React Automation Studio: A New Web Application to Generate Graphical User Interface for EPICS Based Control Systems

G. Savarese[1], D. Marcato[1], D. Bortolato[1], F. Gelain[1], G. Arena[1], M. Roetta[1], V. Martinelli[1], E. Munaron [1]

[1] *INFN, Laboratori Nazionali di Legnaro, Legnaro (Padova), Italy.*

INTRODUCTION

Control System Studio (CS-Studio) is the legacy application being used at Laboratori Nazionali Legnaro (LNL) to generate graphical user interfaces to control EPICS based control systems. This application is widely used in the laboratories and its strength resides in the ease of use. Unfortunately, this application has performance and feature drawbacks. The absence of a backend, executing in background computationally expensive tasks, is a great limit, in fact the application heavily slows down when there are multiple instances with lot of connections to Process Variables (PVs). But, one of the most relevant problem is that, since 2015, we are blocked to an old version of CS-Studio, no more supported by the community and using an old version of JAVA. We are looking for a solution since then.

AN ALTERNATIVE SOLUTION

At first we looked at CSS-Phoebus, a renewed version of CS-Studio, that inherits all its simplicity. On the other hand it inherits a portion of the problems already present in CS-Studio such as the absence of a backend and the absence of a subset of functionalities required by users (possibility to use keyboard inputs on sliders). For these reasons we decided to look for other solutions.

In late 2019, the control system group, found the existence of a new project, developed and maintained by the iThemba Labs: React Automation Studio (RAS). RAS is a web application to generate graphical user interfaces to control EPICS based control systems. RAS is meant to be used from former CS-Studio users, so a large section of its features emulates and improves CS-Studio ones. In 2020, during the lock-down period, our group had the possibility to dive into this project and start an informal collaboration with the authors carrying significant improvements. Consequently we made and participated to seminars to better learn the technologies this project is based on.

RAS APPLICATION ARCHITECTURE

This application uses the docker technology. Docker foresees the usage of *containers*, virtual machines seen as processes from the hosting operating system, to create the correct environment for each application main block.

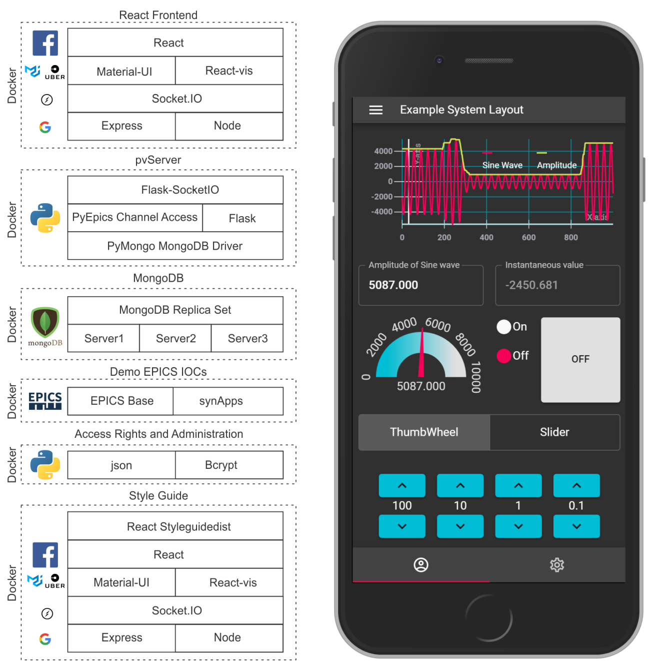


Fig. 1. React-Automation-Studio software stack [1]

The main blocks are the *REACT frontend* and the *python backend*.

The frontend server is written in REACT, the leading javascript framework to write web interfaces. This framework is widely supported and has a large set of libraries to execute different tasks. Components are the fundamental blocks when writing REACT pages. They can be largely reused and can receive parameters called *props* to customize their behavior.

RAS provides a set of standard components, called **widgets**, managing PVs connection. They are based on *Material-UI* components and make use of *socket-io* connections to communicate with the backend.

To create user interfaces, developers must implement their own components which may include widgets, custom components or components belonging to other REACT libraries. Developers can use them as black boxes, they can implement their custom widgets or more complex pages.

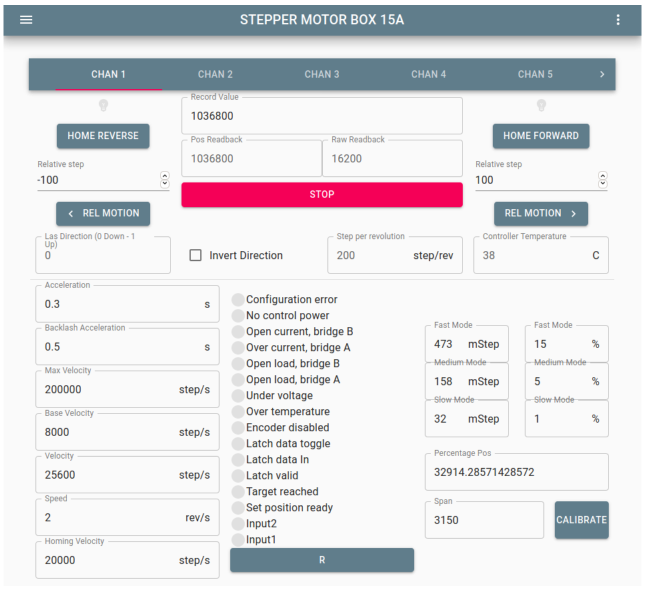
The backend server main duty is to establish the connection with the PVs. The backend is a python flask application and, to manage asynchronous duplex communications, it uses sockets. For each PV it creates a *room* where clients can register and receives updates through socket events. This way, different REACT components, connected to the same PVs, register to the same room. This approach reduces the number of channel access communications and receives updates at the same time.

