ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEAG003

INTRODUCING WEB BASED TECHNOLOGIES AT GANIL SPIRAL2 CONTROL SYSTEM

O. Delahaye*, J. Girault, C. Vivier, C. Tréangle, GANIL, Caen, France

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2025). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

Abstract

The SPIRAL2 accelerator began operating in 2021. One of the key application of the control system is the management of all devices parameters (magnets, RF ...), roughly 80000 EPICS variables. That application is fundamental for optimizing the setup time of the accelerator and for easily reproducing the configuration of a given beam from year to year. Because web-based technologies are believed to offer many advantages—such as portability, easier maintenance, optimized use of hardware resources, and centralized security—we decided to evaluate this technology in order to form our opinion in the perspective of a wider renovation project. This paper will explain how the software architecture is designed, both on client and server side, what technologies we used (web framework, REST API, web server, database and ORM). It will also describe the outcomes we achieved in terms of features of the application such as beam characteristics management, reload of a given beam configuration and application to the devices, storage of the accelerator setup, calculation of parameters based on the concept of optic configurations. After 4 months of operation in 2024 with that new application, we will also discuss around the question: are web based technologies a good choice for SPIRAL2 control system user interface?

GANIL ACCELERATOR FACILITY

The GANIL research center currently hosts two accelerators. The first, known as the original GANIL, dates back to the 1980s. The second is the SPIRAL2 accelerator, which features a superconducting LINAC and an EPICS control system — the focus of this paper.

The SPIRAL2 accelerator includes a deuteron source and an ion source with a charge-to-mass ratio (Q/A) of 1/3, along with a LINAC composed of 26 superconducting cavities. Currently, only the NFS (Neutrons For Science) experimental area is operational. The second area, S3 (Super Separator Spectrometer), is in the commissioning phase. See Fig. 1.

Operating the SPIRAL2 accelerator requires control of approximately 3,000 devices, representing around 80,000 physical parameters.

CONTEXT AND MOTIVATIONS

In 2021, the SPIRAL2 facility entered operational mode following the successful commissioning of a deuteron beam in the NFS experimental area for neutron production. In 2022, the LINAC delivered its first oxygen beam. Over the years, the increasing diversity of available beams (argon, proton, oxygen, etc.) has made the use of a dedicated tuning application essential. The operations team requires a powerful tool capable of efficiently tuning the accelerator's 3,000 devices within a short time frame. This application must be able to:

- Facilitate complex parameter calculations using the concept of optical configurations and transmit the results in a single operation to the targeted devices
- Store all parameter values associated with a given beam in a database and easily reload them for future experiments
- Retrieve live values from a set of devices in one operation and compare them to those stored in the database
- Manage changes in energy, optics, and beam characteristics

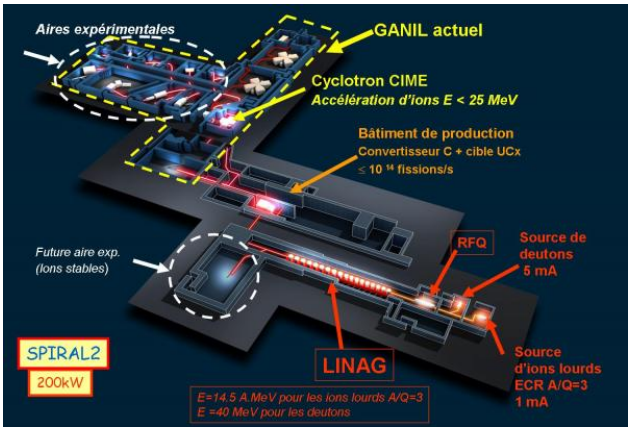

Figure 1: SPIRAL2 beam path.

TECHNOLOGIES CONSIDERED FOR THE APPLICATION

The SPIRAL2 control system is based on the EPICS framework. Several architectural options were considered for developing this application:

- **Thick client**, which is the current solution at SPIRAL2, using either Phoebus or a Java Swing framework interfaced with the Java Channel Access (JCA) library to access Process Variable (PV)
- **Thin client**, such as Python QT (PyQt) application, interfaced with web services to access both the PV and databases
- **Web client**, developed with frameworks like Google Flutter or React, communicating via web services

For the thick client approach, using EPICS CSS-Studio Phoebus was not feasible due to the complexity of the application, which involves database integration, advanced algorithms, and the use of sophisticated GUI components. Although an alternative could have been our existing Java

* olivier.delahaye@ganil.fr

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEAG003

Swing framework, our experience at GANIL has shown that Java is not the most suitable language for developing modern graphical user interfaces.

