ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-FR2BC001

REACT AUTOMATION STUDIO: MODERN SCIENTIFIC CONTROL WITH THE WEB

W. D. Duckitt. Stellenbosch University, Stellenbosch, South Africa
J. K. Abraham. iThemba LABS, Cape Town, South Africa
G. Savarese, D. Marcato. INFN Legnaro National Laboratories, Legnaro, Italy

Abstract

React Automation Studio is a progressive web application framework that enables the control of large scientific equipment through EPICS from any smart device connected to a network. With built-in advanced features such as reusable widgets and components, macro substitution, OAuth 2.0 authentication, access rights administration, alarm-handling with notifications, diagnostic probes and archived data viewing, it allows one to build modern, secure and fully responsive control user interfaces and overview screens for the desktop, web browser, TV, mobile and tablet devices. A general overview of React Automation Studio and its features as well as the system architecture, implementation, community involvement and future plans for the system is presented.

INTRODUCTION

React Automation Studio (RAS) is a progressive web application (PWA) framework that enables the control of large scientific equipment through the Experimental Physics and Industrial Control System (EPICS) [1].

It was born out of the need for cross-platform and cross-device applications that deliver an instantaneous user experience for the web, desktop, on a mobile phone and on tablet devices.

The initial open-source release [2] addressed many of the challenges faced in creating EPICS applications for these devices. Since then, RAS has been through another three major revisions and is now at V4.0.2.

It supports advanced features such as reusable widgets and components, macro substitution, Open Authorization (OAuth) 2.0 authentication, access rights administration, alarm-handling with notifications, diagnostic probes, loading and saving of process variable (PV) data and archived data viewing.

This enables one to build modern, secure and fully responsive control user interfaces and overview screens for the desktop, web browser, television (TV), mobile and tablet devices, such as the examples show in Figs. 1, 2, 3 and 4.

A general overview of RAS, its features as well as the system architecture, implementation, community involvement and future plans for the system is presented.

SYSTEM OVERVIEW

React Automation Studio (RAS) is a PWA framework that enables the control of large scientific equipment through EPICS. It is presented as a Git [3] mono-repository [4]with multiple microservices that are containerized with Docker [5] and orchestrated with Docker Compose [6].

Figure 1: An example of real-time mobile user interface created with RAS.

A high level block diagram of the system, showing the information flow between the primary microservices available in the mono-repository is shown in Fig. 5. Each of these microservices are discussed below:

pvServer

This is the Python [7] process variable server (pvServer). It is based on Flask-SocketIO [8] web application framework and the PyEpics [9] framework to serve the EPICS PVs to clients.

Communication between clients and the pvServer occurs between the data connection wrapper in the client components through Socket-IO [10] and REST application programming interfaces (API). The pvServer handles EPICS Channel Access (CA), authentication, authorisation and

