Πρόγραμμα Μεταπτυχιακών Σπουδών
στις Σύγχρονες Ηλεκτρονικές Τεχνολογίες

Τμήμα Φυσικής
Σχολή Θετικών Επιστημών
Πανεπιστημιο Ιωαννίνων

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ (Μ.Δ.Ε)

Design and Implementation of a Zynq-Based Data Quality Monitoring System Remotely Accessible

---

Κυριάκος Πάσσος
Αριθμός μητρώου: 804

Επιβλέπων καθηγητής: Ιωάννης Παπαδόπουλος

Ιωάννινα
20/06/2024

Abstract

This Master's thesis was conducted at the Department of Physics of the University of Ioannina and under the supervision of Professor Ioannis Papadopoulos. It presents a detailed exploration of data analysis techniques for data coming from a random number generator. The main objective of this research was to develop a program for analyzing data using ROOT, a well known data analysis framework developed at CERN. More specifically this application was designed to construct a histogram from the generated data and apply various statistical fits and understand the underlying data distribution.

This thesis uses an innovative approach to utility backend development with Python, Flask, and PYROOT. This stack was chosen because of its efficiency in data processing and analysis and allowed the application to use web sockets for high-speed communication between the server and the client as well as overall performance.

The front-end project uses React and Plotly along with TypeScript to create client-side applications. This application interactively visualizes the process of data and statistical analysis results. Key features of the client interface include import/export capabilities, real-time data visualization, connection control for exchanging plot configuration data and view fitted data. These features greatly enhance the user interface and experience.

In this thesis, the emphasis is placed on the practical application and utilization of the developed system in analyzing data from a random number generator. The narrative delves into the operational aspects of the system, detailing how the Python-Flask-PYROOT backend synergizes with the React-Plotly frontend to facilitate seamless data processing and visualization.

Special attention is given to the system's capability to handle large data sets and perform complex statistical analyses with efficiency and accuracy. The document discusses how the application, through its intuitive user interface, empowers users to interact with the data, manipulate visualization parameters, and derive meaningful insights from the analysis.

Furthermore, the thesis explores the real-world applicability of the system, considering various scenarios where such a tool can be used. It accounts for potential extensions and adaptations of the system, envisioning its use in broader contexts beyond the scope of the current project. This foresight underlines the system's versatility and potential for future enhancements and applications in diverse data-driven fields.

This thesis does not just present a technical solution but also contextualizes it within the larger framework of data analysis and application, underscoring the significance and practicality of the developed system.

Περίληψη

Η παρούσα μεταπτυχιακή διατριβή πραγματοποιήθηκε στο Τμήμα Φυσικής του Πανεπιστημίου Ιωαννίνων υπό την επίβλεψη του Καθηγητή Ιωάννη Παπαδόπουλου. Αντικείμενο της έρευνας ήταν η διερεύνηση τεχνικών ανάλυσης δεδομένων που προέρχονται από μια γεννήτρια τυχαίων αριθμών, με κύριο στόχο την ανάπτυξη ενός προγράμματος για την ανάλυση αυτών των δεδομένων μέσω του ROOT, μία γνωστή βιβλιοθήκη ανάλυσης δεδομένων που αναπτύχθηκε στο CERN.

Η εφαρμογή που αναπτύχθηκε στο πλαίσιο της διατριβής, έχει ως κύριο στόχο τη δημιουργία ιστογραμμάτων από τα δεδομένα που παράγονται και την εφαρμογή διαφόρων στατιστικών προσαρμογών για την κατανόηση της υποκείμενης κατανομής των δεδομένων. Για την υλοποίηση αυτής της εφαρμογής επιλέχθηκε η χρήση της γλώσσας προγραμματισμού Python σε συνδυασμό με τα Flask και PYROOT, λόγω της αποδοτικότητάς τους στην επεξεργασία και ανάλυση δεδομένων. Αυτές οι τεχνολογίες επιτρέπουν επίσης την χρήση web sockets για την ταχεία επικοινωνία μεταξύ του server και του client, βελτιώνοντας συνολικά την απόδοση του συστήματος.

Το frontend του έργου σχεδιάστηκε με χρήση των React και Plotly μαζί με TypeScript, για τη δημιουργία εφαρμογών που λειτουργούν στην πλευρά του client. Αυτή η εφαρμογή επιτρέπει τη διαδραστική οπτικοποίηση της διαδικασίας ανάλυσης δεδομένων και των αποτελεσμάτων των στατιστικών αναλύσεων. Βασικά χαρακτηριστικά των δυνατοτήτων του client περιλαμβάνουν δυνατότητες εισαγωγής και εξαγωγής δεδομένων, οπτικοποίηση δεδομένων σε πραγματικό χρόνο, έλεγχο της σύνδεσης για την ανταλλαγή δεδομένων διαμόρφωσης γραφημάτων και προβολή των προσαρμοσμένων δεδομένων.

Ιδιαίτερη έμφαση δίνεται στην πρακτική εφαρμογή του ανεπτυγμένου συστήματος στην ανάλυση δεδομένων από τη γεννήτρια τυχαίων αριθμών. Το κείμενο εμβαθύνει στις λειτουργικές πτυχές του συστήματος, περιγράφοντας πώς το backend Python-Flask-PYROOT συνεργάζεται με το frontend React-Plotly για την απρόσκοπτη επεξεργασία και οπτικοποίηση των δεδομένων.

Επιπλέον, η διατριβή εξετάζει την ικανότητα του συστήματος να διαχειρίζεται μεγάλα σύνολα δεδομένων και να εκτελεί σύνθετες στατιστικές αναλύσεις με αποδοτικότητα και ακρίβεια. Επίσης διερευνώνται οι πιθανές επεκτάσεις και προσαρμογές του συστήματος, προβλέποντας τη χρήση του σε ευρύτερα πλαίσια πέρα από το πεδίο της τρέχουσας έρευνας.

Η διατριβή δεν περιορίζεται στην παρουσίαση μιας τεχνικής λύσης αλλά την τοποθετεί στο ευρύτερο πλαίσιο της ανάλυσης δεδομένων και των εφαρμογών της, υπογραμμίζοντας τη σημασία και την πρακτικότητα του ανεπτυγμένου συστήματος. Μέσα από την αναλυτική παρουσίαση της τεχνολογικής υποδομής και της λειτουργικής αλληλεπίδρασης των διαφόρων στοιχείων, αποδεικνύεται η αξία της συνεισφοράς αυτής της έρευνας στη βελτίωση των διαδικασιών ανάλυσης δεδομένων και στην ενίσχυση της δυνατότητας για προσαρμοσμένες λύσεις στον τομέα της στατιστικής ανάλυσης.

Η παρούσα διατριβή αποτελεί μια σημαντική συνεισφορά στον τομέα της ανάλυσης δεδομένων, προσφέροντας τόσο μια καινοτόμο τεχνολογική λύση όσο και μια ευρύτερη θεώρηση των δυνατοτήτων και των μελλοντικών εφαρμογών της.

Contents

1 Introduction

1.1 Project Background and Evolution

The initial concept for this project involved utilizing a Zynq+OS base to handle data acquisition and processing. The Zynq platform, with its combination of FPGA and ARM processor, was selected to implement a Random Number Generator (RNG) on the FPGA, from the Master Thesis of Th. Fotos [1]. The data from the RNG would be processed by a server running on the ARM processor, which would perform statistical analysis and communicate with the client application.

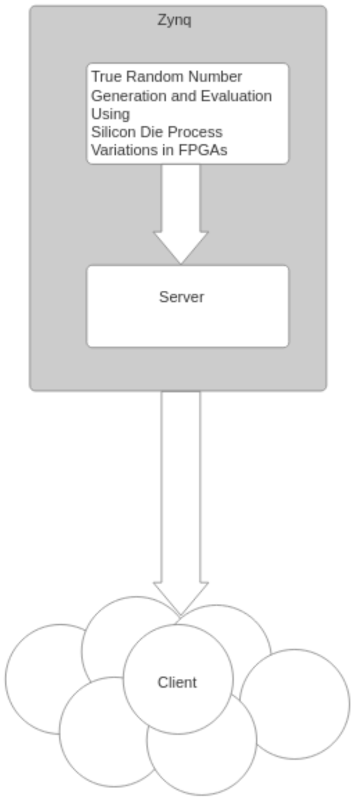

Figure 1: Initial Plan.

However, during the development process, it became clear that implementing the hardware solution within the available time frame would rather be the subject of another master thesis, as the overall work load became excessive.

Thus, we pivoted to use a virtual data generator on the server. This allowed us to simulate the RNG data and focus on developing the server-side statistical analysis and client communication. This approach ensured that the project's primary objectives were met.