Moreover, with GANIL planning a major IT infrastructure update in 2026, it was decided to centralize security policies on the server side. This strategic direction led us to consider only thin or web clients.

We evaluated several web technologies, including Google Flutter and React Java Script (React JS). While React is widely adopted in the web development industry, it is not inherently cross-platform, which could pose a limitation in case performance issues require native deployment. Flutter, on the other hand, offers a significant advantage as a complete framework built around its own language, Dart, which is syntactically close to Java — a language with which GANIL teams are already proficient. This made the learning curve for Flutter considerably smoother. We also considered PyQt as a potential solution. Its main advantage lies in the simplicity and speed of development. However, the graphical interface is clearly less visually appealing compared to what can be achieved with Flutter. Additionally, PyQt is not compatible with web browsers, which means it requires local deployment and consumes more local system resources — a drawback given our goal of centralizing infrastructure and minimizing client-side dependencies. See Table 1 for a comparison between thick, thin and web client technologies.

Table 1: Comparison of Thick, Thin and Web Client

Feature	Thick Client	Thin Client	Web Client
Processing	Local	on server	on server
Installation	Required on each workstation	Minimal local installation	No installation, browser-based
Performance	Fast, uses local resources	Dependent on server	Dependent on Internet connection
Maintenance / Updates	Must update each workstation	Centralized on server	Centralized on server
Connectivity	offline	network connection	Internet connection
Security	dependent on workstation	centralized policy on server side	centralized but depends on web security
debug	Local logs, full access	Server logs	Server logs + browser dev tools

SOFTWARE ARCHITECTURE

The application is based on a classical client-server architecture. See Fig. 2.

- On Front end, the UI is developed using **Google Flutter.**
- On back end, 2 web services are implemented:
  - A channel access web service to interact with EPICS PV
  - A database acess service using with an Object Relationnal Mapping (ORM) framework

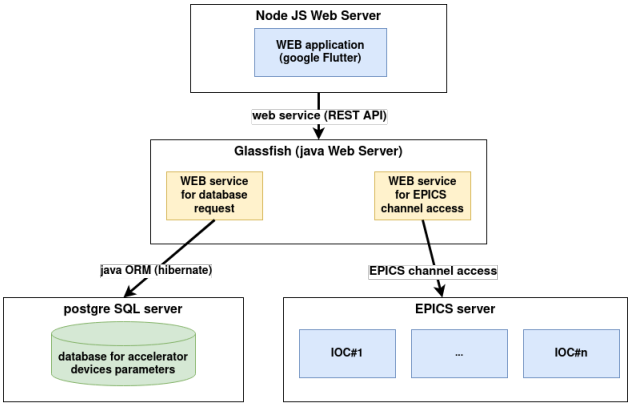

Figure 2: Software architecture.

The main advantage of this architecture is the ability to distribute development tasks efficiently. Each web service can be developed and tested independently, which greatly facilitates team collaboration and modular development.

On the client side, Google Flutter enables the creation of a responsive and modern interface. The application is accessible from any location via a standard web browser, requiring no local installation.

Web Services

All web services are developed using the **REpresentational State Transfer (REST)** architecture (based on the **OpenAPI 3.x** specification) due to its simplicity, lightweight nature, and efficiency. REST requires fewer system resources compared to Simple Object Access Protocol (SOAP) and benefits from features such as HyperText Transfer Protocol (HTTP) caching, which helps reduce server load. Being stateless and flexible, REST supports multiple data formats, including JavaScript Object Notation (JSON), Yet Another Marker Language (YAML), and eXtensible Markup Language (XML) — allowing for versatile integration.

To design and implement the REST APIs, we used **Swagger**, which is part of the OpenAPI ecosystem. Swagger provides a powerful suite of tools for:

- Defining web service APIs using the Swagger Editor, based on OpenAPI files written in **YAML** or JSON, and automatically generating endpoint documentation.
- Testing endpoints with Swagger User Interface (UI), which generates an interactive interface directly in the browser.