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-FR2BC001

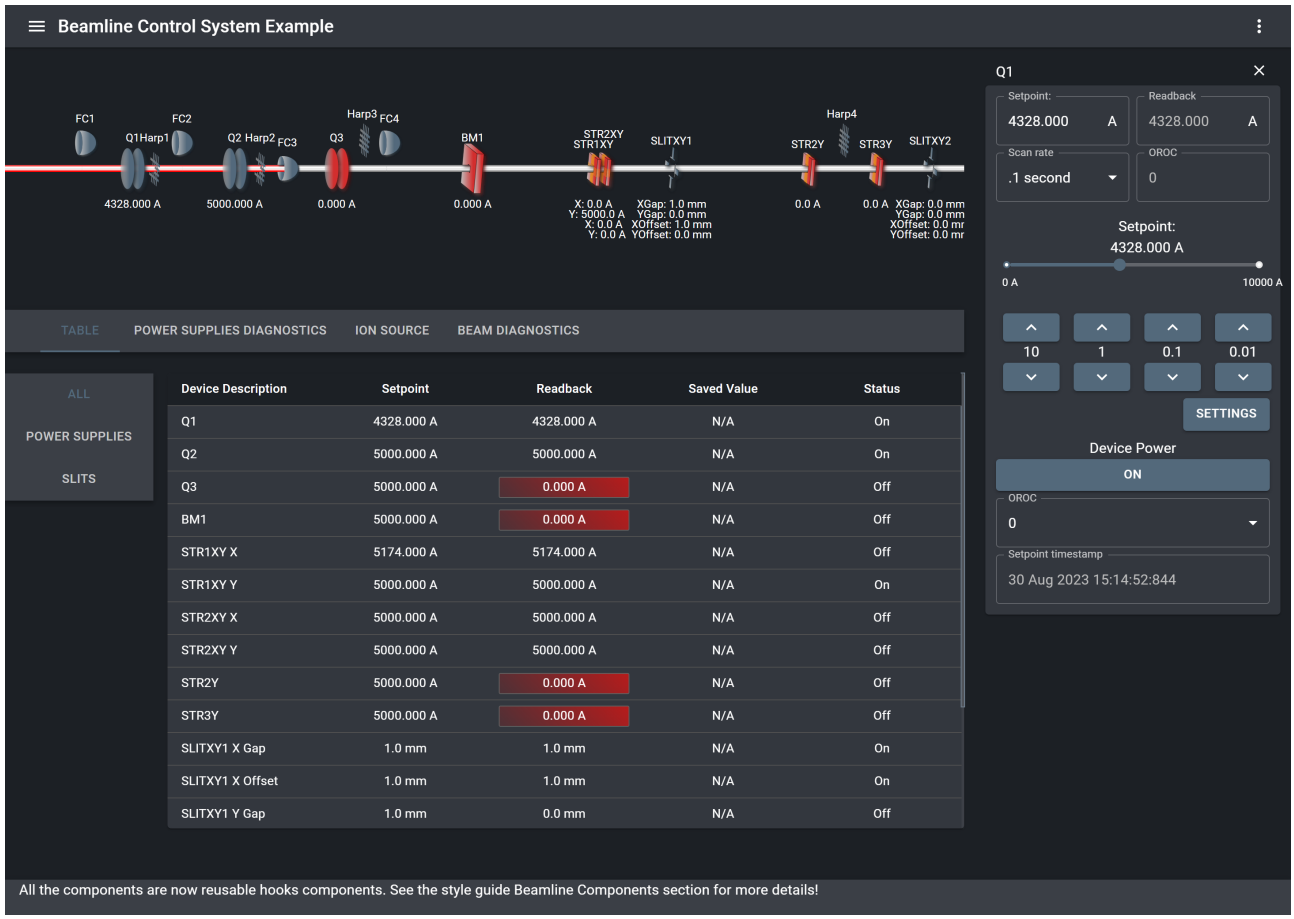

Figure 2: An example complex beam line control system user interface, that can be defined from reusable widgets and control surfaces.

database accesses from the clients.

The information flow between a client and the pvServer occurs as follows:

A client will initially make a Socket-IO connection to the pvServer. Depending on whether authentication is enabled, the client will first be authenticated, thereafter the data connection wrapper will emit Socket-IO events to the pvServer requesting access to the EPICS variable.

An access token is transmitted with each request, and depending on the client's role and access rights granted by the token, access is either denied or the socket connection is placed in a Socket-IO room with read-only or read-write privileges but with same name as PV.

Similarly, for writes to an EPICS variable, the access token is decoded and depending on the access rights, the client is either granted or denied permission to write to the variable.

If access is granted, 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 update of each listening component when the pvServer emits the PVs event update.

React Frontend

React [11] is used to develop the frontend. It allows one to develop in a single language, i.e JavaScript as opposed to conventional web development in Hypertext Markup Language (HTML), JavaScript [12] and Cascading Style Sheets (CSS).

It allows one to create highly responsive UIs with a real-time experience as is shown in the example of a mobile view in Fig. 1.

Since V3.0.0, Plotly-js [13] is used as the graphing framework and from the start, Material-UI [14] (MUI) is used as the primary user interface (UI) library.

For most of the components available in RAS, a centralized widget component and data connection layer handles, input and output, meta-data for labels, limits, precision, alarm sensitivity and initialization from the pvServer.

Some components can handle multiple PVs such as the graph or single PVs such as text inputs.