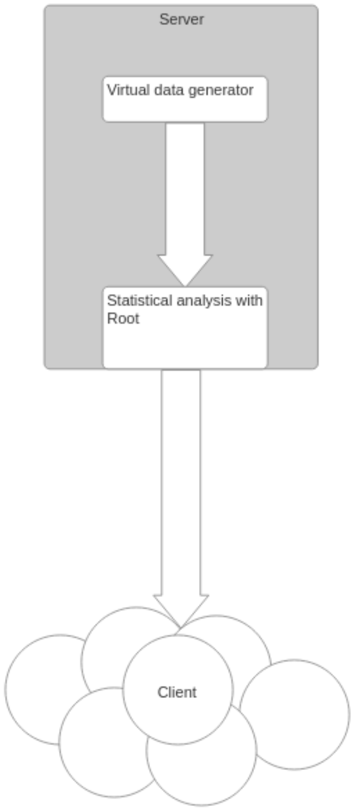

Figure 2: Final Implementation.

1.2 Overview of Data Quality Monitoring Systems

The Evolution of Data Quality Monitoring
Data quality monitoring has undergone significant transformation over the years. Initially, in the era of early computing, the focus was primarily on data collection and storage, with little emphasis on the quality aspect. However, as data's role in decision-making processes became more pronounced, the importance of its quality surged.
In the 1980s and 1990s, with the advent of databases and more sophisticated data storage technologies, organizations began to realize the repercussions of poor data quality. This era saw the development of the first data cleaning and scrubbing tools, aimed at rectifying inaccuracies and inconsistencies in data [2].
The turn of the millennium marked a new era in data quality monitoring, fueled by the explosion of the internet and the advent of big data. This period witnessed the rise of advanced data quality management tools capable of handling vast arrays of complex data. These tools integrated functionalities like data profiling, cleansing, matching, and enrichment, significantly enhancing the data quality process [3].
The Pillars of Data Quality Data quality is founded on several key dimensions:

- **Accuracy:** The degree to which data correctly describes the real-world attributes it is intended to represent [4].

- **Completeness:** Refers to the extent to which all required data is available [5].
- **Consistency:** The uniformity of data across different datasets and systems [6].
- **Timeliness:** The relevance of data at the time of its use [7].
- **Reliability:** The degree to which data accurately and consistently represents information over time [8].

Each of these dimensions plays a crucial role in determining the overall quality of data. In contemporary settings, ensuring these aspects of data quality has become a complex task due to the sheer volume and variety of data handled by organizations.

Brief History and Development of Data Quality Monitoring Systems The consequences of poor data quality are far-reaching and can include:

- **Erroneous Decision Making:** Decisions based on inaccurate or incomplete data can lead to significant financial losses and strategic missteps [9].
- **Operational Inefficiencies:** Poor data can result in wasted resources, increased costs, and lost opportunities [10].
- **Regulatory Compliance Risks:** In many industries, especially finance and healthcare, regulatory compliance hinges on the integrity of data. Non-compliance due to poor data quality can lead to legal penalties and loss of reputation [11].

Data Quality in the Age of Big Data The era of big data and advanced analytics has brought new challenges and opportunities to the field of data quality monitoring. The volume, velocity, and variety of data have made traditional data quality methods inadequate. This has spurred the development of more sophisticated techniques, including the use of machine learning and artificial intelligence, to automate and enhance data quality processes [12].

Future Trends in Data Quality Monitoring Looking forward, the field of data quality monitoring is poised for further evolution. Key trends include:

- **Integration of Artificial Intelligence:** AI and machine learning are increasingly being leveraged for predictive data quality, anomaly detection, and automated error correction [13].
- **Focus on Data Governance:** Organizations are recognizing the importance of data governance in ensuring data quality. This includes policies, procedures, and roles related to data management [14].
- **Real-Time Data Quality Monitoring:** With the increasing need for real-time analytics, real-time data quality monitoring is becoming essential [15].
- **Data Quality as a Service (DQaaS):** The rise of cloud computing has led to the emergence of DQaaS, allowing organizations to leverage external expertise and cutting-edge technologies [16].

1.3 Role and Applications of Random Number Generators in Scientific Research and Simulations

Fundamental Role of Random Number Generators

Random Number Generators (RNGs) are pivotal in scientific research, providing a foundation for simulations and analyses where unpredictability is essential. RNGs generate sequences of numbers devoid of any discernible pattern, imitating the concept of randomness found in various natural processes. In computational science, these generators facilitate the creation of algorithms and models that emulate the stochastic nature of these processes, making them indispensable tools in numerous scientific domains [17].

RNGs in Scientific Simulations

One of the primary applications of RNGs is in the field of scientific simulations, particularly Monte Carlo simulations. These simulations utilize RNGs to model complex systems and phenomena, ranging from subatomic particle interactions in physics to ecological systems in biology. The accuracy and effectiveness of these simulations heavily rely on the quality of randomness produced by RNGs, as they often form the basis for critical scientific inferences and decisions [18].

RNGs in Cryptography and Data Security

In addition to scientific simulations, RNGs have a significant role in the field of cryptography. They are used to generate cryptographic keys, which form the backbone of data security and encryption protocols. The strength and security of cryptographic systems depend heavily on the unpredictability of these keys, highlighting the importance of high-quality RNGs in protecting sensitive information in the digital age [19].

Challenges in Utilizing RNGs

Despite their wide applications, RNGs come with their own set of challenges. One major concern is ensuring the true randomness of the numbers they generate. Many RNG algorithms are deterministic in nature, meaning they could potentially be predicted or replicated under certain conditions. This predictability poses a significant issue in fields where randomness is crucial for the integrity of the results, such as in cryptography and high-stakes simulations [20].

Future Perspectives and Advancements

Looking forward, advancements in RNG technology are likely to focus on enhancing the unpredictability and efficiency of these generators. Developments in quantum computing, for instance, offer promising avenues for generating true randomness, surpassing the limitations of traditional, algorithm-based RNGs. As research in this area progresses, the applications of RNGs are expected to expand, further cementing their role in scientific research and technological development [21].

Random Number Generators play a crucial role in the realm of scientific research and simulations. Their ability to emulate randomness is essential in a wide range of applications, from complex system modeling to data security. As technology advances, the development and refinement of RNGs will continue to be a key area of focus, driving innovation and discovery in various scientific and technological fields [22].

1.4 Introduction to ROOT: The CERN Data Analysis Framework

Historical Background of ROOT

The ROOT system, developed at CERN, stands as a testament to the evolution of data analysis tools in scientific research. Developed in the early 1990s by Dr. Rene Brun and Dr. Fons Rademakers, ROOT was conceived to meet the demanding data analysis needs of high-energy physics experiments, particularly those at the Large Hadron Collider. It represented a paradigm shift, introducing an object-oriented approach using C++, which allowed for more sophisticated and scalable data analysis capabilities than previously possible [23].

Key Features and Capabilities of ROOT

ROOT offers a multitude of features that make it a versatile and powerful tool for data analysis:

- **Data Processing and Statistical Analysis:** ROOT excels at handling and processing large datasets, equipped with an array of statistical functions for comprehensive data analysis [24].
- **Visualization Tools:** Its advanced visualization capabilities allow for the effective representation of data, crucial for both analysis and presentation purposes [23].
- **Object-Oriented Storage:** ROOT uses a unique file format for data storage, designed for high performance with complex data structures [23].
- **Extensibility and Scalability:** The framework's architecture allows for the development of custom extensions, making it adaptable to various data analysis requirements [24].

ROOT in Modern Scientific Data Analysis

ROOT's impact extends far beyond high-energy physics. Its ability to manage, process, and analyze large volumes of data efficiently makes it a valuable tool in fields such as astrophysics, bioinformatics, and environmental science. The framework has significantly contributed to the advancement of research methodologies, enabling scientists to handle increasingly complex data sets and extract meaningful insights [24].

Challenges and Future Prospects

While ROOT is highly effective, it also poses challenges, particularly in terms of its learning curve and complexity. Future developments are expected to focus on enhancing user accessibility and integrating emerging technologies like machine learning and artificial intelligence to broaden its application spectrum [24].

ROOT has revolutionized data analysis in scientific research, offering a robust, flexible, and comprehensive solution. Its ongoing evolution and adaptation continue to make it an invaluable asset in the scientific community's pursuit of knowledge and discovery.

1.5 Python, Flask, and PYROOT: Integrating Technologies for Data Analysis

Python in Backend Development: Python's emergence as a leading programming language in backend development is attributed to its simplicity, readability, and versatility. With its extensive standard library and support for multiple programming paradigms, Python provides a solid foundation for building a wide range of applications. In data analysis, Python's ability to handle large datasets, perform complex calculations, and integrate with various data sources makes it an ideal choice [25].

Python's ecosystem is rich with frameworks and libraries specifically tailored for backend development, data processing, and scientific computing. Libraries like NumPy and Pandas have become staples in data manipulation and analysis, while frameworks such as Django and Flask offer robust tools for web application development [26].