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2025). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEAG003

© Content from this work may be used under the terms of the CC BY 4.0 licence (© 2025). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

- Generating source code in various languages (**Java**, Python, C#, TypeScript) from the OpenAPI specification.

For code generation, we selected Java to maximize reuse of existing SPIRAL2 components. Swagger enables code generation for the **Jersey** framework, which is based on **(Java API for RESTful Web Services) (JAX-RS)**. JAX-RS is fully integrated into the Java Enteprise Edition (Java EE) ecosystem, allowing deployment on a Java EE-compliant server. It supports both JSON and XML data formats. We chose JSON for its simplicity and widespread use. JAX-RS also provides a robust exception handling mechanism via ExceptionMapper, facilitating clean error management.

The services currently run on a **GlassFish** java web server, as it was already deployed at GANIL. However, due to GlassFish becoming outdated (limited support, fewer updates, and suboptimal performance in modern environments), we plan to migrate to **Payara** Server, an actively maintained and enhanced fork of GlassFish, which offers better support and performance optimizations.

EPICS Channel Access Web Service The REST web service provides access to accelerator devices through **EPICS Process Variables (PVs)**. It exposes a set of simple endpoints and supports both **HTTP** and **WebSocket** protocols for communication:

- `GET /channel-access/read` — Reads the current value of a specified PV
- `GET /channel-access/write` — Writes a given value to a specified PV
- `GET /channel-access/status` — Retrieves the connection status of a PV (notification feature is under consideration)

The service is built on the **JCA** library, which enables interaction with the EPICS control system from Java applications.

Database Web Service The second web service provides access to the database where all accelerator device parameters and the various accelerator configurations—which depend on the beam to be accelerated—are stored. The database has been fully redesigned to support the needs of the new application.

It contains about **60 tables** and currently holds approximately **20 MB** of data which is very low because SPIRAL2 doesn't manage a lot of beam configurations for the moment.

We chose **PostgreSQL** as the database management system because our data is **highly structured**, and PostgreSQL offers native support for **complex data types** (such as `JSON`, arrays), and is **optimized for handling large volumes of data**. Moreover, it is **highly scalable**, which is critical as the volume of stored data will continue to grow over time.

All database access is handled through **Java Hibernate ORM**. The key advantage of using Hibernate is **database abstraction**: if we decide to switch to another database technology in the future, the core web service code remains unchanged. Additionally, the **PostgreSQL Java Database Connectivity (JDBC) driver** is mature and integrates seamlessly with Java frameworks like Hibernate.

The web service offers a number of endpoints grouped into functional categories:

Beam Configuration

- `GET /load-beam/{beam-id}` — Load a specific beam configuration
- `GET /list-beams` — Retrieve the list of available beams
- `GET /list-archives/beam/{beam-id}` — Retrieve archived device parameters for a given beam
- `GET /loaded-beam` — Get the configuration of the currently loaded beam

Device Parameter Read Operations

- `POST /list-devices` — Get the list of devices for a given section of the accelerator
- `POST /read/theoretical/beam/{beam-id}` — Read theoretical values of devices for a given beam
- `POST /read/parameters/beam/{beam-id}` — Read the configured parameter values of devices for a given beam
- `POST /read/archived/beam/{beam-id}` — Read archived parameter values for a given beam

Device Parameter Storage Operations

- `POST /storage/beam` — Temporarily store device parameters (acts as a buffer)
- `POST /archiving/beam` — Archive device parameters after tuning, enabling reuse in future operations

Optic Configuration

- `POST /list-optics` — Get the list of available optical configurations
- `POST /change-optic/optic/{optic-id}` — Modify the optic configuration for the selected accelerator section

Energy Configuration

- `POST /change-energy/{energy}` — Adjust device settings to match a specified energy level

Web Application

The application is entirely developed using **Google Flutter**, with a Software Development Kit (SDK) version constraint of `>=2.17.5 <3.0.0`.

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEAG003

Synoptic Views Synoptic diagrams (see Fig. 3) are implemented using **Scalable Vector Graphics (SVG) technology**, utilizing the `flutter_svg` and `svg_path_parser` packages. This approach enables **interactive synoptics**, where users can click on specific parts of the diagram to select or interact with elements.

