ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP014

BLUESKY WEB CLIENT AT BESSY II

Huiling He[1*], William Smith[1†], Sebastian Sachse[1‡], Gabriel Preuß[2§], Ruslan Ovsyannikov[2¶]
Helmholtz Zentrum Berlin, Berlin, Germany

Abstract

Considering the existing Bluesky control framework at BESSY II, a web client with React based on Bluesky HTTP Server is being developed. We hope to achieve a cross-platform and cross-device system to realize remote control and monitoring of experiments. The implemented functionalities for now are monitoring of the Bluesky Queue Server status, controlling over a Bluesky Run Engine environment, browsing of Queue Server history as well as editing and running of Bluesky plans. Challenges around the presentation of live data using Tiled are explored. This work builds on that of NSLS II who created a React based web interface and implements a tool for BESSY II.

INTRODUCTION

In today's era of rapid digital innovation, web applications are playing an increasingly significant role in experiment control. This paper presents a comprehensive exploration of the Bluesky [1,2] control system and its web user interface and offers insights into the integration, development, and core functionalities.

The motivation behind the innovative web-based interface is twofold. Firstly, it arises from the recognition of the potential of web applications. Secondly, it aligns with the adoption of the Bluesky project at Bessy II [3]. Although current Bluesky applications at Bessy II are underutilized, they hold great promise for the future of control systems. Understanding the scalability and versatility of Bluesky control is crucial. Web user interfaces are needed to meet the dynamic demands of experimental setups and simplify the experiment process. Moreover, a web client empowers remote control, enabling users to initiate, monitor and manage experiments from anywhere with internet access.

Our primary goal of the web client is to gain a comprehensive understanding of the Bluesky QueueServer [4] and its interaction with web operations. This encompasses real-time status monitoring, console tracking, experimental data display and experiment execution. These web-based capabilities aim to replace conventional reliance on manual instructions with command-line interfaces or Qt GUIs [5] at experimental stations.

This article delves into the architecture of web interactions with related back-end servers and databases and provides a detailed exploration of the functional components for the Bluesky web client. In the following sections, we will introduce the system architecture, back-end servers with Docker usage, layout and functionalities of the web client, software implementation and conclusion.

* huiling.he@helmholtz-berlin.de
† william.smith@helmholtz-berlin.de
‡ sebastian.sachse@helmholtz-berlin.de
§ gabriel.preuss@helmholtz-berlin.de
¶ ovsyannikov@helmholtz-berlin.de

BLUESKY WEB CLIENT

System Architecture

The system architecture illustrated in Figure 1, follows a client-server model with two distinct facets: one for experiment control and the other for data display. Bluesky [1] components are responsible for experiment control, while Tiled [6] is dedicated to data presentation.

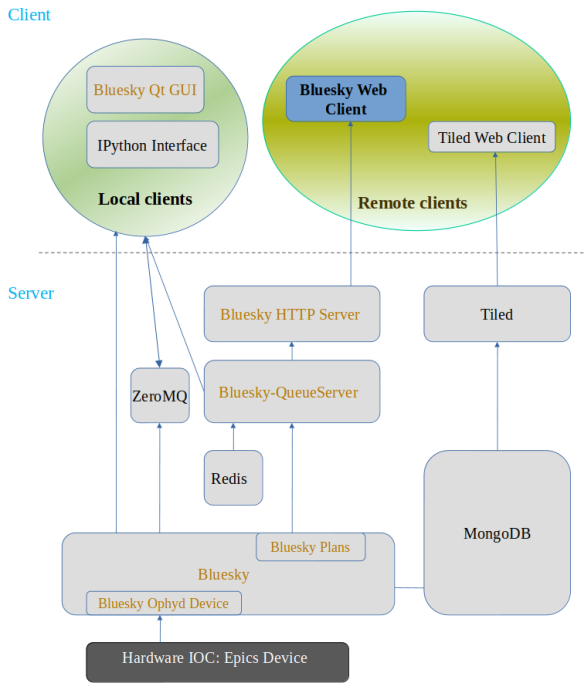

Figure 1: Server-Client architecture.

On the server side, Bluesky components are configured and managed within an IPython profile, which grants the Bluesky QueueServer [4] complete authority. Consequently, any authorized client can interface with the Bluesky environment via the Bluesky QueueServer [4]. This architecture has been successfully deployed at BESSY II [3], accommodate the demand for Bluesky control with IPython command-line tools, Qt GUIs [5] and web clients.