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-FR2BC001

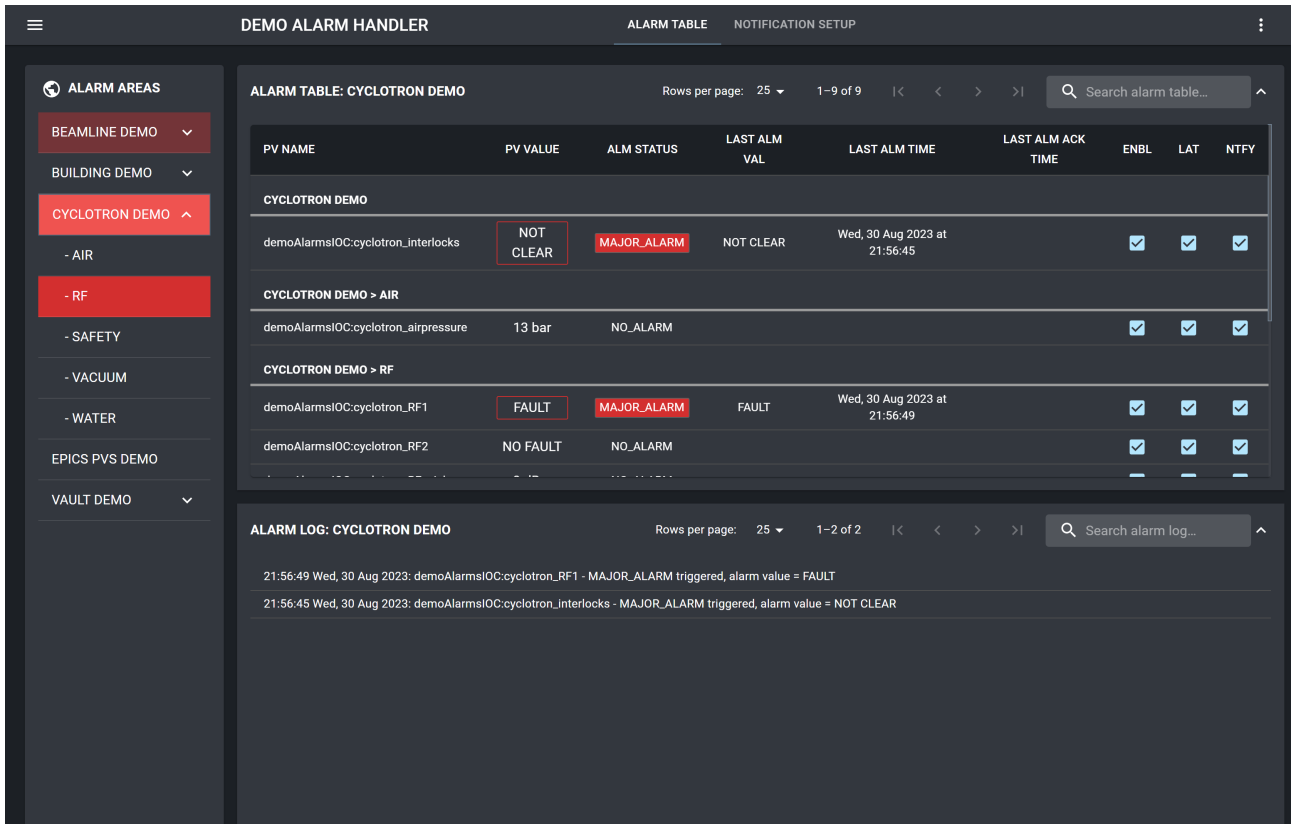

Figure 3: An example of the AlarmHandler system component showing multiple alarm areas, with active and in alarms.

For each of the components the PVs 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. An example of a complex user interface, one can create with RAS and the reusable widgets for a beam line control system, is shown in Fig. 2.

Many of the components such as TextInputs and TextOutputs have embedded diagnostic features such as a context menu and diagnostic probe as shown in Fig. 6.

Styleguide

A style guide based on React Styleguidedist [15] 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 containerized demo IOC. All the properties of each of the components are documented and examples of their usage are shown.

MongoDB

Since V2.0.0, React-Automation-Studio is integrated with MongoDB [16] to store persistent data. The PyMongo [17] driver is used within the pvServer to connect to a MongoDB replica set. React hooks are available that setup a watch, perform an update or an insert to MongoDB replica set within the pvServer.

Demo IOCs

RAS ships containerized EPICS IOCs that contain EPICS records used to test and evaluate each of the RAS components as well as the complex UIs. These simulated control system can therefore be used to evaluated RAS's full functionality before developing any code.

NGINX

Nginx [18] is a web server that can also be used as a reverse proxy, load balancer, mail proxy and HyperText Transfer Protocol (HTTP) cache.