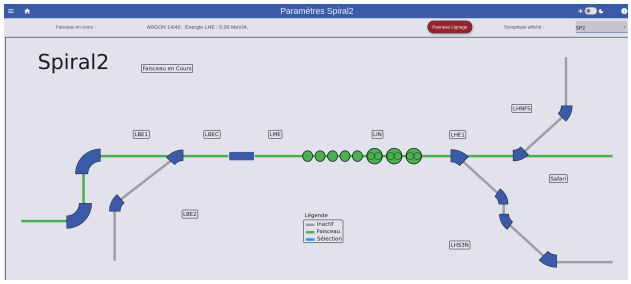

Figure 3: Synoptics for selecting an accelerator section.

Modular Code Architecture To ensure code reusability and maintainability across future projects, the codebase has a modular architecture (using Git submodules) and uses the **Model-View-ViewModel (MVVM) pattern**, especially:

- **Common UI Module**: Contains shared components and utilities across multiple applications, such as Starter templates, Page layouts, Standard UI widgets (e.g., text fields, labels, tables, popups, and line charts)
- **Application-Specific Module**: Contains widgets and logic tailored to this application domain, including Optical configuration selectors, Beam parameter forms, Synoptic views and associated logic.

Backend Communication Backend interaction is handled through two dedicated modules, each designed with a clear separation of concerns:

- One module is responsible for communication with the **Database Web Service**
- The other handles interactions with the **EPICS Channel Access Web Service**

This modular architecture ensures a clean separation of responsibilities, which significantly improves the testability, maintainability, and scalability of the application. It also facilitates future adaptation to other control systems or accelerator infrastructures.

APPLICATION FEATURES

Application Menu

The menu (see Fig. 4) on left side allows to select the action to perform.

- read device values (live, theorical, stored, archived)
- Set theorical / stored / archived values to the devices
- compare live device values with theorical / stored / archived values
- save device values (archiving or storage)
- load a beam configuration
- load an optic configuration

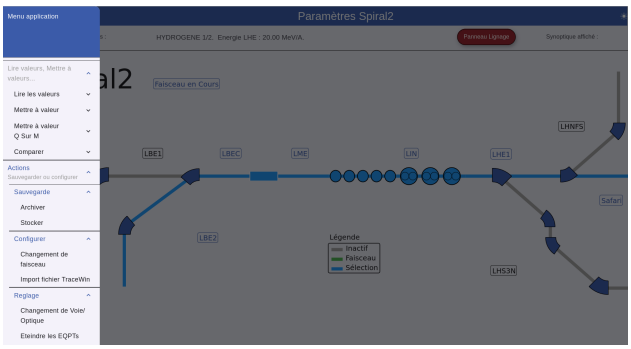

Figure 4: Application menu.

Loading a Beam Configuration

This process involves retrieving all device parameters associated with a given beam configuration—defined by its mass number (A), charge state (Q), atomic number (Z), energy, and intensity—from the database. See Fig. 5.

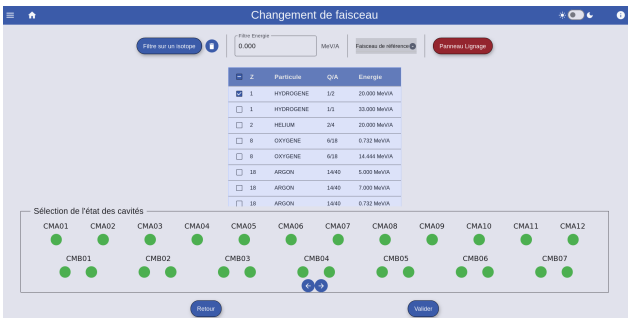

Figure 5: Loading a beam.

This ensures that the operator is fully prepared to begin working with the beam and proceed with tuning it on the accelerator.

Calculation of Magnetic Device Parameters Based on the Concept of Optic Configurations

Magnetic devices constitute the majority of components in an accelerator. They play a key role in shaping and steering the beam at various sections of the machine. To manage this, the application uses the concept of an optic configuration.