Synergy with ROOT through PYROOT: The integration of Python with ROOT is achieved through PYROOT, a Python extension that provides access to all the functionalities of the ROOT framework. This integration allows for leveraging ROOT's powerful data processing and visualization capabilities within the Python environment. PYROOT thus becomes a bridge, combining Python's ease of use and flexibility with ROOT's advanced computational functionalities [24].

This synergy is particularly beneficial in scientific research where complex data analysis is required. PYROOT enables researchers to script ROOT-based applications in Python, making it more accessible to those who are more familiar with Python than C++ (the language ROOT is primarily written in) [23].

Advantages of Python and Flask: The combination of Python with Flask offers a lightweight yet powerful solution for developing web-based applications. Flask's minimalistic and modular design allows developers to build applications rapidly, with the flexibility to choose the tools and components best suited for their project. This aspect is crucial when dealing with data-intensive applications where custom solutions are often needed [27].

Furthermore, Python and Flask benefit from a vast and active community. This community provides extensive support, contributing to a wealth of documentation, tutorials, and third-party libraries. For developers, this means better resources for problem-solving and keeping up-to-date with the latest advancements in technology [27].

The integration of Python, Flask, and PYROOT offers a powerful combination for data analysis and web application development. This trio caters to the growing demand for flexible, scalable, and efficient tools in the realm of scientific computing and data analysis. Their widespread adoption and community support further underline their importance in the contemporary landscape of technology and research.

1.6 Client-Side Development: React, TypeScript, and Plotly

React and TypeScript in Frontend Development: React, a JavaScript library for building user interfaces, stands out for its component-based architecture, making it a popular choice for developing complex and interactive web applications. React's ability to handle dynamic content updates smoothly, thanks to its virtual DOM system, enhances user experience by providing efficient, seamless interactions [28]. TypeScript, a superset of JavaScript, complements React by offering type safety, which helps in managing large codebases and reduces the likelihood of runtime errors [29].

The combination of React and TypeScript has become increasingly prevalent in modern web development. TypeScript's static typing adds an extra layer of structure and robustness to React applications, making the code more maintainable and easier to scale. This pairing is especially beneficial in large-scale projects where stability, scalability, and developer collaboration are key [29].

The Role of Plotly in Data Visualization: Plotly, a graphing library, plays a crucial role in data visualization within web applications. Its compatibility with a variety of programming languages, including Python, makes it a versatile choice for displaying complex data visualizations. In the context of React applications, Plotly can be integrated to render interactive and visually appealing charts and graphs. These visualizations are not only dynamic but also responsive,

adapting seamlessly to different screen sizes and user interactions [30].
Plotly's interactivity is a significant advantage, as it allows users to engage with the data in more meaningful ways. Features such as zooming, panning, and tooltips enhance the user's ability to explore and understand complex datasets. This level of interactivity is particularly valuable in applications where data insights and user engagement are pivotal [30].
Advantages of the React-TypeScript-Plotly Stack: The combination of React, TypeScript, and Plotly offers a powerful tech stack for frontend development. React's efficient rendering and TypeScript's type safety form a solid foundation for building robust applications, while Plotly's advanced visualization capabilities allow for the creation of interactive data-driven interfaces [28].
This stack is particularly advantageous for projects requiring complex data handling and visualization, such as dashboards, data analytics platforms, and scientific data exploration tools. The synergy of these technologies enables developers to build applications that are not only functionally rich but also user-friendly and aesthetically pleasing [29].
The integration of React, TypeScript, and Plotly in client-side development represents a significant stride in the field of web application development. This combination brings together efficient UI building, type-safe coding practices, and advanced data visualization, culminating in a development experience that is both developer-friendly and user-centric. As web technologies continue to evolve, the adoption and adaptation of such tech stacks will play a crucial role in shaping the future of interactive web applications.

1.7 Importance and Applications of Data Analysis

Broad Scope and Impact Across Sectors: Data analysis has become a fundamental aspect of numerous sectors, driving decision-making and strategic planning. In business, data analysis helps in identifying market trends, understanding customer behavior, and optimizing operational efficiency. In healthcare, it aids in patient data management, treatment personalization, and medical research advancements. Similarly, in finance, data analysis is crucial for risk assessment, fraud detection, and investment strategies. The environmental sector relies on data analysis for climate modeling and conservation efforts. This widespread application underscores the versatility and critical role of data analysis in modern society [31].
Case Studies and Breakthroughs: Several case studies highlight the transformative power of data analysis. For instance, in healthcare, the use of big data and analytics in genomics has led to groundbreaking discoveries in personalized medicine. In finance, sophisticated data models have drastically improved risk management systems. Another notable example is the use of data analysis in optimizing supply chain and logistics in the retail sector, leading to enhanced efficiency and customer satisfaction. These cases demonstrate how data-driven approaches can lead to significant improvements and innovations [32].
Future Trends in Data Analysis: The future of data analysis is poised for exciting advancements. The integration of artificial intelligence and machine learning is expected to automate and enhance analytical processes. Big data analytics will continue to grow, handling increasingly larger and more complex datasets. The emergence of edge computing promises faster, real-time analytics. Additionally, the democratization of data analysis through user-friendly tools will enable more individuals and organizations to harness the power of data. These trends indicate a trajectory towards more sophisticated, efficient, and accessible data analysis methodologies [33].

The importance and applications of data analysis are vast and diverse, permeating various aspects of modern life. Its role in driving innovation, efficiency, and decision-making is undeniable. As technology evolves, so will the methods and applications of data analysis, continuing to shape industries and society at large [31].

1.8 Staying Ahead with New Technologies: The Need for Continuous Innovation

Adopting New Technologies in Data Analysis: The landscape of data analysis is continually evolving, driven by the rapid development of new technologies. Adopting these technologies is not just beneficial but essential for staying competitive in various fields. Innovations such as artificial intelligence, machine learning, and cloud computing are revolutionizing the way data is analyzed and interpreted. They enable more sophisticated, faster, and more accurate analysis, providing deeper insights and enhancing decision-making processes. In sectors ranging from healthcare to finance, the ability to leverage these advanced tools can be a determining factor in success [34].
Benefits of Technological Advancements: Staying current with technological advancements can significantly enhance the efficiency and effectiveness of data analysis. Advanced algorithms can process and analyze vast amounts of data more quickly than traditional methods, leading to quicker insights. Machine learning models are capable of uncovering patterns and correlations that might be missed by human analysts, leading to more comprehensive and accurate results [35]. Furthermore, these advancements often come with improved user interfaces, making complex data analysis more accessible to a broader range of users [36].
Balancing Innovation with Reliability: While embracing new technologies is crucial, it's equally important to balance innovation with reliability. Established methods and technologies have a proven track record and are often well-understood in terms of their capabilities and limitations. New technologies, while promising, can be untested in certain applications and may have unforeseen challenges. Therefore, a balanced approach that combines the reliability of established methods with the innovative potential of new technologies can often yield the best results. This approach ensures a foundation of trustworthiness while still pushing the boundaries of what's possible in data analysis [37].
The integration of new technologies into data analysis is a necessity in the modern era. It enables more efficient, accurate, and comprehensive analysis, which is vital across numerous domains. However, this pursuit of innovation must be tempered with a consideration for reliability and established practices. Striking this balance is key to advancing the field of data analysis while maintaining the integrity and dependability of its outcomes [34].

1.9 Global Accessibility: The Need for Remotely Accessible Applications

Growing Demand for Remote Accessibility: In today's interconnected world, the demand for remotely accessible software applications is surging. This trend is driven by the global nature of business, the increasing prevalence of remote work, and the need for real-time collaboration across geographical boundaries. Remote accessibility is no longer a luxury but a necessity, enabling users to access vital applications and data from anywhere, at any time. This shift has implications for a wide range of sectors, from education and healthcare to business and government [38].

Benefits of Remote Access: The benefits of remote access are manifold. Firstly, it facilitates collaboration by allowing team members to work together seamlessly, regardless of their physical location. This ability is crucial in today's globalized work environment, where teams are often spread across different regions. Secondly, remote accessibility offers unmatched flexibility, enabling users to stay productive and connected even when they are away from the traditional office setting. This flexibility is essential for adapting to the rapidly changing work environments and schedules. Lastly, remote access broadens the impact of applications, making them accessible to a larger and more diverse user base, which is particularly important in fields like education and healthcare, where broad accessibility can have profound social impacts [39].
Examples of Successful Remote Applications: Numerous applications across various domains have successfully harnessed the power of remote accessibility. In the field of education, platforms like Moodle and Google Classroom allow students and teachers to interact and access educational materials from anywhere [40]. In healthcare, telemedicine apps have revolutionized patient care by enabling remote consultations and monitoring [41]. In the business world, tools like Salesforce and Slack facilitate remote work and collaboration, enabling businesses to operate efficiently regardless of their employees' locations [42].
The shift towards globally accessible, remotely operated applications is a defining trend in modern software development. This shift is not just about technological advancement; it's about adapting to the changing ways people live and work. As such, the development of remotely accessible applications will continue to be a key focus for software developers and companies, striving to meet the growing needs of a globally connected society [38].