Since Release 3.0.0, Nginx is included in RAS and serves the static files for the React frontend and the style guide, it also handles the transport layer security and performs load balancing. Scripts were created to dynamically configure Nginx based on the environment variables. For load balancing, Nginx balances between the multiple instances of the pvServer microservice.

AlarmHandler

The AlarmHandler microservice forms part of the RAS AlarmHandler component. It is discussed separately in the AlarmHandler component section.

ACCESS RIGHTS AND ADMINISTRATION

The Uniform Resource Locator (URL), protocol selection for Transport Layer Security (TLS), authentication and

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-FR2BC001

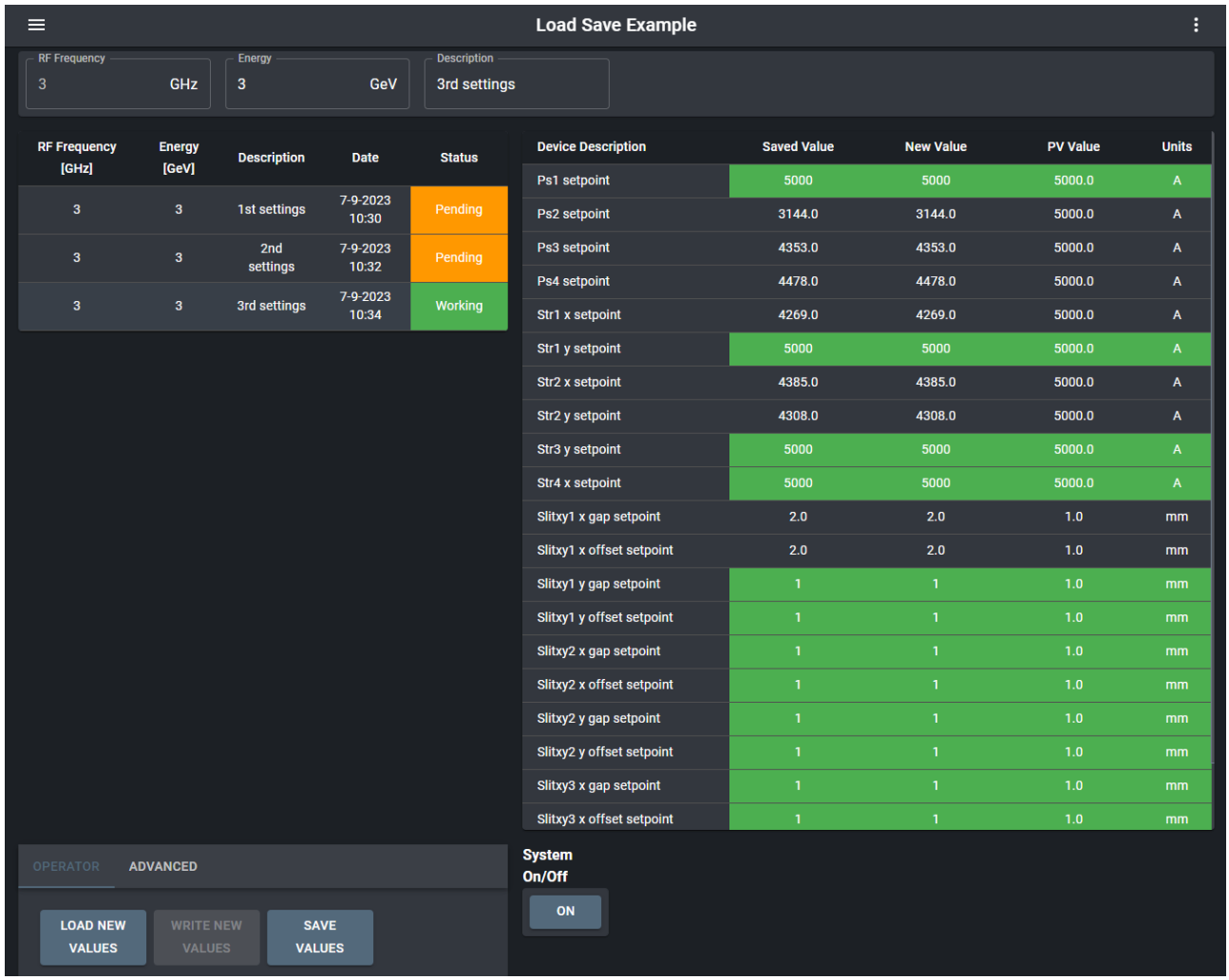

Figure 4: An example of the load save system component, showing with multiple save settings and and table showing the live, new and saved values.

server ports are controlled through the environment variables.

