ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-FR2BC003

TARANTA PROJECT - UPDATE AND CURRENT STATUS

Y. Li*, V. Hardion, M. Eguiraun, J. Forsberg, M. Leorato
MAX IV Laboratory, Lund, Sweden
D. Trojanowska, M. Gandor, S2Innovation, Kraków, Poland
M. Canzari, INAF-OAAB, Teramo, Italy
V. Alberti, INAF-OATs, Trieste, Italy
H. Ribeiro, Atlar Innovation, Portugal
A. Dubey, Persistent Systems, Pune, India

Abstract

Taranta, developed jointly by MAX IV Laboratory and SKA Observatory, is a web based no-code interface for remote control of instruments at accelerators and other scientific facilities. It has seen a great success in system development and scientific experiment usage. In the past two years, the panel of users has greatly expanded. The first generation of Taranta was not able to handle the challenges introduced by the user cases, notably the decreased performance when a high number of data points are requested, as well as new functionality requests. Therefore, a series of refactoring and performance improvements of Taranta are ongoing, to prepare it for handling large data transmission between Taranta and multiple sources of information, and to provide more possibilities for users to develop their own dashboards. This article presents the status of the Taranta project from the aspects of widgets updates, packages management, optimization of the communication with the backend TangoGQL, as well as the investigation on a new python library compatible with the newest python version for TangoGQL.

In addition to the technical improvements, more facilities other than MAX IV and SKAO are considering to join Taranta project. One workshop has been successfully held and there will be more in the future. This article also presents the lesson learned from this project, the road map, and the GUI strategy for the near future.

INTRODUCTION

A web-based Graphic User Interface (GUI) application tailored for big research facilities represents a pivotal tool in advancing scientific research and experimentation within this specialized field. In this academic context, we will introduce a web-based GUI application that is involved in big research facilities and plays a critical role in facilitating and enhancing various aspects of scientific experiments and data visualization.

Taranta serves as a bridge between a Tango ecosystem and the researchers who utilize its resources. It provides an intuitive and accessible platform through web browsers, simplifying experiment setup, data acquisition, and post-processing tasks. It helps the end users to build a user oriented software. This article will explore the improvement on the fundamental components of Taranta in the synchrotron and radio telescope context, emphasizing their roles in device debugging, beamline control, and experiment management.

Taranta, initially named Webjive, was started in 2018 [1] at MAX IV shortly thereafter joined by SKAO. This web application aims at providing a no-code interface for end-users to easily build up their own GUI with minimal coding skills [2]. In 2019, this project was renamed to Taranta and in 2021, officially entered in the Tango Controls Collaboration [3] portfolio. Since then, Taranta has undergone a continuous process of development and exploration, with the aim of incorporating new features and optimizing the user experience. As the spectrum of user requirements continues to broaden, the expansion of current context of Taranta becomes foreseeable.

After these years of development and learning from users' experiences, Taranta version 2.x was released. This article delves into the limitations have been seen on previous release of Taranta (version 1.x) and the improvements achieved in Taranta version 2.x, specially the enhanced frontend performance in handling large number of data points. The improvements on data transmission and widgets are also highlighted, affording researchers the ability to seamlessly monitor equipment, tune parameters, and greater control over experimental processes.

FRONTEND IMPROVEMENT

Refactoring on Data Management

The refactoring on the data transmission and management is one of the biggest improvement of this second version. Taranta currently consists of two views: Devices and Dashboards. In Devices, a generic view of all devices available on the specified Tango database is provided, where users can navigate through the device tree and interact with a selected device. An example of the device viewer is shown in Fig. 1. One can read the device's current state and interact with its attributes and commands, and access the user actions on this tango device through Taranta.

In the Dashboards view, a variety of widgets can be utilized to create a graphical user interface for different purposes on an empty canvas. It allows users to easily develop and customize a dashboard and can immediately run the dashboard to interact with Tango devices. Once created, the dashboards can be saved, shared within the same user group and exported to a json file.

* yimeng.li@maxiv.lu.se

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: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-FR2BC003

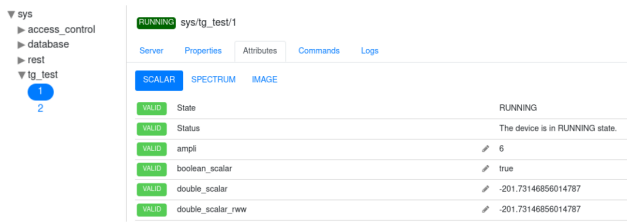


Figure 1: Taranta Devices panel.