Defining an optic involves assigning, for each magnetic device, a magnetic field gradient value (in T/m) corresponding to a given reference magnetic rigidity Bρ (in T·m). All optics configurations are stored in the database in the following format:

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2025). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEAG003

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2025). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

Table 2: Defining an Optic Configuration

Device	Optic	Gradient	Reference Bρ
LHE-Q11	standard	3 T/m	1 T.m
LHE-Q11	NFS	2,5 T/m	1 T.m
LHE-Q11	S3	1,5 T/m	1 T.m
LHE-Q12	standard	3 T/m	1 T.m
LHE-Q12	NFS	2,5 T/m	1 T.m
LHE-Q12	S3	1,5 T/m	1 T.m

It should be noted that the beam energy is directly linked to the magnetic rigidity (Bp). Therefore, for a given energy and a specific optics configuration, the gradient value for each device can be calculated using a straightforward cross-multiplication. To generate the necessary magnetic field, the application then determines the corresponding current to apply to each power supply, using a conversion from gradient to current based on a predefined calibration curve.

This approach, widely used at the GANIL CYCLOTRON accelerator, enables the configuration of SPIRAL2 accelerator's magnetic elements based on the beam energy and optics settings. The operator only needs to select the optics for a specific section of the accelerator (see Fig. 6), and the application calculates all the necessary parameters. These parameters are then saved as so-called theoretical values.

Figure 6: Optic selection at SPIRAL2.

Set Values on the Devices

If the beam has never been configured on the accelerator (for example, during an initial tuning), the operator can apply the so-called theoretical values from the database to the devices.

If the accelerator has been tuned with the same beam in previous years, the operator can apply a more precise configuration retrieved from the archives (see Fig. 7).

Compare the Device Values

Once the configuration is loaded, the operators fine-tune the accelerator to achieve the optimal settings. At any time, the operators can compare archived or stored values with the live device values to verify if any issues have occurred. See Fig. 8.

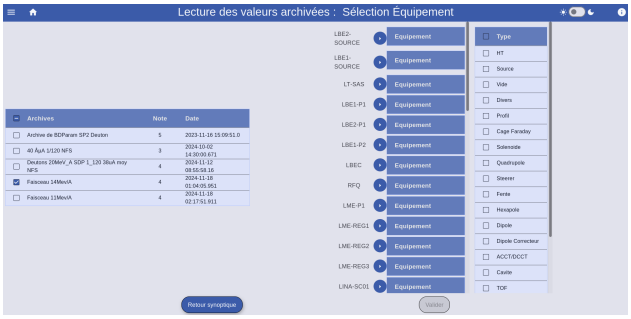

Figure 7: Set values on the devices.

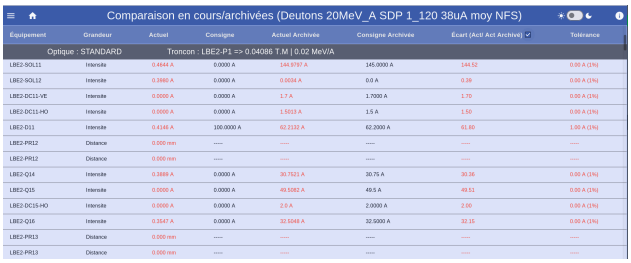

Figure 8: Comparison of devices values.

Read Device Values

At any time during the tuning, the operators can read the following device values to check the accelerator configuration as shown in Fig. 9: live, stored and archived values.

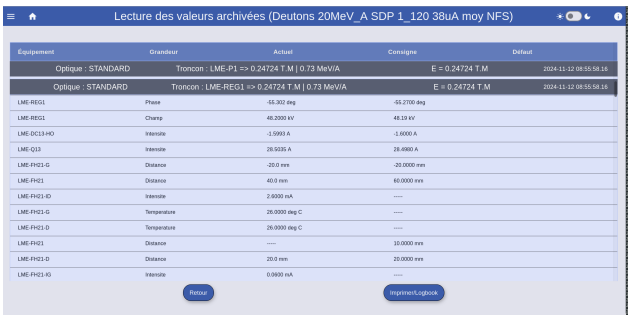

Figure 9: Read device values.

Save the Beam Configuration

If they need to pause the tuning process and resume it later, they can save the current device configuration to a temporary buffer using the storage feature.