The Bluesky Queue Server [4] introduces a queuing system that facilitates experiment management, enabling researchers to execute experiments in a queue-based manner. This works as the cornerstone of the Bluesky web client, meanwhile, Bluesky HTTP Server [7] operates as a communication bridge to simplify and streamline the interactions between user operations on the web client and Bluesky QueueServer [4].

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP014

Turning to the client side, the web client integrates with the Bluesky HTTP Server [7] via HTTP requests and HTTP responses. These enable data retrieval from the Bluesky QueueServer and support data submission to it.

For data storage and retrieval, the combination of MongoDB [8] and Tiled [6] take the reins, serve as repositories for experiment data. With the configuration in Tiled server, We can view the data from a certain MongoDB on the Tiled web client. The interaction among Tiled [6] and MongoDB [8] is also illustrated in Figure 1. Exploration for real-time data plotting with Tiled is underway.

Docker Usage

In the system architecture shown in Figure 1, there are multiple servers with complex dependencies that need to support the web client. In order to run and manage the multiple servers, Docker Compose [9] is used. Docker Compose is a tool of Docker [10]. We can use Docker Compose to streamline the setup and management of our required services. It provides a consistent and reproducible environment that can be easily shared and scaled. Currently, there are seven services for the Bluesky web client: `mongodb`, `redis`, `zmq-proxy`, `bluesky-queueserver`, `bluesky-httpserver`, `tiled`, and `bluesky-webclient`. In the Figure 2, we can see the seven services hosted in Docker Compose.

Name	Command	State	Ports
bluesky-webclient	docker-entrypoint.sh yarn start	up	0.0.0.0:3000->3000/tcp,:::3000->3000/tcp
bluesky-httpserver	./start.sh	up	0.0.0.0:60610->60610/tcp,:::60610->60610/tcp
bluesky-queueserver	./start_re.sh	up	0.0.0.0:60615->60615/tcp,:::60615->60615/tcp, 0.0.0.0:60625->60625/tcp,:::60625->60625/tcp
redis	docker-entrypoint.sh redis ...	up	0.0.0.0:6379->6379/tcp,:::6379->6379/tcp
zmq-proxy	bluesky-0MQ-proxy 5567 5568	up	0.0.0.0:5567->5567/tcp,:::5567->5567/tcp, 0.0.0.0:5568->5568/tcp,:::5568->5568/tcp
tiled	tiled serve config /deploy/config/config.yml --host 0.0.0.0	up	0.0.0.0:8000->8000/tcp,:::8000->8000/tcp
mongodb	docker-entrypoint.sh mongod	up	0.0.0.0:27017->27017/tcp,:::27017->27017/tcp

Figure 2: Services in Docker Compose.

The `mongodb` service is responsible for data storage and retrieval and is mapped to port 27017 for external access. The `tiled` service serves as a data retrieval system, exposing its services on port 8000. The `redis` service is set up for caching and high-speed data storage and is accessible on port 6379. Within the Docker Compose setup, custom containers are included for `zmq-proxy` and `bluesky-queueserver`. These components are built from Docker files located in local directories. The `zmq-proxy` container listens on ports 5567 and 5568 for communication, while the `bluesky-queueserver` container manages the Bluesky Run Engine [11]. The `bluesky-queueserver` container listens on ports 60615 and 60625 and relies on `redis`, `zmq`, and `tiled`. The `bluesky-httpserver` listens on port 60610 and provides support for the web client by depending on the `bluesky-queueserver`. Finally, the `bluesky-webclient` is a React-based web application accessible on port 3000, and it depends on the `bluesky-httpserver`.

User Interface Design

Layout and Features To ensure consistency with the existing Bluesky Qt GUI used by Emil [3], the Bluesky web client is designed with a similar layout, consisting of seven sections: Status, RE Manager, Console, Plan Editor, Queue List, Plan History, and Plot Data. An overview of the layout is illustrated in Figure 3.

