ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THPDP073

SciLog: A FLEXIBLE LOGBOOK SYSTEM FOR EXPERIMENT DATA MANAGEMENT

K. Wakonig†, A. Ashton, C. Minotti, Paul Scherrer Institut, Villigen PSI, Switzerland

Abstract

Capturing both raw and metadata during an experiment is of the utmost importance, as it provides valuable context for the decisions made during the experiment and the acquisition strategy. However, logbooks often lack seamless integration with facility-specific services such as authentication and data acquisition systems and can prove to be a burden, particularly in high-pressure situations, for example, during experiments. To address these challenges, SciLog has been developed at the Paul Scherrer Institut. Its primary objective is to provide a flexible and extensible environment, as well as a user-friendly interface. SciLog relies on atomic entries in a database that can be easily queried, sorted, and displayed according to the user's requirements. The integration with facility-specific authorization systems and the automatic import of new experiment proposals enable a user experience that is specifically tailored for the challenging environment of experiments conducted at large research facilities. The system is currently in use during beam time at the Paul Scherrer Institut, where it is collecting valuable feedback from scientists to enhance its capabilities.

INTRODUCTION

Metadata is defined as the data providing information about one or more aspects of the data; it is used to summarize basic information about data that can make tracking and working with specific data easier. [1] It includes, among many, information about the source of the data, its process and responsible people and the location on a computer network where the data was created and collected.

It also covers unstructured information collection, for example, notes on a data acquisition process or keeping track of important TODOs. SciLog [2, 3] was developed specifically with this in mind, namely to improve the storing and reuse of unstructured metadata and as a consequence improve the FAIRness of data [4]. It aims at replacing legacy pen and paper experimental logbooks, often used in large-scale facilities during beamtime and often messy to consult and prone to information loss. It also supports features to monitor an experiment, by watching messages posted by the beamline to it. Each message is an atomic entry in the database, which means that every message can be decoupled from the rest of the environment. It also supports collaborative editing.

First, we will introduce the components of SciLog and a few key concepts that are useful to better comprehend the rest of the article. Then we will move to present the creation of logbooks, the search functionality, the TODOs support and the main logbook widget, which enables displaying, adding and modifying messages. The widget can display messages ordered by date and can provide information about, for example, the time of insertion and the author.

We will address how SciLog can be scaled to accommodate high volumes of metadata and usage.

In order to maximize data dissemination, we will present the Python [5] libraries that have been developed to interact with SciLog to post and get metadata.

We will finally close the article with future directions that the community envisions for SciLog.

DESIGN OVERVIEW

The next sections discuss the technicalities of the implementation of SciLog, including the choice of the underlying technologies and frameworks.

The SciLog stack is organised following a microservice architecture, where each service can be containerised and configured to interact with the others and the pre-existing facility infrastructure, following standard TCP [6] communication protocols, such as HTTP(s) [7] and Web-Sockets [8]. All SciLog services communicate with each other through HTTP or Web-Sockets.

The backbone of the ecosystem is the *backend* which relies on a Mongo Database [9], the connection to which must be configured as part of the setup. The *backend* is also responsible for defining the data model which formalizes the scaffolding of the metadata fields, setting the required ones and leaving great flexibility for customization.

Data Model

The data model defines which information is stored and how it is structured, by mapping the SciLog entities with records on the database.

Each SciLog entity has a representation in the data model in the underlying MongoDB and a subset of fields is controlled by validation rules imposed by the *backend*.

The majority of entities share the same common structure and fields subset, and they differ by adding entity-specific fields or mentioning the entity type specifically in one (*snippetType*). It is often convenient to store dependencies between SciLog entities in the database, and this can be achieved using the concept of one-to-many relationships between documents in MongoDB [10].

The main SciLog entities are:

- Basesnippets
- Locations
- Logbooks
- Images
- Paragraphs
- Tasks

† klaus.wakonig@psi.ch

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THPDP073

Basesnippets: it's the collection which groups all other aforementioned entities. Its purpose is to define fields common to all other entities and apply common rules and logic. It enforces, for example, documents to have a *createdAt* field and of type date. Notably, it ensures that all child entities specify a *snippetType* field, which will allow distinguishing between the type of entities in the database. It also requires child entities to specify ACLs [11], which check authorizations comparing the entities with the user attributes.

Locations: it is the group of *Basesnippets* with *snippetType=location*. It facilitates the assignment of logbooks and entries to a specific instrument, beamline and facility.