In Taranta, a single thread controls the communication between the front end and the data source, which means it opens a socket when a dashboard switches from edit to run mode and closes when the dashboard stops running. This architecture had a good-enough performance for small databases, the response was fast and immediate. However, when the number of attributes increased, a performance issue was noticed. Over time, a running dashboard may experience gradual performance degradation, occasionally leading to a complete freeze in functionality.

Complexity of a dashboard also decreases the front end performance. It took more time to re-render a complicated running dashboard as any attribute change event caused the whole dashboard to be re-rendered. In order to optimize communication, reuse the logic for existing views, and more importantly for any future expansion, a new architecture for data management within the Redux store [4] was introduced. The diagram of the new structure is shown in Fig. 2. In-

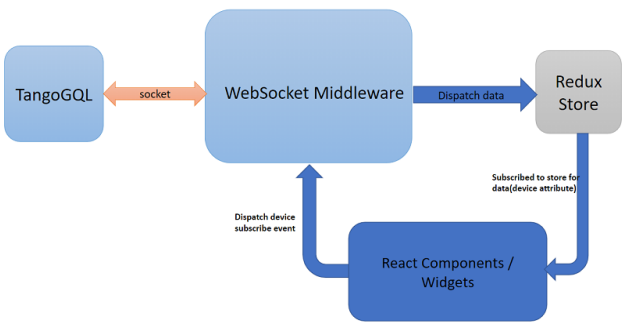


Figure 2: Websocket block diagram.

stead of directly fetching data from the backend server TangoGQL, a websocket middleware was established between the Redux store and TangoGQL, facilitating data retrieval from TangoGQL to be stored in the Redux store. Additional to the socket listener, the middleware also continuously listens to redux events, such as subscribe event and unsubscribe event [5]. It also opens the possibility to create customized listeners in the middleware to meet potential future requirements. The structure of the store is shown in Fig. 3. One significant improvement on this structure is that any change in the store will only cause the respective subscribed components to re-render, avoiding unnecessary rendering of static components. Another feature that is worth to mention is the data storage after a dashboard stops running. In the previous versions (version 1.x), there is no data stored in the front end.

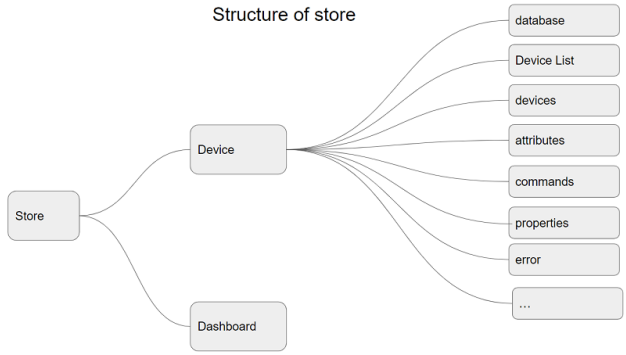


Figure 3: Structure of the Redux store.

Once the user stops running any dashboard, the subscription stops immediately. For some use cases, it is not sufficient for a continuous monitoring. Therefore, from Taranta 2.0, in the event of a dashboard's interruption, the most recent attribute value will be retained in the Redux store and presented to users upon the dashboard's resumption.

Runtime Performance Analysis A runtime performance analysis was undertaken to compare the two architectures of communication on the MAX IV largest database by running a dashboard with all existing widgets and 20 different attributes, where data points are injected every three seconds. The performance was measured by recording a running dashboard for around 1 min on Chrome DevTool. The versions in the comparison are Taranta 1.3.12 and Taranta 2.4.0. The test results for the first running minute are shown in Fig. 4, where the rendering and scripting in version 2.4.0 take only half the time compared to version 1.3.12. After running 30 minutes, the performance was measured again and the comparison are shown in Figs. 5 and 6. We can see that in version 1.3.12, scripting takes 397 ms (48%) of the CPU time in one data injection period, while in version 2.4.0, it only takes 144 ms (27%). Also, the rendering time is reduced by 62% compared to version 1.3.12.

Table 1 shows the runtime heap size usage where the value range represents the min and max usage in the measurement. The heap size in version 1.3.12 is notably increasing with time while in version 2.4.0 it shows a comparatively stable value. Due to the all rendering issue in Taranta 1, data messages cannot be processed fast enough, which results in a message queue and could possibly lead to a memory leak.