1.10 Summary

This introduction has laid the groundwork for the exploration of the integration and application of advanced technologies in data analysis and software development. We have delved into the significance of data quality monitoring systems, the critical role of random number generators in scientific research, and the synergy between Python, Flask, and PYROOT in backend development. The discussion extended to the client-side development using React, TypeScript, and Plotly, highlighting their importance in creating dynamic user interfaces. The necessity of staying updated with technological advancements and the increasing demand for globally accessible applications were also emphasized.
As we move into the main body of this thesis, the subsequent chapters will focus on a detailed analysis of each of these components. We will examine specific case studies, practical applications, and the technical challenges encountered in the integration of these technologies. These discussions aim to provide a comprehensive understanding of the current state and potential future developments in this rapidly evolving field.
The objective of this thesis is to contribute to the understanding of how modern software and data analysis technologies can be effectively integrated to create powerful, efficient, and user-friendly applications. It is anticipated that this work will not only add to the academic discourse but also offer practical insights that can be applied in various sectors where data analysis and software development are pivotal.

2 Technology Overview

2.1 Python

Python, created by Guido van Rossum and first released in 1991, has evolved significantly over the past three decades. Initially conceived as a successor to the ABC language, Python was designed to be enjoyable to use and easily readable, yet powerful enough for complex software development. Its philosophy emphasized code readability and simplicity, which resonated with many programmers [25].
The early versions of Python already included exceptional features like exception handling and functions. The introduction of Python 2.0 in 2000 was a major milestone, bringing in Unicode support and a full garbage collector. However, it was the release of Python 3.0 in 2008 that marked a significant turning point. This version was a major overhaul, not backward compatible with Python 2, but it set the stage for the language's future, emphasizing consistency and clarity [43].
Today, Python's popularity has soared, in part due to its versatility and the vast ecosystem of libraries and frameworks it supports. It's widely used in web development, with frameworks like Django and Flask, in scientific computing and data analysis with tools like NumPy, SciPy, and Pandas, and in artificial intelligence and machine learning through libraries such as TensorFlow and PyTorch [26]. Python's simplicity makes it accessible to beginners, while its power and flexibility make it a favorite among experienced developers for complex applications.
From its modest beginnings, Python has grown to become one of the most widely used programming languages in the world, valued in both academic and professional realms for its efficiency and readability [25].

2.2 JavaScript and TypeScript

JavaScript, initially created by Brendan Eich in 1995 and named Mocha, was developed to add interactive elements to websites in the nascent days of the web. Renamed to JavaScript, this scripting language quickly gained popularity due to its ability to create dynamic content in web browsers. It became a core technology of the World Wide Web [44].
Despite its widespread use, JavaScript's lack of strong typing and the challenges of managing large-scale applications led to the development of TypeScript. Introduced by Microsoft in 2012, TypeScript is a superset of JavaScript that adds static typing. This feature allows developers to catch errors at compile time, rather than at runtime, which is crucial for larger, more complex projects [29].
Today, JavaScript, bolstered by numerous frameworks and libraries like React, Angular, and Vue.js, is a cornerstone of modern web development. It powers interactive and dynamic web applications, ranging from single-page applications to complete web solutions. TypeScript, with its enhanced features, has been adopted by many organizations for large-scale application development due to its ability to improve code quality and maintainability [29].
From its early conception as a simple scripting language, JavaScript has evolved into a powerful tool for front-end web development. With TypeScript, it now also meets the needs of enterprise-level applications, showcasing its versatility and continued relevance in the ever-evolving landscape of web technologies [44].

2.3 React

React, also known as React.js or ReactJS, is a JavaScript library for building user interfaces, particularly for single-page applications. It was developed by Jordan Walke, a software engineer at Facebook, and was first deployed on Facebook's newsfeed in 2011 and later on Instagram in 2012. React was open-sourced at the JSConf US in May 2013 [28].
The primary objective behind React's creation was to address the challenge of building large-scale applications with data that changes over time. Its key innovation lies in the introduction of the virtual DOM, which allows for efficient updates and rendering of UI components. React's component-based architecture has made it easier to develop and maintain large web applications, as it promotes reusable UI components [28].
Today, React is one of the most popular front-end libraries in the world. It is widely used in the development of complex, interactive web applications due to its efficiency and flexibility. Major companies such as Facebook, Instagram, and Netflix use React in their production environments, testifying to its scalability and robustness [28].
React's ecosystem has grown significantly, with the addition of tools like React Native for mobile app development, and the introduction of hooks in React 16.8, further enhancing its capabilities and developer experience. The library continues to evolve and adapt, playing a crucial role in modern web development practices [28].

2.4 WebSockets

WebSockets provide a full-duplex communication channel over a single, long-lived connection, allowing servers and clients to exchange data freely and efficiently. This technology is essential for applications requiring real-time data transfer, such as live chat applications, real-time notifications, and interactive games [45].
Unlike the traditional request-response model used in HTTP, WebSockets enable a two-way interaction between the client and server. This interaction begins with a handshake, facilitated by an HTTP upgrade request from the client, which requests the server to open a WebSocket connection [45].
The WebSocket handshake is a crucial step in establishing a WebSocket connection. It upgrades the HTTP protocol to the WebSocket protocol, using the HTTP request upgrade header. This process ensures compatibility with the existing web infrastructure [45].
Once the handshake is successful, the WebSocket connection is established, and data can be sent back and forth between the client and server without the need to reopen the connection. This persistent connection reduces latency and overhead, providing a seamless, real-time user experience [45].
WebSockets are widely used in applications that require live, real-time communication, including:

- Live chat applications
- Real-time financial trading platforms
- Multiplayer online games
- Live sports updates and streaming

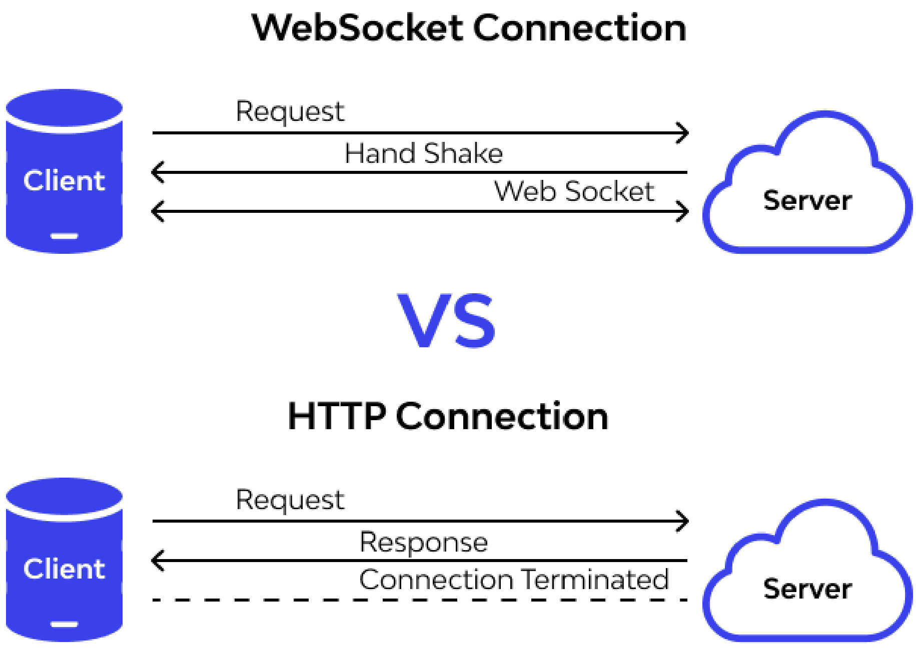

Figure 3: Websocket vs HTTP connection.

The primary advantages of using WebSockets include:

- **Reduced Latency:** Immediate data transfer without the need to establish new connections for each message.
- **Lower Overhead:** After the initial handshake, data frames can be sent with minimal overhead.
- **Full Duplex Communication:** Simultaneous data flow in both directions.

3 Implementation Details

The primary objective of this thesis is to design and implement a web-based application that facilitates the analysis of datasets generated by a random number generator. This analysis is conducted utilizing the ROOT library. Upon analysis, the processed data are transmitted to the client-side application, which is responsible for rendering the results in a visually compelling and interactive manner. This feature significantly augments the user interface and enriches the user experience by providing intuitive data visualization tools and functionalities. Moreover, this thesis lays the foundation for future scalability and extensibility. The server-side architecture is developed with an object-oriented approach, allowing for seamless integration of additional modules and functionalities for future development. Similarly, the client-side application, constructed using React, offers a flexible and modular framework allowing for further expansion and customization. This design pattern ensures that the application not only meets the current analytical needs but also accommodates future advancements and extensions.