Logbooks: distinguished by *snippetType=logbook*, this *Basesnippet* encodes the meaning of a logbook the scientist would allocate when the experiment starts. It is often used to group messages together and can be referenced by *Paragraphs*, as will be explained later. Think of it as the book where the notes on the experiment would belong.

Images: grouped by *snippetType=image*, it's where images or attachments are stored. This makes use of the MongoDB GridFS functionality.

Paragraphs: with *snippetType=paragraph*, it is a representation of the paragraphs a scientist would write in a logbook. They reference *Logbooks* in the *ParentId* field, which enables linking them to the *Logbook* they belong. Its *textcontent* field allows storing the text of the paragraph, and if HTML-encoded it is rendered accordingly in the SciLog *frontend.*

Tasks: as *Basesnippets* with *snippetType=task*, it allows creating TODOs. Each *Task* is a MongoDB document and, as for the *Paragraphs* case, it can reference the *Logbook* the task is applicable to in the *ParentId* field.

Technologies

The core services of SciLog are the *backend*, encapsulating the core logic of the data catalogue, and the *frontend*, presenting the information to the users.

The *backend*, providing the RESTful API [12], user authentication and authorization, data management and the interface to the database, is the portion of SciLog running server-side which operates on and stores the information in the underlying database. It manages the user login and enforces access permissions. It is developed in Typescript [13] using the framework Loopback4 [14]. It uses MongoDB as a database through a Loopback4-specific MongoDB ORM [15]. The *backend* implements the classical REST API with ACLs for authorization. It supports local administrative accounts and OIDC authentication [16]. It supports all CRUD [17] operations to operate on metadata records, such as *Basesnippet*, *Logbooks*, or *Paragraphs* creation, updates and display.

The *backend* provides a Swagger UI [18] which exposes the endpoints, detailing the expected input and output formats and types. It also complies with the OpenAPI initiative [19].

It allows configuration through environmental variables and JSON files. The administrators can deploy the backend on bare OS or within a Docker [20] container through an orchestration system like Kubernetes [21]. The latter is the preferred solution in many adopting facilities. The minimum required configuration consists of setting the connection to the database. The backend allows for additional customisation, although it would also require a customised deployment and additional existing infrastructure, for example, a connection to an Identity Access Management [22] system which supports OIDC.

The *frontend* is the portion running on the client browser. It is a single-page [23] application developed with the Angular framework [24] which is based on Typescript. It provides a user-friendly UI to display and search *Logbooks*, according to the user's authorization. It supports the user to create customised views of *Logbooks*, including the *Paragraphs* and *Task* CRUD operations. It only presents information and for the computation leverages the backend REST API.

Scalability

SciLog harnesses the power of Docker containers and Kubernetes orchestration to achieve seamless scalability and efficiency. By embracing the containerization paradigm, SciLog empowers research institutions and laboratories with a sophisticated solution, enabling them to effortlessly store, "organize, and retrieve metadata while adapting to the ever-evolving demands of data volumes.

In this ecosystem, Docker containers provide flexibility and consistency. These self-contained units encapsulate the entire metadata catalogue along with its dependencies, ensuring that each environment remains isolated and coherent. This approach not only simplifies the deployment process but also guarantees that the application functions uniformly across various setups.

SciLog publishes builds of its code as containers that can be deployed in a variety of container deployment solutions, such as Kubernetes.

MongoDB as the underlying database ensures fast querying together with powerful search capabilities.

FEATURES AND FUNCTIONALITIES

We move now to present some of the features SciLog provides. In the following sections, different capabilities are presented, focusing in particular on the most used.

Authentication, Authorization and Sharing

SciLog supports the OIDC protocol, thus allowing for flexible user authorization without the need to create new accounts for users and for the facility to manage multiple users' identities. SciLog can leverage the internal authorization mechanism if needed, which can be expanded using its sharing functionality.

Logbooks Overview and Search

After login, the users are confronted with a page showing the list of *Logbooks* (i.e. experiments) they can access. Each displays a minimal subset of information, to ease the browsing. The user can also search across *Logbooks*, with the search bar at the top of the page, as shown in Fig. 1.

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THPDP073

This sends a query to the *backend* to the *Logbooks* endpoints.