The configuration can be flexible, for example, if RAS is installed on a host where there is no need to enable authentication and it is deemed the host authentication system will protect access, then authentication need not be turned on.

However, if RAS is exposed on a network then at least one the modes of authentication should be enabled.

Since Release V3.0.0, RAS supports web based administration of user access rights.

It also supports external authentication through Active Directory and an OAuth2.0 authentication for Google and local authentication.

For the local authentication passwords are stored in the database using encrypted format using Bcrypt [19].

For all of the authentication modes, the client is kept authenticated using encrypted Jason Web Token (JWT) based, refresh and access tokens. When served over (Hypertext Transfer Protocol Secure) HTTPS, the refresh tokens are store in an encrypted cookie with HTTP only mode on the client and the shorter lived access tokens are kept in memory.

The memory based access token is used to check authorization and access rights for every PV request and write. The access token is short lived, whilst the refresh token typically has a longer lifetime, a week for example, to allow the user to remain logged in to the system whilst offline. The access and refresh tokens are periodically updated and if either access token is invalidated by the pvServer then user will be required to log in.

Access rights can be controlled though web based administrator which contains user access groups, roles and rules for defining PV access using regular expressions in the similar way that the EPICS Gateway [20] access is defined. All the components in RAS currently indicate access rights to the PV.

The access rights for each user are managed in the web administrator. If logged in as an admin role, the web administrator is available to the user.

The system is initially seeded with default access rights. Thereafter, further regular expression rules for user access

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-FR2BC001

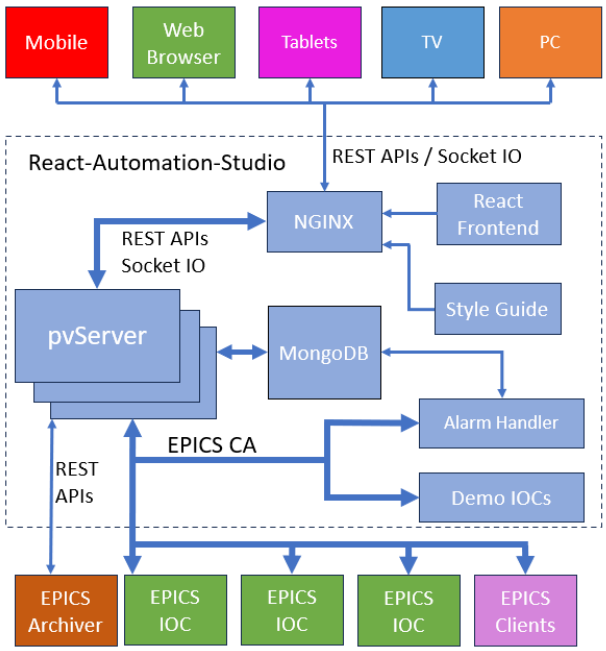

Figure 5: A high-level system diagram of RAS.

groups (UAG) can be created and used to evaluate the read and write access rights.

The order in which the UAGs and rules are defined are important. The first rule applied is the DEFAULT, all users will get this. The final access group rules to be applied are the ADMIN rules to the applicable user groups.

For example, in the DEFAULT UAG, the rules disables write access and enable read access for all usernames and process variables. This is useful for creating overview screens or sharing operator interfaces where write access is not allowed.

To enable write access for everyone one could check the write access check boxes of the DEFAULT UAG. To disable read access and therefore prevent access by anyone by default one could deselect the read checkboxes.

In the pvServer, for each read and write access request of a PV, the rules in the UAG are applied if the username is defined in the UAG and the regular expression match function is satisfied. If the match is true, then the rule is applied.

In theory, all regular expression searches allowed by Python regex can be used, allowing the creation of UAGs suited for the roles of engineers, scientists and operators.

SYSTEM COMPONENTS

Since Release 3.0.0 of RAS, system components for alarm handling, archived data viewing and loading and saving of data have been included in RAS. Each of these components are discussed below:

Alarm Handler

The Alarm Handler, shown in Fig. 3, allows users to configure all aspects of the alarms and search through the entire alarm log. Alarm areas, sub-areas and PVs can also be added/removed from the front end by alarm Admin role users.