The implementation strategy for this concept can be broken down to several key segments:

1. **Data Analysis with ROOT on the server side:** Utilize the ROOT library for detailed statistical analysis and interpretation of the data.
2. **Server-Client Communication:** Establish a robust and efficient mechanism for data exchange between the server and the client, employing web sockets technology.
3. **Data visualization and enhancement of user interface on the client side:** Implementation of advanced visualization techniques to represent the analyzed data effectively and develop and integrate features that improve navigation, interaction, and accessibility of the web application on the client side.

This thesis aims to contribute to the field of data analysis by showcasing the potential of integrating cutting-edge computational tools and web technologies to enhance data interpretability and user engagement.

3.1 Server

3.1.1 Server Implementation and Flask Integration

The server-side framework of our application is devised through an intricate synergy of Python, PyRoot, and Flask. This combination is meticulously tailored for the efficient handling and analysis of data. At the heart of our server's logic is a single-threaded strategy dedicated to the generation of random numbers, which could of course be implemented on hardware[1], their subsequent distribution analysis, histogram construction, and the fitting of these histograms to various statistical distributions, namely Gaussian, Landau, Breit Wigner, and Double Gaussian. The results of these computations are then seamlessly transmitted to the client-side, facilitating sophisticated data visualization.

3.1.2 Technical Architecture

Our server's architecture is predicated on a carefully structured class design, each component of which serves a pivotal role within the broader ecosystem of the application. This object-oriented approach not only enhances the maintainability and re-usability of the code but also lays a solid groundwork for future scalability and enhancements.

Integration of Flask in the Application Flask, a lightweight Web Server Gateway Interface (WSGI) web application framework, is employed as a critical component in our server-side architecture. It offers the flexibility needed to create and manage web applications. In our project, Flask serves several vital functions:

- Initializing the web server instance, thereby acting as the gateway for client-server communication.
- Facilitating Cross-Origin Resource Sharing (CORS) through its extensions, allowing our application to share resources across different origins securely.
- Managing WebSocket connections via Flask-SocketIO, which enables real-time bidirectional communication between the client and the server. This capability is crucial for dynamically updating the client-side with new data without the need for manual refreshing.

The integration of Flask significantly contributes to the robustness of our application, ensuring efficient data flow and enhancing user interaction through real-time data visualization.

3.1.3 Optimization with Numpy

Numerical computations, especially those involving complex array operations, are optimized using the Numpy library. Owing to its C foundation, Numpy ensures that our server-side logic is executed with efficiency and accuracy. This particular selection underscores our dedication to utilizing premier tools for data processing, thereby establishing a strong foundation for a scalable server-side architecture.

3.1.4 Class Design

The backbone of our server-side application is constituted by the following classes:

1. **Histogram Class:** Inherits from ROOT's `TH1`, the fundamental histogram class in ROOT. This class extends TH1 to encapsulate the histogram's properties and functionality, requiring five parameters upon instantiation:
   - Name: A unique identifier for the histogram.
   - Title: Descriptive title of the histogram.
   - Number of bins: Resolution of the histogram.
   - Low limit for x-axis: Defines the lower boundary of the histogram's domain.
   - Upper limit for x-axis: Defines the upper boundary of the histogram's domain.
2. **Distribution Analysis Class:** Analyzes the random data to ascertain its distribution, utilizing statistical methods and algorithms to classify the underlying distribution pattern accurately. Requires a single parameter upon instantiation:

- Histogram: A histogram object instantiated by the Histogram class.

3. **Fit Class:** Dedicated to fitting the histogram with predefined distribution models. This class serves as the statistical processor of our application and requires two parameters upon instantiation:
    - Histogram: A histogram object instantiated by the Histogram class.
    - Distribution: The distribution type derived from the Distribution Analysis Class.

3.1.5 Core Functionalities

The Histogram class, pivotal to our data analysis process, extending ROOT's `TH1` capabilities, supplemented with custom methods tailored to our application's requirements:

- **Fill Method:** Adds new data points into the histogram, using `TH1F.Fill()` for data insertion, thereby enriching the dataset with every iteration.
- **GetYMax Method:** Extracts the peak value of the y-axis, using `TH1F.GetMaximum()`, for data normalization and scaling.
- **GetHistogramData Method:** Retrieves the histogram's data in a structured format for further computational analysis or visualization.

The **Distribution Analysis Class** is a cornerstone in our analytical process, equipped with a pivotal method:

- **Find Distribution Method:** This method employs a rigorous statistical approach to fit the histogram data across various predefined distribution functions, such as Gaussian, Landau, and others. It compares the goodness-of-fit statistics for each distribution, aiming to identify the most accurate model that represents the data. The evaluation is based on the chi-squared over the degrees of freedom (χ^2/ndf) ratio, with the optimal fit being the one that closely approximates a value of one. This method systematically returns both the best-fitting distribution function and a comprehensive array of goodness-of-fit statistics, providing a detailed analysis of the data underlying distribution.

The **Fit Histogram Class** plays a critical role in our application, distinguished by its unique operational paradigm. Unlike typical classes, it is designed without a constructor, opting instead for an approach that generates single instances of histogram objects. These instances are strategically designed for reuse within the application's life cycle, significantly optimizing computational efficiency and memory usage. This design choice mitigates the need for repetitive instantiation for each analysis, particularly beneficial for recurring fits. Key methods within this class include:

- **Get Fit Parameters:** It extracts the fit parameters from the histogram, encompassing statistical measures such as amplitude, mean, sigma, and their respective errors, alongside the number of degrees of freedom (NDF) and chi-square values. The output is organized into a tuple, providing a structured overview of the fit's characteristics.

- **GetYFit:** This method is generating the y-axis values of the fitted curve. By evaluating the fit function at each bin's x-axis value, it constructs a list of y-axis values, offering a detailed representation of the fitted curve across the histogram.
- **Get Double Fit Parameters:** For more complex analyses, this method retrieves the parameters associated with a double Gaussian fit. It gets the parameters for both components of the Gaussian fit, presenting them in a structured tuple. This dual-parameter extraction is crucial for analyses that require an understanding of the data's distribution characteristics.

These classes and their methods underscore the analytical capabilities of our application, demonstrating a robust framework for data analysis that not only ensures precision and efficiency but also extensibility and scalability in handling complex data distributions.

The operational lifecycle of our threading mechanism is designed to ensure efficient and continuous data generation and analysis. The process unfolds as follows:

Initially, a histogram object is instantiated externally to the thread. Subsequently, the thread embarks on a repeated cycle of generating random data across any given distribution, amassing up to ten thousand data points per second.

Upon generation, the data are promptly relayed to the *Distribution Analysis* class, which deduces the most probable distribution. In instances where a double Gaussian distribution is identified, a tolerance threshold of 10% is applied to accommodate potential anomalies, such as spikes resulting from noise, thereby allowing it to be classified as a single Gaussian distribution.

Subsequent to the distribution determination, the application proceeds to compute fits for Gaussian, Landau, and Breit-Wigner distributions. A fit specific to a double Gaussian distribution is presented only when the double Gaussian analysis is acceptable.

Upon the completion of the fitting process, the socket provides the necessary data to the client. This includes the histogram's x and y values, identified distribution type, histogram's maximum y value, along with an array of fit parameters such as the amplitude, sigma, mean, their respective errors, the number of degrees of freedom (NDF), and chi-squared values.

Additionally, the server is equipped with two functions that allow direct command execution: one to clear the histogram's data and initiate anew, and another to suspend the data generation process temporarily. This capability ensures dynamic interaction and control over the data analysis process.

In addition to these operations, the socket, loaded with the processed data, is configured to listen on port 49152, standing ready to serve the client's requests.

3.2 Server-Client Communication via Flask and WebSockets

The real-time interaction between the server and the client in our application is performed through the use of Flask in conjunction with WebSockets. This combination not only ensures seamless data transmission but also enables the application to handle continuous streams of data efficiently, which is important for updating the client-side visualization in real-time. One of Flask's strengths lies in its extensibility, demonstrated by its compatibility with extensions that enhance its functionality, such as Flask-SocketIO for WebSocket communication.

3.2.1 WebSocket Communication

WebSockets provide a full-duplex communication channel over a single TCP connection, allowing for bidirectional data flow between the server and the client. This is particularly advantageous for applications requiring real-time updates[45]:

1. **Persistent Connection:** Unlike traditional HTTP connections, which are stateless and closed after a response is sent, a WebSocket connection remains open, allowing continuous data exchange throughout the life of the connection.

2. **Real-Time Data Transfer:** WebSockets eliminate the need for polling or long-polling by enabling the server to push updates to the client as soon as new data is available. This is crucial for our application, where histogram data needs to be updated in real-time.

3. **Efficiency and Scalability:** By reducing overhead and latency associated with establishing connections, WebSockets enhance the efficiency and scalability of applications dealing with real-time data.