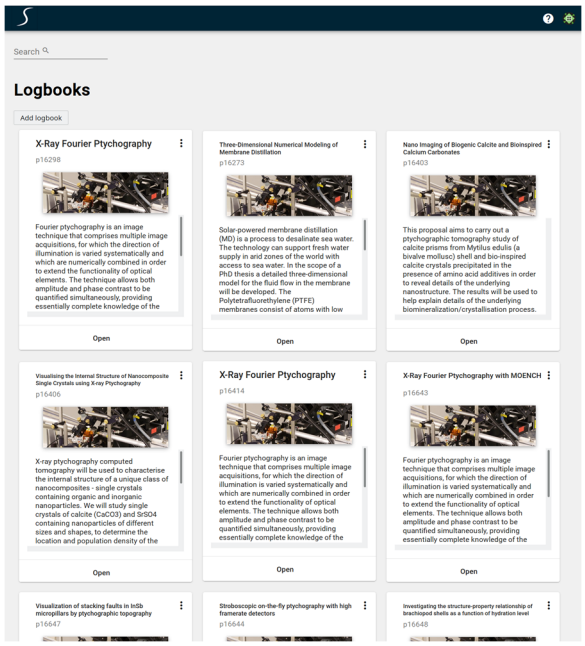

Figure 1: Example overview of *Logbooks*. In the top left, the *Search* can filter the displayed *Logbooks*.

Logbook Display

Selecting a *Logbook* brings the user to a *View* which formats information in a format previously selected by the user or applies a default. Typically, the user would see a widget containing the list of *Tasks* relative to the *Logbook*, and a "chat-like" widget where members of the experiment can send, edit and delete messages collaboratively. An example is shown in Fig. 2.

It is worth briefly introducing the distinction between a *Logbook* and a *Logbook View*, or simply a *View*. A *Logbook* is an entity in the database that can be compared to a "book", as mentioned before. A *View* is an additional document in the database that stores information on what to display when a particular user clicks on a *Logbook* from the overview page. The *View* controls what the user sees and it is not bound to display information on a single *Logbook*. As part of the configuration of the *View* from the UI, the user can in fact select which *Logbooks* to display in every widget of the *View*. We will then call the *Logbook* the user clicked on in the overview *Target Logbook* and the list of additional ones *Additional Logbooks*. Different widgets in the same *View* can have different *Target Logbooks* and *Additional Logbooks*, which can be set individually by the user. The default is to set the *Target Logbook* to the clicked one and *Additional Logbooks* as an empty list.

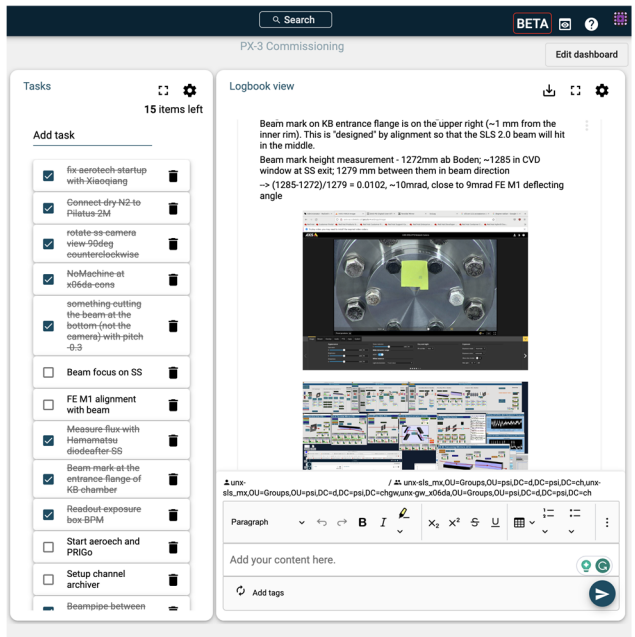

Figure 2: Example of a *Logbook View*. On the left, the *Task* widget exemplifies the task support. The widget on the right shows the "chat-like" *Logbook* widget.

Tasks Manipulation

Using the *Task* widget the user can create, delete or mark as done tasks that should be remembered. The left part of Fig. 2 shows a typical *Task* widget usage, guiding the user through the beamtime.

Paragraphs' Manipulation

In the *Logbook* widget, the user can see paragraphs relative to the *Target Logbook* and *Additional Logbooks*, sorted in descending or ascending order and can append further paragraphs. Multiple types of *Paragraphs* are supported, such as simple text, images, attachments, code snippets, etc. Users can also comment, edit, and reply to a *Paragraph*, as shown in Fig. 3.