The analysis also includes the peak response time of live data plotting, which was identified as the most resource-intensive in terms of RAM consuming. Figures 7 and 8 show a single animation frame where all widgets are rendering. In version 2.4.0, only the plot widgets are rendering since the other attributes are not updating at that moment. However, before refactoring, Taranta was not able to only render the new coming values. We can see that it caused a peak CPU time in version 1.3.12 while in version 2.4.0 the time is short even if it still causes a frame rate drop, as the top red bar implies.

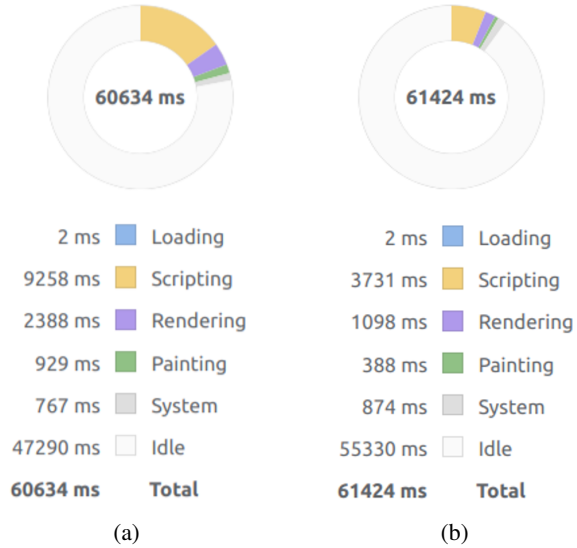
In this test, on average, the runtime performance for ver-

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: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-FR2BC003

Table 1: Heap Size Usage

Taranta Version	short run	long run
1.3.12	25.8 MB-41.5 MB	172 MB-246 MB
2.4.0	49.1 MB-64.7 MB	52.2 MB-81.7 MB



(a) (b)

Figure 4: Performance analysis for the first running minute in version 1.3.12 (a) and version 2.4.0 (b).

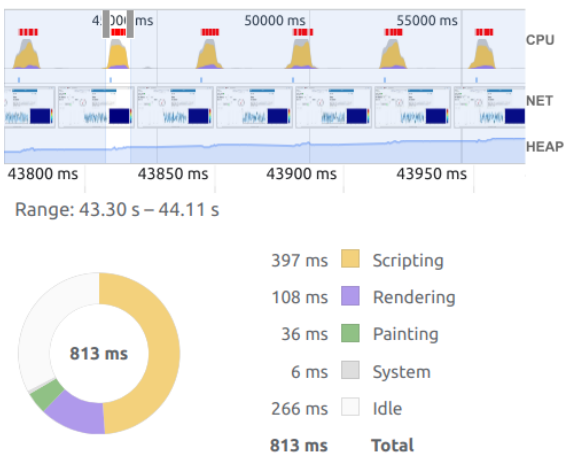


Figure 5: Taranta 1.3.12 one cycle update after running 30 minutes.

sion 2.4.0 is improved 54% compared to the version before refactoring.

Widgets Update

Due to the changes introduced with the new data communication architecture, existing widgets must be refactored. In addition, a few more widgets have been developed to meet the increasing needs from facility users. The widgets can be categorized by dealing with different use cases, as shown in Table 2. There are nine groups of widgets so far [6]. The label widget can be used for displaying thematic blocks and

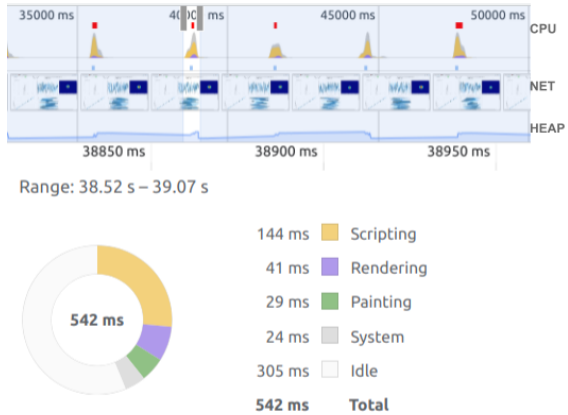


Figure 6: Taranta 2.4.0 one cycle update after running 30 minutes.