When the tuning is finalized and meets the requirements, they use the archiving feature to save the configuration in the database (see Fig. 10). Unlike the temporary storage, archived configurations can be reloaded for future experiments.

OUTCOMES

Usability

Compared to Java Swing, which was the technology previously used at SPIRAL2 for application development, this

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEAG003

Figure 10: Archiving a beam configuration.

represents a significant step forward. The graphical interface is more attractive, smoother, and much more user-friendly.

Reliability

Flutter is reliable. It is maintained by Google, supports cross-platform development, and offers stability by compiling to native code. The ecosystem is strong, with excellent documentation, an active community, and robust plugins.

Maintainability/Debug

It depends clearly the way the developer organized its code but flutter encourages clear and modular structure with reusable widgets. Flutter uses a declarative approach, which makes UI easier to understand and reduces boilerplate compared to some other frameworks.

In terms of debug, Flutter integrates seamlessly with **Dart DevTools**, providing features like step-by-step debugging, widget tree inspection, performance analysis, and **hot reload** (this feature lets you see UI changes instantly without restarting the app, greatly speeding up development and debugging)

Performances

In terms of performance, it fully meets our requirements. Even if our application is compiled to run in web browsers, it runs flawlessly both Firefox and Chrome.

Security

Flutter doesn't have built-in secure storage but packages are available for that.

Currently, communication between our front end and back end is not yet secured. At this stage, we only have basic login features in place, which allow us to monitor abnormal connections on the backend. This is clearly an area that needs improvement and will be addressed as part of GANIL's IT security policy. We plan to implement HyperText Transfer Protocol Secure (HTTPS) with Secure Sockets Layer (SSL) and Transport Layer Security (TLS) certificates, combined with JSON Web Tokens (JWT) for backend authentication. On the client side, the `flutter_secure_storage` package can be used to securely store the JWT. In addition, backend requests will be protected against Cross-Site Request Forgery (CSRF) attacks to strengthen overall security. We could use the `dio` package, which supports HTTPS and offers more advanced features compared to the standard `http` package.

Deployment

Deploying a Flutter application as a web version is very easy on a Node.js server. However, we currently do not have Continuous Integration and Continuous Deployment (CI/CD) implemented to ensure synchronization between the front end and back end. As a result, the front end may become out of sync when the back end version is updated. To address this, we plan to implement Application Programming Interface (API) versioning and establish automated testing.

Here are in the Table 3 the ratings that could be given to the application.

Table 3: Application Assessment

Criteria	Rating scale (1 to 5)
Usability	5
Reliability	4
Maintainability/Debug	4
Performances	4
Security	4

CONCLUSION

Are web based technologies a good choice for SPIRAL2 control system user interface?

Yes, I believe web-based technologies are a good choice for the SPIRAL2 control system user interface. They work well in our environment, and more importantly, this architecture allows for a clear separation between the graphical interface and the device control logic, where the system's core intelligence resides. If Flutter were to become less relevant in the future, we would be able to switch to another front-end technology with minimal effort and resources, thanks to this modular approach.

LIST OF ACRONYMS

API Application Programming Interface

CSRF Cross-Site Request Forgery

CI/CD Continuous Integration and Continuous Deployment

EPICS Experimental and Physics Inductrial Control System

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2025). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEAG003

HTTP HyperText Transfer Protocol

HTTPS HyperText Transfer Protocol Secure

Java EE Java Enteprise Edition

JAX-RS (Java API for RESTful Web Services)

JCA Java Channel Access

JDBC Java Database Connectivity

JSON JavaScript Object Notation

JWT JSON Web Tokens

MVVM Model-View-ViewModel

ORM Object Relationnal Mapping

PV Process Variable

PyQt Python QT

React JS React Java Script

REST REpresentational State Transfer

SDK Software Development Kit

SOAP Simple Object Access Protocol

SQL Structured Query Language

SSL Secure Sockets Layer

SVG Scalable Vector Graphics

SPIRAL2 Système de Production Ions Radioactifs Accéléré en Ligne 2

TLS Transport Layer Security

UI User Interface

XML eXtensible Markup Language

YAML Yet Another Marker Language

© Content from this work may be used under the terms of the CC BY 4.0 licence (© 2025). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.