3.2.2 Integration of Flask with WebSockets

Our application uses Flask-SocketIO, an extension that integrates WebSocket communication into Flask applications. This allows for event-driven communication between the server and the client[46]:

1. **Event Handling:** Flask-SocketIO facilitates defining custom events that the server can emit and the client can listen for, enabling structured and efficient data exchange. For instance, when new histogram data is ready, the server emits an event with the updated data payload, which the client listens for and processes to update the visualization.

2. **Room Support:** It also supports the concept of rooms, allowing the server to broadcast messages to subsets of connected clients, further enhancing the application's scalability and flexibility.

3. **Fallback Support:** Flask-SocketIO provides built-in support for fallbacks to long-polling in environments where WebSocket is not supported, ensuring broad compatibility and reliability.

All the data necessary for the client are calculated from our python server and emited on the *update_histogram* event, which the client can listen to and collect the data

The integration of Flask and WebSockets into our application architecture represents a sophisticated approach to achieving real-time server-client communication. This setup not only facilitates the immediate transmission of histogram updates to the client but also exemplifies a modern web application capable of handling dynamic data streams with high efficiency and low latency. The use of Flask-SocketIO embodies the synergy between traditional web server frameworks and contemporary real-time communication protocols, laying a robust foundation for the development of interactive and responsive web applications.

3.3 Client

Before delving into the specifics of the client-side implementation, it is imperative to outline the infrastructure that enables remote access to the application's user interface. Our solution uses Duck DNS[47], a dynamic DNS service, to associate a memorable domain name with the application's public IP address, thus facilitating access from any location. The application is hosted on a server configured to listen on port 3000, with port forwarding set up on the router to direct incoming requests to the appropriate server within the local network.
However, a challenge with dynamic IP addresses assigned by Internet Service Providers (ISPs) is their propensity to change, particularly following a power outage or router reset. Traditionally, this issue is mitigated through the use of static IP addresses, but our approach incorporates a cost-effective and efficient workaround.

3.3.1 Automated DNS Update Mechanism

We devised a Python script that executes at 20-minute intervals to monitor changes in the server's public IP address. Should a discrepancy be detected, indicating an IP change, the script updates the Duck DNS entry via its API. This automated mechanism ensures that the domain name consistently resolves to the current IP address, thereby maintaining uninterrupted access to the application's UI without the need for manual intervention or the expense of a static IP.

3.3.2 Client-Side Implementation Using React and TypeScript

The client-side architecture of our application is built upon React and TypeScript, chosen for their respective advantages in facilitating web application development. React's design philosophy promotes the decomposition of the user interface into discrete components, enhancing modularity and reusability. This framework employs a reactive data flow, wherein the DOM is efficiently updated in response to state changes within components. Such an approach ensures that modifications at any level of the component hierarchy triggers a selective re-rendering process, optimizing performance and user experience.
React's component-based architecture is complemented by TypeScript, which extends JavaScript by adding type safety and static typing. This integration brings together the flexibility of JavaScript with the robustness of static type checking, significantly reducing runtime errors and facilitating large-scale application development.

3.3.3 State Management and Component Re-rendering

In React, the re-rendering of elements depends upon state updates. When a component's state changes, React updates the DOM to reflect these changes, re-rendering only the affected components rather than the entire application. This approach to DOM updates is important for achieving high performance and responsive interfaces. For instance, a state change in a parent component cascades down, prompting the re-rendering of its child components. Conversely, state updates in a child component do not affect its siblings or parent, minimizing unnecessary rendering and enhancing the application's efficiency.

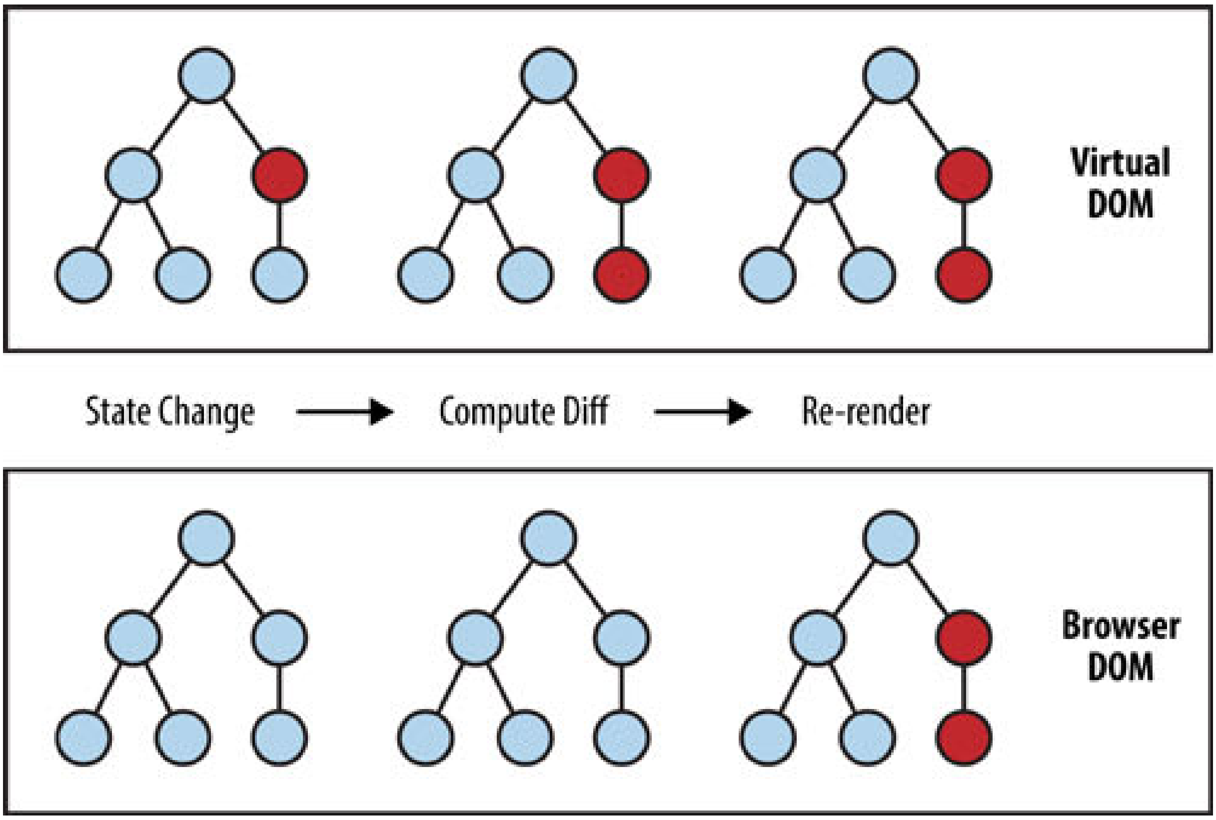

Figure 4: React Re-Rendering Components.

3.3.4 TypeScript for Enhanced Development Experience

Complementing React's dynamic capabilities, TypeScript introduces type safety and static analysis to the development process. By enforcing typing disciplines, TypeScript aids in catching errors early in the development cycle, debugging and ensuring more reliable code. This synergy between React's reactive component model and TypeScript's static typing creates a robust foundation for developing complex, scalable web applications.

The utilization of React and TypeScript is a good choice for our application, taking advantage of React's efficient rendering strategies and TypeScript's type safety to deliver a scalable, maintainable, and user-friendly client-side experience.

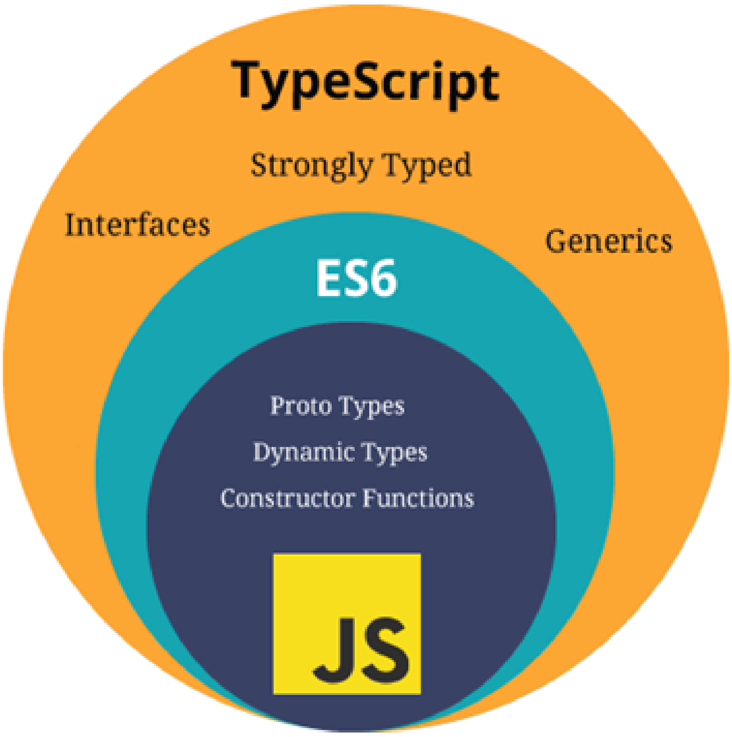

Figure 5: Typescript superset of Javascript.