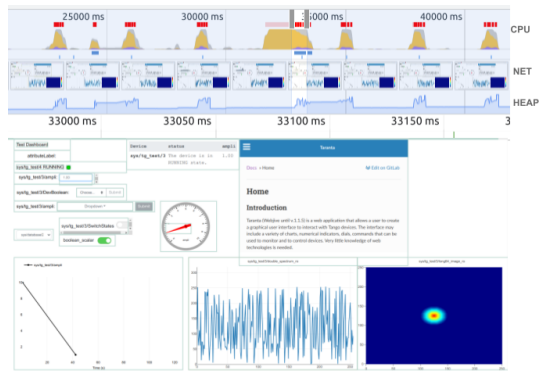


Figure 7: All widgets rendering on Taranta 1.3.12.

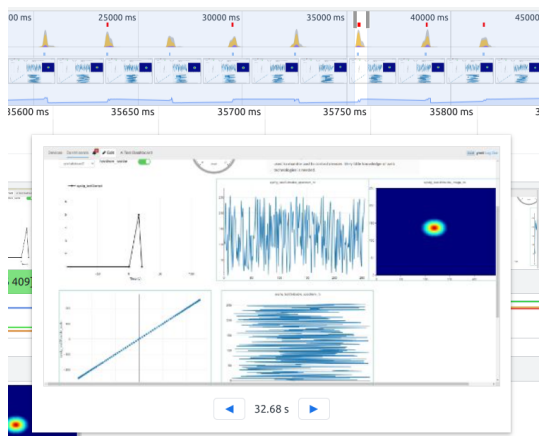


Figure 8: All plot widgets rendering on Taranta 2.4.0.

proves valuable for organizing distinct areas within a single dashboard. Attribute related widgets are most commonly used. There are different attribute widgets capable of reading and writing attributes of various data types. Monitoring data can be realized by running plot widgets, in which the X and Y axis can be configured dynamically by users.

Sending commands to control devices can be done through a group of command widgets. From Taranta version 2.x, a few simple command widgets such as command executor and writer are deprecated. Instead, a new combined

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: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-FR2BC003

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

Table 2: Widgets Categories

Categories	Widget Name
Attributes	Attribute Display
	Attribute Dial
	Attribute Writer
	Attribute Writer Dropdown
	Boolean Display
	LED Display
	Spectrum Table
	Tabular View
Commands	Command File Widget
	Command Switch
	Command Widget
Dashboard	Embed Page
	Variable Selector
Data Plot	Attribute Heat Map
	Attribute Plot
	Attribute Scatter
	Spectrum 2D widget
	Timeline Widget
Grouping Widgets	Box Widget
Images	Image Display
	Image Table
Labels	Label Widget
Logs	Attribute Logger
	Elasticsearch Log Viewer
Sardana	Macro Button
	Sardana Motor

command widget is created to handle variant command input argument and execution. "Command Switch" is a special widget in which users can listen to a device attribute and specify an ON and OFF status separately. When a dashboard starts running, if the attribute's value equals either of the specified statuses, the command switch will stay as the corresponding execution state. If the attribute's value is neither ON or OFF, a red exclamation mark will show up with a warning message.

Box widget was developed to handle multiple widgets together. It allows users to group multiple widgets inside a box, enabling users to define complex dashboards. One layer of box nesting is also supported. Box widget makes it easier to batch process widgets and managing more complicated dashboards.

Monitoring device running logs and camera streaming are also available in Taranta version 2.x. Image display widget is able to stream tango image attribute. This feature facilitates remote data acquisition monitoring for users. For handling large image transmission, data compression is applied by using aiohttp web response compression method [7] in TangoGQL. Elasticsearch log viewer widget was introduced to show device running logs from Elasticsearch server. This functionality proves highly beneficial for analyzing device performance. Users can also customize the log level and refresh time, as well as the displaying field context.

As part of the Tango control community, Taranta also supports interaction with Sardana [8], a program for experiment control. With this feature, data acquisition can be steered from Taranta. To control a motor, one can use Sardana motor widget. It provides an interface to move a motor to a specified position while monitoring its state transition. In addition, executing Sardana macros is also available from Taranta. Specifying corresponding Pool, Macroserver and Door devices, one can select a macro and specify arguments, to run experiments in run mode.

With the increase of demands on dashboard complexity, the "Dashboard Variable" was introduced to handle device change for multiple widgets. Each dashboard can define a list of dashboard variables, each one is assigned with a device. The variable can be selected the same way as other tango devices. The connected device can be changed via the Device Selector widget. Once the device is changed, all widgets with this variable selected will be synchronized to the new device.