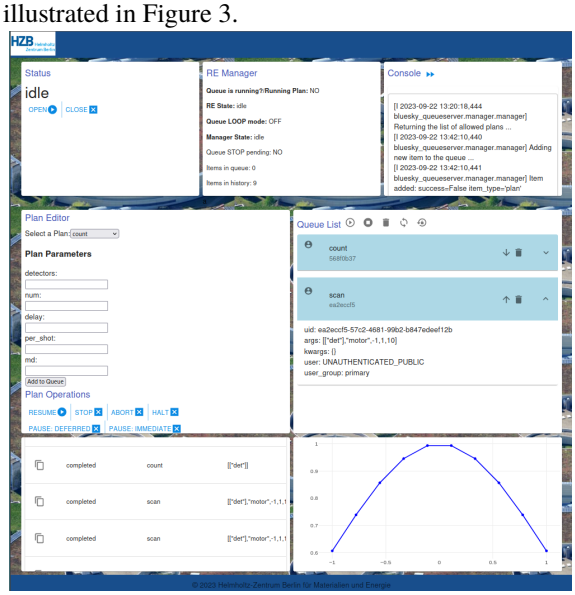

Figure 3: Overview of Bluesky web client.

- **Status**
  This section provides real-time status updates for the Bluesky Run Engine [11]. Users can open or close the Run Engine worker environment.
- **RE Manager** [12]
  Users can access information related to the RE Manager [12], including details such as queue status, RE state, Queue Loop mode, manager state, Queue stop pending, queue items, history items, and so on.
- **Console**
  It provides users with the capability to monitor Bluesky QueueServer activities, track queue operations, log crucial events, and access pertinent messages. Users can expand the console to occupy an entire page by clicking the arrow on the right side, ensuring comprehensive visibility into Bluesky QueueServer events.
- **Plan Editor**
  Users can get all available plans defined in the IPython profile in the Bluesky QueueServer [4]. Upon selecting a plan, its default parameters are presented, and users can give input of the plan. With the 'Add to Queue' button, the plan will be added into the queue list. Operational buttons enable users to resume, stop, abort, halt, or pause the running plans.

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP014

- **Queue List**
  Plans in the queue are listed. Users have control over starting, stopping, and deleting the queue. Additionally, the 'Loop' mode can be toggled on or off. In 'Loop' mode, the queue continuously repeats, while in 'OFF' mode, it runs only once. Plan order can be adjusted by moving plans up or down, and individual plans can be deleted.
- **Plan History**
  After plan execution, the results and process data are shown in the plan history table. This section allows users to access and analyze historical data, including plan arguments, keyword arguments, user information. Plans from the history can be added to the queue list, facilitating the retrieval and reuse of previous executions.
- **Plot Data**
  Currently, data can be visualized by using the UID of a plan from Tiled. Future efforts are focused on achieving real-time data visualization during plan execution.

Software Implementation Inspired by the Bluesky Qt GUI at Bessy II, the web client strives to replicate the layout, enhance and integrate various functionalities in Bluesky QueueServer [4] and Tiled server [6].

React [13] framework is chosen to realize the real-time data handling, server interactions, and dynamic user interfaces for its component-based architecture, cross-platform compatibility, performance optimization, and extensive developer ecosystem. Its ability to target both web and mobile platforms while maintaining code reusability is also advantageous in achieving a cross-device system. TypeScript [14], considered as the programming language for its ability to enhance code quality, maintainability, collaboration, and developer productivity while providing a safety net against type-related errors. In addition, React has official support for TypeScript, and many popular libraries offer TypeScript typings. In order to make the user interface structured and beautiful, Material UI [15] is used. The components from Material UI such as Box, Table, Button, IconButton are initially defined and can be customized. It allows us to focus on expanding the page functions instead of the CSS format for each component. Another consideration to use React and TypeScript is the continuity and scalability with the NSLS II's existing web client [16], which also uses the same structure.

How the web client communicates with the servers is illustrated in Figure 4. When performing a web operation such as adding a plan to the Bluesky QueueServer, React [13] orchestrates the process by sending HTTP requests to the Bluesky QueueServer [4]. The Bluesky QueueServer [4] receives and processes these requests accordingly. Through the Bluesky HTTP Server [7], we retrieve the HTTP responses, which React interprets and displays within the web client. This seamless interaction enables the web client to receive real-time updates and information from the Bluesky QueueServer [4].

Furthermore, during the process of data display, the web client initiates HTTP requests directed towards the Tiled server. These requests are processed by the Tiled [6] server, which formulates the corresponding HTTP responses.