3.3.5 Techincal Architecture

The architecture of our application is based on the principle of maximizing reusability through modular component design. This approach not only enhances the efficiency of the application but also facilitates ease of maintenance and scalability. Central to our architectural strategy is the decomposition of the application into discrete, purpose-specific components.

3.3.6 React Hooks: useState and useEffect

React Hooks, introduced in React 16.8, represent a significant advancement in the way functional components are written, and state is managed in React applications. Among the Hooks, `useState` and `useEffect` are two fundamental hooks that enable stateful logic and side effects in functional components, which were previously only possible in class components. This section provides a detailed description of these hooks and their role in the technical implementation of the web application[48].

useState Hook

The `useState` hook is a cornerstone of state management within functional components. It allows developers to declare state variables in a component, thereby enabling the component to preserve state between re-renders.

Declaration and Initialization The `useState` hook is invoked within a functional component and requires an initial state value. It returns an array containing two elements: the current state value and a function to update it. The syntax for declaring a state variable with `useState` is as follows:

```
const [state, setState] = useState(initialState);
```

The `initialState` can be a simple value like a number or string, or a complex object, depending on the requirements of the component. This initial state is only used during the first render.

State Updates The state updating function, commonly denoted as `setState`, is used to schedule updates to the component's state. When state changes occur, React re-renders the component with the updated state, allowing the application to respond to user input, API calls, or other events.

useEffect Hook

The `useEffect` hook serves as the gateway for performing side effects in functional components. Side effects are operations that can affect other components, perform I/O operations, or execute asynchronous tasks, such as data fetching, subscriptions, or manually changing the DOM.

Effect Execution `useEffect` is called inside the component to register an effect. It takes a function, known as the effect callback, which contains the code to be executed when the effect runs. Optionally, a second argument, an array of dependencies, can be passed to `useEffect`, which determines when the effect should re-run.

```
useEffect(() => {
    // Side effect logic here
}, [dependencies]);
```

Dependency Array The dependency array is a feature of `useEffect`. If the array includes variables or props, the effect will re-run only when those dependencies change. If the array is empty, the effect runs once after the initial render, mimicking the behavior of `componentDidMount` in class components. If no array is provided, the effect runs after every render.

Cleaning up an Effect Effects may also return a cleanup function that React calls when the component unmounts or before re-running the effect due to dependency changes. This cleanup function is crucial for preventing memory leaks, removing event listeners, or canceling network requests.

3.3.7 Core Components

The application's structure is anchored by the `App` folder, which splits into two primary components: the *Home Page* and the *Histogram Page*.

Histogram Page

The *Histogram Page* serves as the nucleus for the application's data visualization capabilities, particularly the rendering of histograms. This page is further linked to two significant components:

1. **Plot Component:** Utilizes the Plotly library to provide dynamic and interactive histogram visualization. This integration uses Plotly's comprehensive plotting capabilities to render histograms based on real-time data.

2. **Button Bar Component:** Consolidates all interactive elements of the application, including:

    - Information tool tips for user guidance.
    - Selection options for different fit visualizations.
    - Functionality to find the maximum values within the plot.
    - Options to toggle the plot's scale (linear or logarithmic).
    - Data export to ODS format and import from ODS, facilitating data portability.
    - Controls to stop data fetching, clear existing data (both client-side and server-side), and restart data acquisition.
    - Editing tools for histogram axis titles, allowing customization of the visualization.

3.3.8 Navigation and User Interface

Enhancing the user experience, the application features a *Navigation Bar*, enabling seamless transitions between the Home Page and Histogram Data visualization. This navigation mechanism preserves the state of each page, obviating the need for re-rendering upon return, thereby optimizing performance and user experience. Moreover, the Navigation Bar offers an option to activate a *Dark Mode*.

3.3.9 HistogramData Component Analysis

The `HistogramData` component in React represents a crucial element within the application, responsible for rendering the histogram visualization and managing its associated state. This component utilizes the functional component paradigm in React, using hooks such as `useState`, `useEffect` and `useRef` to maintain and manipulate its state and side effects.

State Management

- The `plotData` state initializes the data structure for the plot with default values provided by the `defaultPlotValues` object, ensuring a consistent starting point for the histogram rendering.

- The `titles` state manages the axis labels and the histogram's title, which dynamically reflect the current distribution type being visualized.

- The `histogramShowBooleans` state contains a set of boolean flags that control various UI elements, such as the visibility of results, fit, and the use of a logarithmic scale.

- The `distributionsBooleans` state determines which distribution fits should be rendered based on user interaction.
- The `layout` state maintains the graphical layout of the histogram, which includes the plot's dimensions, axis ranges, and annotations.

Effect Hooks Several `useEffect` hooks are utilized to handle side effects such as:

- Adjusting to screen size changes
- Establishing and cleaning up the WebSocket connection
- Fetching the current IP at regular intervals
- Updating the plot's layout when dependent state variables change

Socket.IO Integration The component establishes a WebSocket connection to the backend server using the `socket.io-client` library. It listens for `"update_histogram"` events emitted by the server, which trigger updates to the `plotData` state, thus re-rendering the histogram with the latest data.

Responsive Design The `handleResize` function adjusts the plot size based on the user's window dimensions, ensuring that the histogram remains visually appealing and functional across different devices and screen sizes.

Dynamic IP Handling The component features a method, `getCurrentIp`, which fetches the current IP address. This method is called periodically, aligning with the dynamic DNS updates to maintain access to the server if the public IP changes due to network resets or other reasons.

Plot Rendering The component uses the `Plot` component from the `react-plotly.js` library to render the histogram. This is a React wrapper around the Plotly.js library, providing a rich set of features for data visualization.

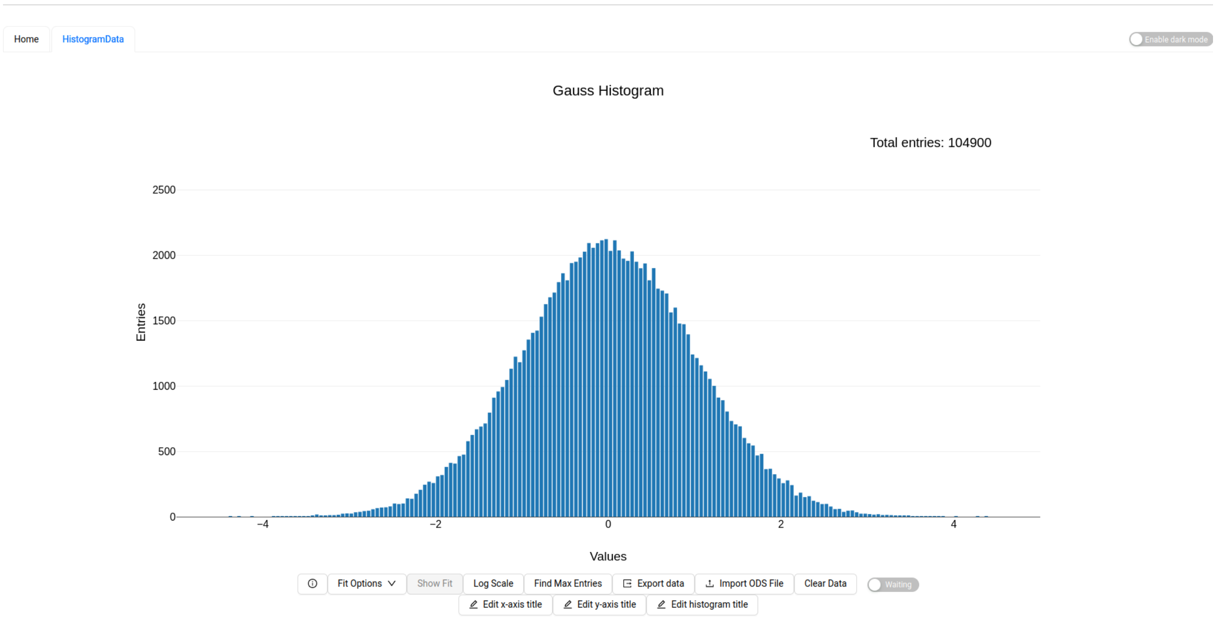

Figure 6: Histogram Page.