Packages Management

In order to enhance the security of Taranta and ensure the continuity of development, a great effort was made to resolve the critical vulnerabilities of this project. In this migration, 446 vulnerabilities were reduced to 24 warnings, in which all critical vulnerabilities were resolved. Critical vulnerabilities in web applications encompass a spectrum of weaknesses that, if left unaddressed, can lead to severe security breaches. These vulnerabilities may arise from coding, configuration, or design issues, and they open doors for various types of attacks, including but not limited to Cross-Site Scripting (XSS) [9], Cross-Site Request Forgery (CSRF) [10], and remote code execution. Many of these vulnerabilities in Taranta project are related to outdated libraries, so the first step was to upgrade the main libraries and the dependency chain. React was upgraded from version 16 to version 17 and some of the class components were refactored to function components for easier management of state and side effects with hooks [11]. Since it is important to keep the balance between delivering new features and maintenance, it was not directly upgraded to React 18. But that is certainly in the next step of the migration. The Node.js image for Docker was also upgraded from 14-alpine to 18-alpine and accordingly, a series of refactoring on the affected components were involved to ensure the application is compatible with Node.js 18.

TANGOGQL

TangoGQL is a GraphQL [12] web server which integrates with Tango database and can directly communicate with Tango devices [13]. TangoGQL is used as a back end service in Taranta project and connects Taranta and the Tango control system. The module diagram is shown in Fig. 9, where TangoGQL mainly is built on top of PyTango [14], aiohttp [7] and Graphene [15]. PyTango is used for querying the Tango database at a specified tango host, and commu-

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-FR2BC003

nication with Tango devices. Aiohttp is a HTTP server for Python. Graphene maps the predefined GraphQL schema to the endpoints and connects GraphQL queries with the Tango system.

To improve the application performance, the communication with TangoGQL is also optimized for Taranta version 2.x. The task cancellation is optimized so that the subscription can be smoothly terminated even before all listeners have been set up for a subscription. It reduces the chance to cause resource leakage in the long run, which could affect performance. Previously, only exported tango devices was fetched due to the performance issue. However, it was preventing properties access on non-running devices. With the communication refactoring in Taranta version 2.x, the performance issue has been substantially mitigated. Consequently, the query has been adapted to not require a queried device to be exported. It will simply return the device information as long as it is registered in the Tango database. If the queried device does not exist, an error would be returned to the front end.

In addition, the linting in continuous integration (CI) pipeline have been enhanced and the documentation for local development has been revised to facilitate the development process for developers.

To keep up with the CPython release [16], the call for TangoGQL to support Python 3.10 is crucial. Package graphql-ws is used to support the websocket communication for GraphQL but unfortunately it is incompatible with Python 3.10. Therefore, a new library is needed for a replacement. A preliminary investigation was done and Ariadne was found to be a possible replacement on graphql-ws. Ariadne is a python library for building GraphQL APIs. It simplifies the process of creating and serving GraphQL APIs in Python by providing a framework for defining schemas, resolvers, and handling GraphQL queries and mutations [17]. A prototype of integrating Ariadne in TangoGQL was developed and tested working with Taranta. The drawback of this approach is that there are implicit dependencies, which makes the implementation not straight forward. More work on completing the GraphQL schema and better structuring the code will be looked into as the next step in TangoGQL.

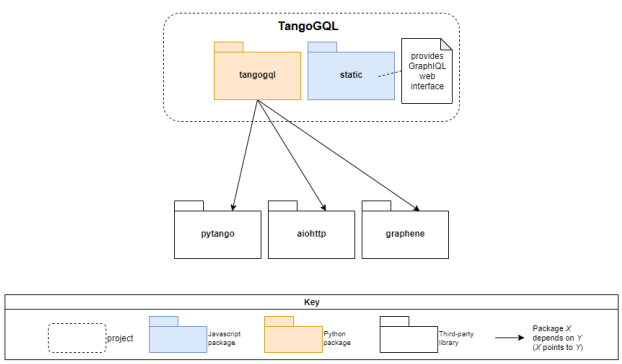


Figure 9: Taranta back end module view.