The Alarm Handler can be initialized through JSON files that populate the MongoDB alarm handler database. This database is also used to persist all alarm events and activity logs.

A user notification platform has also been created for the alarm handler. This platform allows a user to target specific PVs to be notified about using JavaScript regular expressions. Users can be notified via email and Signal messenger.

The Alarm Handler can be built into RAS system, for convenience a standalone Alarm Handler project [21] also exists to deploy the Alarm Handler site wise at facilities.

Archiver Data Viewer

The Archiver Data Viewer is an interface to display EPICS archived data. It's built with backend APIs that allow connection up an EPICS archiver. The component allows one to view the live values and the historical data of PV. A valid archiver URL needs to be provided, and the viewer will indicate if a connection can't be established to the PV's archived data. For demo purposes, the RAS project provides a containerized demo EPICS archiver [22].

Load Save Component

The Load Save Component is built on a RAS based frontend that connects to the MongoDB database that contains the saved settings data. An example Load Save user interface is show in Fig. 4. The live values can be saved with along with metadata and the saved settings data can then be written to live processes variables in a controlled way by first loading the saved values to new values and then finally writing the new values to PV values. The life-cycle of the saved settings can also be managed on the advanced tab.

DEPLOYMENT AND ORCHESTRATION

RAS have been containerized with Docker, and a sole prerequisite to install the system is Docker. Users are encouraged to read the full documentation [4] as multiple Docker Compose configurations exist in the mono-repository, to bring up the microservices in a development or production mode.

COMMUNITY INVOLVEMENT AND FUTURE PLANS

Since the open-sourcing of the initial release of RAS [2], RAS has been through another 3 major versions and is now at V4.0.2. Only through community involvement, has it been possible to keep track and stay up to date with the JavaScript and Python frameworks behind the scenes. RAS has been deployed in various forms at several laboratories [2, 23–25].

