaluate and contribute to React Automation Studio.

REFERENCES

[1] EPICS, https://epics.anl.gov/

[2] W. D. Duckitt, J. K. Abraham, J. L. Conradie, M. J. Van Niekerk, and T. R. Niesler, "A new digital low-level RF control system for cyclotrons", in *Proc. 21th Int. Conf. on Cyclotrons and their Applications (Cyclotrons'16)*, Zurich, Switzerland, Sep. 2016, pp. 258–262. doi:10.18429/JACoW-Cyclotrons2016-WEB01

[3] W. D. Duckitt, J. L. Conradie, M. J. van Niekerk, J. K. Abraham, and T. R. Niesler, "The design and implementation of a broadband digital low-level RF control system for the cyclotron accelerators at iThemba LABS", *Nucl. Instrum. Methods Phys. Res., Sect. A*, vol. 895, Jul. 2018, pp. 1-9, ISSN 0168-9002. doi.org/10.1016/j.nima.2018.03.064

[4] I. H. Kohler *et al.*, "Progress in the conversion of the in-house developed control system to EPICS and related technologies at iThemba LABS", in *Proc. 13th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'11)*, Grenoble, France, Oct. 2011, paper MOPMS013, pp. 347–350.

[5] Control System Studio, http://controlsystemstudio.org/

[6] J. K. Abraham and W. D. Duckitt, "Integration of EtherCAT hardware into the EPICS based distributed control system at iThemba LABS", presented at the 22nd Int. Conf. on Cyclotrons and their Applications (Cyclotrons'19), Cape Town, South Africa, Sep. 2019, paper MOP015, this conference.

[7] R. Mercado, I. J. Gillingham, J. Rowland, and K. G. Wilkinson, "Integrating EtherCAT based IO into EPICS at Diamond", in *Proc. 13th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'11)*, Grenoble, France, Oct. 2011, paper WEMAU004, pp. 662–665.

[8] Progressive Web App https://developers.google.com/web/progressive-web-apps/

[9] Docker, https://www.docker.com/

[10] Git, https://git-scm.com/

[11] Python, https://www.python.org/

[12] PyEpics, https://cars9.uchicago.edu/software/python/pyepics3/

[13] Flask, https://palletsprojects.com/p/flask/

[14] Flask-SocketIO, https://flask-socketio.readthedocs.io/en/latest/

[15] Socket.Io, https://socket.io/

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

[17] JavaScript, https://developer.mozilla.org/en-US/docs/Web/JavaScript

[18] Material-UI, https://material-ui.com/

[19] React-Vis, https://uber.github.io/react-vis/

[20] Bcrypt, https://pypi.org/project/bcrypt/

[21] Jason Web Token, https://tools.ietf.org/html/rfc7519

[22] EPICS Gateway, https://epics.anl.gov/extensions/gateway/index.php

[23] React Styleguidedist, https://github.com/styleguidist/react-styleguidist

[24] J. L. Conradie *et al.*, "The South African Isotope Facility", in *Proc. 9th Int. Particle Accelerator Conf. (IPAC'18)*, Vancouver, Canada, Apr.-May 2018, pp. 1240–1243. doi:10.18429/JACoW-IPAC2018-TUZGBF4