ROADMAP AND FUTURE

The goal of Taranta project is to build a no-code platform to enable the facility users to easily develop their own user interface. With its growing interest to a broader user base, this project has gained higher expectations on its possibilities. Accessing multiple tango databases is on the top list. Some user groups need to access to multiple databases' devices. For example, for the vacuum team at MAX IV, it is important to access vacuum devices throughout the facility, to be able to analyze the system performance. However, current Taranta was built to only support communications within one database. We have developed a prototype for enabling cross-database communication from the Dashboards view, and we will persist on developing a comprehensive and mature solution.

Another interesting topic is supporting SVG synoptic viewing on Taranta. The synoptic view is a holistic perspective of a particular control system. It's an overview which allows users to understand the entirety of a complex system by presenting essential information in a clear and concise manner. Since the synoptic application at MAX IV is developed as a desktop application, it is difficult for the users to access from outside of the control room. Supporting synoptic views from Taranta would make remote access much easier and the owner of the synoptic view can immediately update it without repackaging and installation.

Additionally, dashboard versioning with MongoDB and a better management on widgets are also considered in our future work. Moreover, modularization of widget development is also considered an important component of our roadmap.

CONCLUSION

With the development of Taranta and the growth of the user community, challenges and potentials have arisen. To better align Taranta with prospective user requirements, a series of improvements have been put into effort in this project. This article presents the enhancements that have been implemented in Taranta version 2.x, encompassing aspects such as communication architecture, widgets refinements, package management and backend communication. In the new released version, critical vulnerability issues have been largely resolved, four new widgets have been developed and a notable enhancement has been achieved in terms of runtime performance, resulting in a 50% improvement in both scripting and rendering time. Looking ahead, continued efforts are needed to develop new features, further optimize the TangoGQL communication, expand current widget functionality to accommodate diverse user requirements.

REFERENCES

[1] M. Eguiraun *et al.*, "Web Interface to Tango Control Systems at MAX IV", in *12th NOBUGS Conference, New Opportunities for Better User Group Software*, 2018.

[2] M. Eguiraun, V. Alberti, A. Amjad, M. Canzari, J. Forsberg, V. Hardion, *et al.*, "Taranta, the No-Code Web Dashboard in

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: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-FR2BCO03

Production", in *Proc. ICALEPCS'21*, Shanghai, China, Oct. 2021, pp. 1017–1022. doi:10.18429/JACoW-ICALEPCS2021-FRAR01

[3] A. Götz *et al.*, "The Tango Controls Collaboration Status in 2021", in *Proc. ICALEPCS'21*, Shanghai, China, Oct. 2021, pp. 544–549. doi:10.18429/JACoW-ICALEPCS2021-WEAR01

[4] Redux Introduction, https://redux.js.org/introduction/

[5] M. Canzari *et al.*, "Improving Performance of Taranta: Analysis of Memory Requests and Implementation of the Solution", presented at ICALEPCS 2023, Cape Town, South Africa, 2023, paper TUPDP044, this conference.

[6] aranta - Tango on web, https://taranta.readthedocs.io/en/latest/widgets.html

[7] Asynchronous HTTP Client/Server for asyncio and Python, https://docs.aiohttp.org/en/stable/web_reference.html#aiohttp.web.StreamResponse.enable_compression

[8] Sardana 3.4 Documentation, https://www.sardana-controls.org/users/overview.html

[9] D. Zubarev and I. Skarga-Bandurova, "Cross-Site Scripting for Graphic Data: Vulnerabilities and Prevention", *Proc. DESSERT 2019*, 2019, pp. 154-160. doi:10.1109/DESSERT.2019.8770043

[10] A. Barth *et al.*, "Robust Defenses for Cross-Site Request Forgery", *15th ACM conference on Computer and communications security*, 2008, pp. 75–88. doi:10.1145/1455770.1455782

[11] D. Bugl, *Learn React Hooks: Build and refactor modern React.js applications using Hooks*. Packt Publishing, 2019.

[12] GraphQL, A query language for your API, https://graphql.org/

[13] TangoGQL, https://gitlab.com/tango-controls/web/tangogql

[14] S. Rubio-Manrique *et al.*, "Dynamic Attributes and Other Functional Flexibilities of PyTango", in *Proc. ICALEPCS'09*, Kobe, Japan, Oct. 2009, paper THP079, pp. 824–826.

[15] GraphQL in Python, https://docs.graphene-python.org/en/latest/quickstart/#introduction

[16] Python Enhancement Proposals, https://peps.python.org/pep-0602/

[17] Ariadne, https://ariadnegraphql.org/docs/intro

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