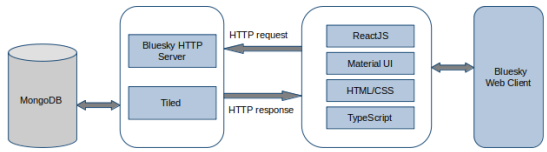

Figure 4: Data interaction between Bluesky web client and back-end applications.

To create the Bluesky web client with React, we can follow the following steps:

1. **Set Up the React Project with TypeScript**
   Using the following command in the terminal:

```
npx create-react-app bluesky-webclient
--template typescript
```

   A React project named "bluesky-webclient" with TypeScript support is created.
2. **Install Additional Dependencies**
   Dependencies can be installed using Yarn [17]. For instance, to add Axios [18], use the following command:

```
yarn add axios
```

3. **Create and Implement Components**
   First, create React components in TypeScript files. Next, import and use these components within the main App component "App.tsx," which serves as the main entry point of the application. Finally, the components will be visible in the web client. Below is an example of the code in "App.tsx":

```
// App.tsx
import React from "react";
import Statusbox from "./Statusbox";
import Remanager from "./Remanager";
import Console from "./Console";
import PlanEditor from "./PlanEditor";
import QueueList from "./QueueList";
import History from "./History";
import Plotdata from "./Tiled";

function App() {
  return (
    <div>
      <Statusbox />
      <Remanager />
      <Console />
      <PlanEditor />
      <QueueList />
      <History />
```

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP014

```
        <Plotdata />
      </div>
    );
}
export default App;
```

4. **Run the Application**
   In the terminal, execute the following command to start the development server and access the Bluesky web client in a web browser at "http://localhost:3000":

   ```
   yarn start
   ```

   When deploying the application in Docker, this step can be omitted and is typically used in the production process.

CONCLUSION

The Bluesky Web Client, built upon the foundation of Bluesky QueueServer [4] and Bluesky HTTP Server [7], presents an adaptable solution for experiment management in the realm of scientific research. While acknowledging the need for diligent attention to challenges like security and hardware compatibility, it remains powerful by its strengths in real-time control, remote accessibility, and streamlined experiment queue management. It fosters collaboration, ignites innovation, and optimizes scientific experimentation processes.

ACKNOWLEDGEMENT

We gratefully acknowledge the support and resources provided by the Bluesky team at NSLS II to make this research and development possible in the first place. We thank our team members and colleagues for their conscientious execution, responsibility, cooperation and selfless help throughout the project.

REFERENCES

[1] Bluesky Project, https://Blueskyproject.io/

[2] Plans, https://nsls-ii.github.io/bluesky/plans.html

[3] W. Smith, S. Kazarski, R. Muller, P. Schnizer, S. Vadilonga, and L. Vera Ramirez, "Status of Bluesky Deployment at BESSY II", in *Proc. ICALEPCS'21*, Shanghai, China, Oct. 2021, pp. 1064–1068. doi:10.18429/JACoW-ICALEPCS2021-FRBR03

[4] Bluesky QueueServer, https://blueskyproject.io/bluesky-queueserver/

[5] Qt GUI, https://doc.qt.io/qt-6/qtgui-overview.html

[6] Tiled Server, https://blueskyproject.io/tiled/

[7] Bluesky HTTP Server, https://blueskyproject.io/bluesky-httpserver/

[8] Mongodb, https://www.mongodb.com/docs/

[9] Docker Compose, https://docs.docker.com/compose/

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

[11] Bluesky Run Engine, https://nsls-ii.github.io/bluesky/tutorial.html##the-runengine

[12] Run Engine Manager, https://blueskyproject.io/bluesky-queueserver/re_manager_api.html

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

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

[15] Material UI, https://mui.com/

[16] Maksim Rakitin, Stuart Campbell, Daniel Allan, Thomas Caswell, Dmitri Gavrilov, Marcus Hanwell and Stuart Wilkins, "Next generation experimental data access at NSLS-II", in *J. Phys. Conf. Ser.*, vol. 2380, 2022, p. 012100. doi:10.1088/1742-6596/2380/1/012100

[17] Yarn, https://yarnpkg.com/

[18] Axios, https://axios-http.com/docs/intro