The frontend and the backend both live in their own container. Other relevant blocks are: the container with the style-guide, largely used when developing new pages to inspect widgets props and behavior; the container with a demo IOC, RAS demos use those PVs to show example pages (those PVs are also accessible to developers but they are not accessible outside that container); the MongoDB Database to store persistent data (for example alarm history) and more.

APPEARANCE AND FEATURES

RAS clients only need a browser and the correct *url* to access the control pages.

During this year the control system group developed some example pages with good results. The application provides a large set of widgets similar to the ones provided by CS-Studio (*TextInputs, TextUpdates, Sliders, Gauges* and more...) but with new functionalities and a new graphic. Moreover widgets appearance and features can be customized by users.



STEPPER MOTOR BOX 15A

CHAN 1 | CHAN 2 | CHAN 3 | CHAN 4 | CHAN 5

Record Value: 1036800
Pos Readback: 1036800
Raw Readback: 16200
Relative step: -100
Relative step: 100
Las Direction (0 Down - 1 Up): 0
Invert Direction
Step per revolution: 200 step/rev
Controller Temperature: 38 C
Acceleration: 0.3 s
Backlash Acceleration: 0.5 s
Max Velocity: 200000 step/s
Base Velocity: 8000 step/s
Velocity: 25600 step/s
Speed: 2 rev/s
Homing Velocity: 20000 step/s

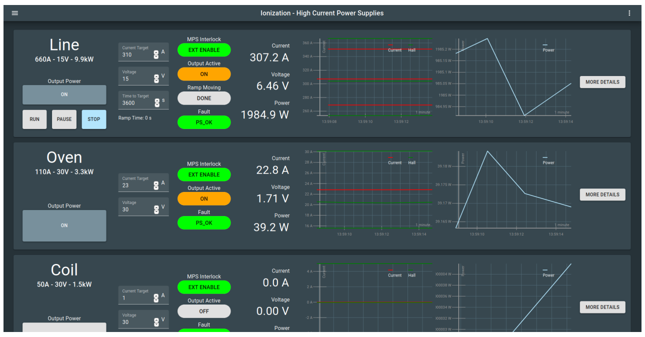
Configuration error
No control power
Open current, bridge B
Over current, bridge A
Open load, bridge B
Open load, bridge A
Under voltage
Over temperature
Encoder disabled
Latch data toggle
Latch data In
Latch valid
Target reached
Set position ready
Input2
Input1

Fast Mode: 473 mStep
Medium Mode: 158 mStep
Slow Mode: 32 mStep
Fast Mode: 15 %
Medium Mode: 5 %
Slow Mode: 1 %
Percentage Pos: 32914.28571428572
Span: 3150

Fig. 2. Stepper motor box control page developed by the control system group

Good news come also for graphs based components. These components extend the ones defined in the react-vis library: lines, lines with error marks, scatter plots, horizontal and vertical bars, area graphs and more.

In figure 2 and figure 3 we provide some example pages developed by our group in this period to show the look and feel of the new graphics. Finally the application already provide different themes which can be automatically applied without writing new code.



Ionization - High Current Power Supplies

Line — 660A - 15V - 9.9kW
Output Power: ON
Current Target: 310 A
Voltage: 15 V
Time to Target: 3600 s
Ramp Time: 0 s
MPS Interlock: EXT ENABLE
Output Active: ON
Ramp Moving: DONE
Fault: PS_OK
Current: 307.2 A
Voltage: 6.46 V
Power: 1984.9 W

Oven — 110A - 30V - 3.3kW
Output Power: ON
Current Target: 23 A
Voltage: 30 V
MPS Interlock: EXT ENABLE
Output Active: ON
Fault: PS_OK
Current: 22.8 A
Voltage: 1.71 V
Power: 39.2 W

Coil — 50A - 30V - 1.5kW
Output Power
Current Target: 1 A
Voltage: 30 V
MPS Interlock: EXT ENABLE
Output Active: OFF
Fault
Current: 0.0 A
Voltage: 0.00 V
Power

Fig. 3. Ionization control page developed by the control system group

PROS AND CONS

As already anticipated this application has a lot of advantages: it is written in REACT which is the leading framework for web applications and it is widely supported; clients require only a browser to access to the application; it gives the possibility to perform auditing; developers can create their custom widgets and share them with the community; code re-usability thanks to components; multiple themes and responsiveness to all resolutions and more.

We can't hide that the application currently has some cons: there is not the possibility to drag and drop components as for CS-Studio; it has some performances issues; it requires knowledge about REACT and Material-UI; still bugs to discover due to its youth.

FUTURE IMPROVEMENTS

The decision to emulate CS-Studio features has recently lead the authors to develop an Alarm Handler (AH) system and an Archiver system. They both uses a MongoDb Database, a document oriented DB, to store data. MongoDb privileges the data readability and simplicity. A new feature of the AH will be the possibility to send emails or telegram notifications.

In late 2020, our group suggested the extension of the authentication system in order to integrate custom external authentication and authorization systems. After different meetings, discussions and a proof of concept, provided by us, on how to integrate a custom OpenIDConnect authentication system, the main team is currently integrating our suggestions in the main project.

This project is only at the beginning of its life but we have high expectations about its future; for this reason we are planning to substitute the legacy CS-Studio with RAS.

[1] https://github.com/wduckitt/React-Automation-Studio.