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-FR2BC001

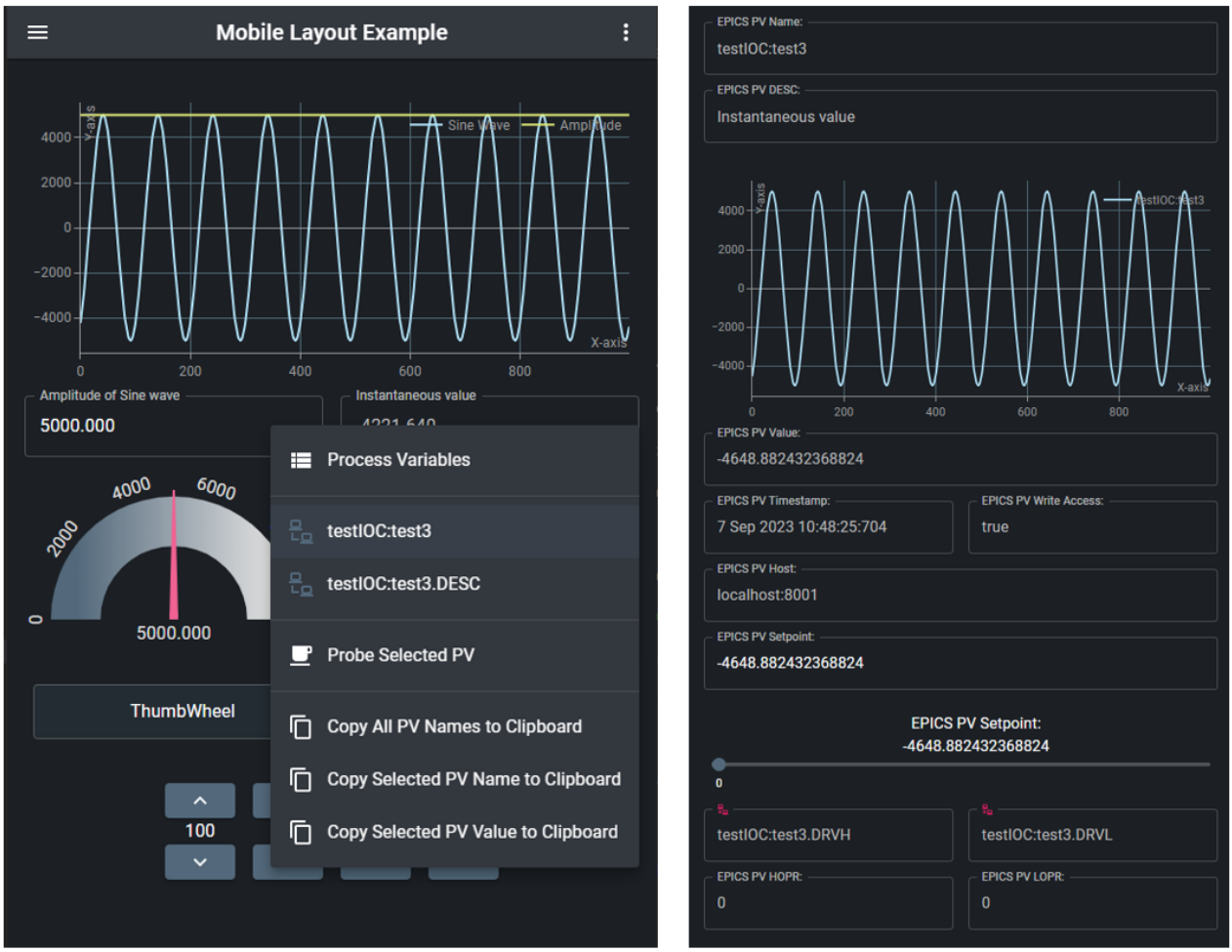

Figure 6: An example user interface showing the built-in diagnostic probes context-menu on the left and the pop-up diagnostics probe window on right.

We encourage other EPICS community users to test and evaluate RAS, contribute to features and to get involved on the Git repository [4].

CONCLUSION

The current React-Automation-Studio (RAS) architecture and features have been presented. Additional features, such as, Open Authorization (OAuth) 2.0 authentication, access rights administration, alarm-handling with notifications loading and saving of process variable (PV) data and archived data viewing have been added to the system since the initial release, have been discussed. We encourage other EPICS community users to test and evaluate RAS, contribute to features and to get involved on the Git repository [4].

REFERENCES

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

[2] W. Duckitt and J. Abraham, "React Automation Studio: A New Face to Control Large Scientific Equipment", in *Proc. Cyclotrons'19*, Cape Town, South Africa, 2020, pp. 285–288. doi:10.18429/JACoW-Cyclotrons2019-THA03

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

[4] React-Automation-Studio V4.0.2, https://github.com/React-Automation-Studio/React-Automation-Studio/tree/V4.0.2

[5] Docker: Accelerated Container Application Development, https://www.docker.com/

[6] Docker Compose overview | Docker Docs, https://docs.docker.com/compose/

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

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

[9] PyEpics-Epics Channel Access for Python, https://pyepics.github.io/pyepics/

[10] Introduction | socket.io, https://socket.io/docs/v4/

[11] React, https://react.dev/

[12] Javascript | mdn, https://developer.mozilla.org/en-US/docs/Web/JavaScript

[13] Plotly JavaScript graphing library in JavaScript, https://plotly.com/javascript/

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-FR2BC001

[14] MUI: The React component library you always wanted, https://mui.com/

[15] React Styleguidist: Isolated React component development environment with a living style guide | React Styleguidist, https://react-styleguidist.js.org/

[16] MongoDB, https://www.mongodb.com/

[17] PyMongo, https://pymongo.readthedocs.io/en/stable/

[18] Advanced Load Balancer, Web Server, & Reverse Proxy - NGINX, https://www.nginx.com/

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

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

[21] React-Automation-Studio-Alarm-Handler-Standalone, https://github.com/React-Automation-Studio/React-Automation-Studio-Alarm-Handler-Standalone

[22] React-Automation-Studio-Demo-Archiver, https://github.com/React-Automation-Studio/React-Automation-Studio-Demo-Archiver

[23] A. Havranek *et al.*, "Conceptual design of the COMPASS-U control systems", *Fusion Eng. Des.*, vol. 170, p. 112 550, 2021. doi:10.1016/j.fusengdes.2021.112550

[24] K. Yan *et al.*, "High-voltage detuning power system of HIRFL-CSRm electron cooler for Dielectronic-Recombination experiments", *Nucl. Instrum. Methods Phys. Res. A*, vol. 1046, p. 167 699, 2023. doi:10.1016/j.nima.2022.167699

[25] P. Weigel *et al.*, "The EPICS control system for IsoDAR", *Nucl. Instrum. Methods Phys. Res. A*, vol. 1056, p. 168 590, 2023. doi:10.1016/j.nima.2023.168590