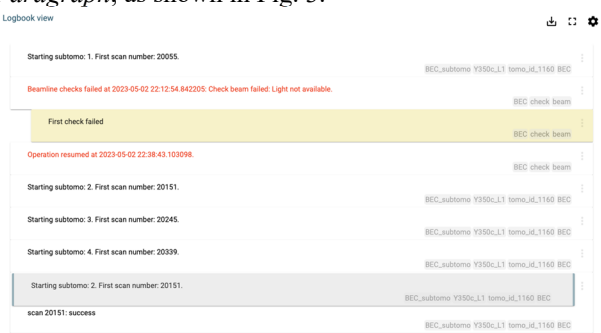

Figure 3: Example of *Paragraphs* types. Normal logbook entries are shown in white. In addition, users can reply to already existing entries with a greyed-out reference to the original entry. Alternatively, entries can be further annotated using comments, highlighted in yellow.

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THPDP073

Paragraphs can also be tagged, which can be used to later filter Paragraphs of a *Target Logbook*, as shown in Fig. 4.

Figure 4: Example usage of tags as filters.

Ingestion

It is often useful to automatically create an empty *Logbook* on experiment proposal approval. SciLog covers this by exposing REST APIs for each entity and providing a Python SDK [25] which can ease its interaction. This means that a facility could easily implement appropriate Python scripts that are responsible for POSTing *Logbooks* on proposal approval, assigning the *Logbook* to the correct group and setting preliminary information. This solution is currently in place at PSI.

OUTLOOK

We aim to increase the number of features supported by SciLog, in particular by extending the Python SDK and the UI, for example, adding support for *Threads* in the *Logbooks* widget, which would allow users to start communication threads from a *Paragraph*. The widget would then group these together and show them accordingly.

We also plan to extend the authorization model, moving from a simple *reader-owner* model to a full-fledged role-based one.

Enabling other beamlines at PSI to roll out SciLog is given great importance, as the more users the greater the commitment, and the faster the development.

Linking the SciCat data catalogue [26, 27] with SciLog is also looked at, as preserving logbooks as part of experimental metadata is more and more becoming a fundamental need.

CONCLUSION

The development of the Python SDK and the REST API that the backend implements have proven a successful choice as they have fostered the development of tools to either automatically create *Logbooks*, automatically POST messages and finally export *Logbooks* from other ELNs, translate and import into SciLog.

The growing adoption of beamlines promises to enhance SciLog's visibility and foster a more extensive and vibrant community, both within and beyond PSI.

ACKNOWLEDGEMENTS

We would like to warmly thank Dr. Stephan Egli who massively contributed to the development of SciLog, actively and through his strenuous support.

We wish him a happy retirement.

REFERENCES

[1] *A Guardian Guide to your Metadata*, The Guardian, Jun. 2013, theguardian.com

[2] SciLog, https://paulscherrerinstitute.github.io/scilog

[3] SciLog at PSI, https://scilog.psi.ch

[4] FAIR, https://www.go-fair.org/fair-principles

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

[6] *Transmission Control Protocol*, Wikipedia, Sep. 2023, https://en.wikipedia.org/wiki/Transmission

[7] *HTTP*, Wikipedia, Sep. 2023, https://en.wikipedia.org/wiki/HTTP

[8] *WebSocket*, Wikipedia, Sep. 2023, https://en.wikipedia.org/wiki/WebSocket

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

[10] MongoDB One-to-Many, https://www.mongodb.com/docs/manual/tutorial/model-referenced-one-to-many-relationships-between-documents

[11] *Access-control list*, Wikipedia, May 2023, https://en.wikipedia.org/wiki/Access-control_list

[12] *REST*, Wikipedia, Sep. 2023, https://en.wikipedia.org/wiki/REST

[13] Typescript, https://www.typescriptlang.org

[14] Loopback4, https://loopback.io/doc/en/lb4

[15] Loobpack4 MongoDB connector, https://loopback.io/doc/en/lb4/MongoDB-connector.html

[16] *OpenID Connect*, Wikipedia, Aug. 2023, https://de.wikipedia.org/wiki/OpenID_Connect

[17] *Create, read, update and delete*, Jul. 2023, Wikipedia, https://en.wikipedia.org/wiki/Create,_read,_update_and_delete

[18] Swagger UI, https://swagger.io/tools/swagger-ui

[19] OpenAPI initiative, https://www.openapis.org

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

[21] Kubernetes, https://kubernetes.io

[22] *Identity management*, Wikipedia, Aug. 2023, https://en.wikipedia.org/wiki/Identity_management

[23] *Single-page application*, Wikipedia, Sept. 2023, https://en.wikipedia.org/wiki/Single-page_application

[24] Angular, https://angular.io

[25] SciLog Python SDK, https://paulscherrerinstitute.github.io/scilog/Ingestor/PythonSDK.html

[26] SciCat, https://scicatproject.github.io

[27] SciCat at PSI, https://discovery.psi.ch