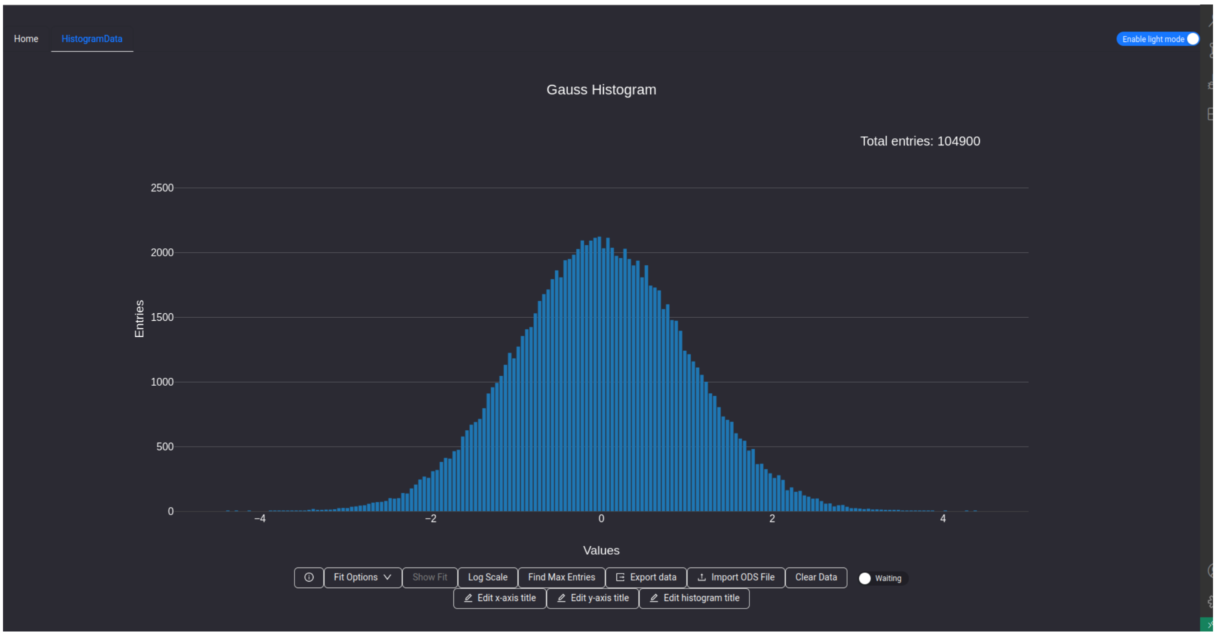

Figure 7: Histogram Page Dark Mode.

3.3.10 HistogramData Functional Flow

The client's operational flow is initiated upon the reception of data through the established `socket.io` connection, triggering the `update_histogram` event. This connection is needed for the real-time update of the `plotData` state with the incoming dataset, providing the histogram's `x` and `y` values, the computed `y` fit data for each distribution model, and the corresponding fit parameters along with their errors. The server-determined distribution type dictates the histogram's title, defaulting to "Gauss Histogram" in the event of a Gaussian distribution.

3.3.11 Dynamic Plot Layout Adjustment

A function dedicated to updating the plot's layout is invoked concurrently, tasked with checking the data to optimize the plot's presentation—automatically adjusting limits, height, and other layout parameters. This function is also responsible for reacting to user interaction that alters the plot's current view, such as zooming in, toggling logarithmic scales, or displaying fit curves. As a result, the histogram plot is dynamically refreshed to reflect the latest data and user actions.

3.3.12 Interactive Plot Features

The plot uses an interactive interface, displaying the total number of entries in the top-right corner and allowing users to zoom, pan, and navigate through the data space.

3.3.13 Button Bar Interactions

Each button in the button bar has a specific role in manipulating the histogram display or the underlying data. Below is a detailed analysis of each button's functionality and the resulting changes in the application's user interface, illustrated with corresponding figures.

Information Tooltip Upon hovering over the information tooltip, a brief explanation of the histogram's functionalities is displayed to the user. This tooltip is designed to assist new users in navigating the application and understanding the available tools.

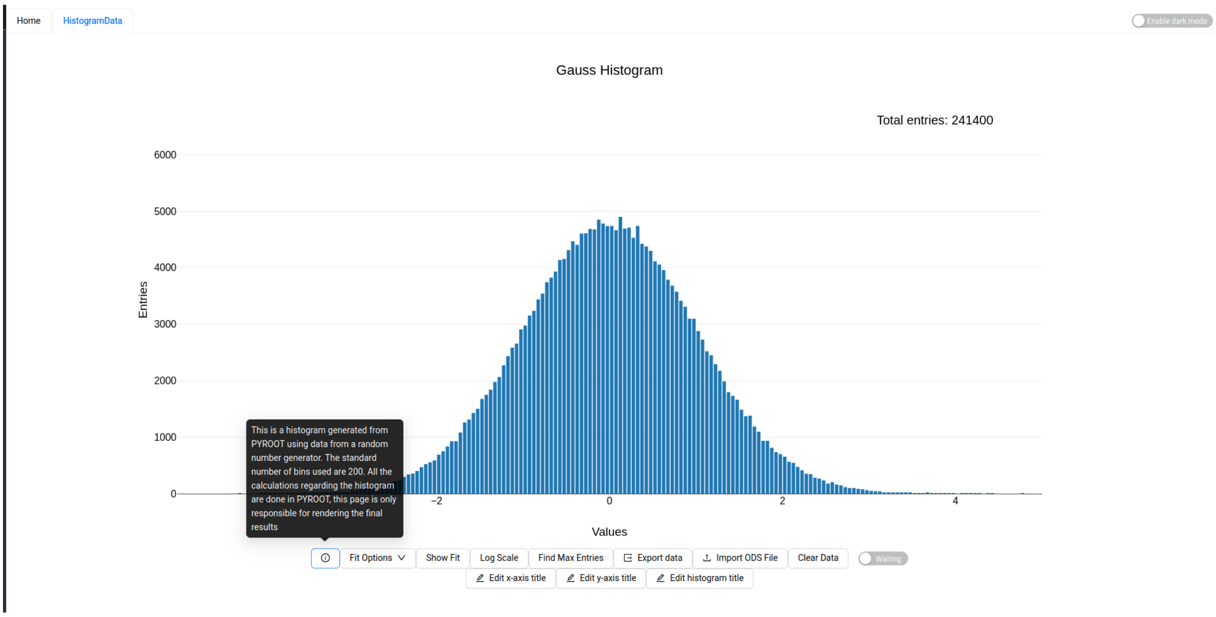

Figure 8: Display of the information tooltip providing guidance on the histogram's functionalities.

Fit Selection Dropdown Menu Users can select from a variety of statistical fits — Gaussian, Landau, Breit-Wigner, and Double Gaussian — via the fit selection dropdown menu. The application dynamically calculates and visualizes the selected fit(s) overlaying them on the histogram data.

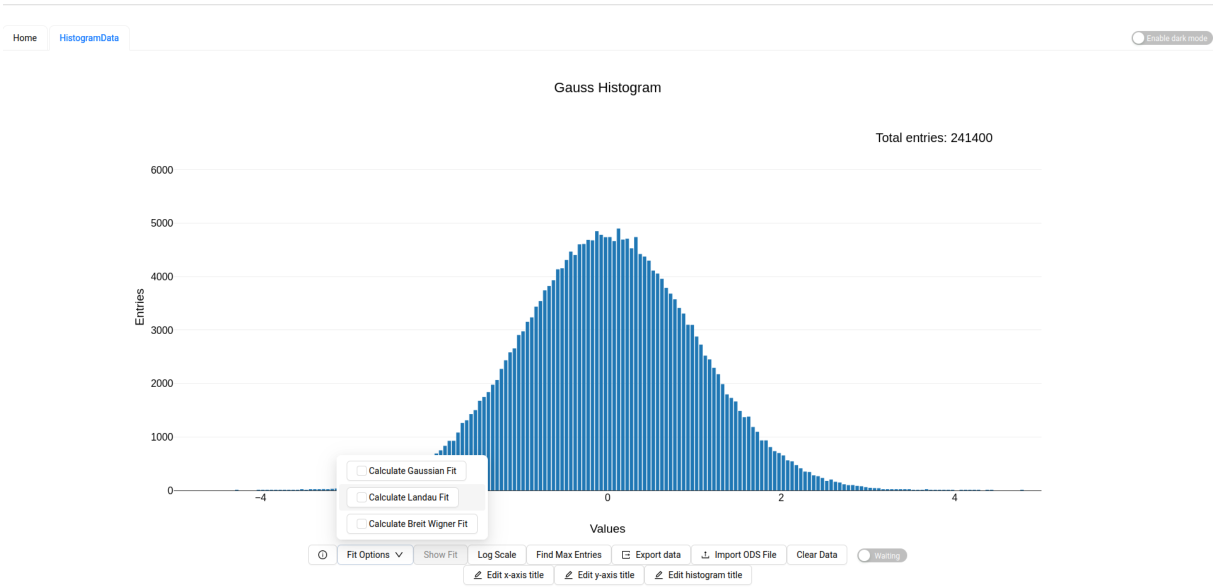

Figure 9: Fit selection dropdown with multiple options for fitting the histogram data.

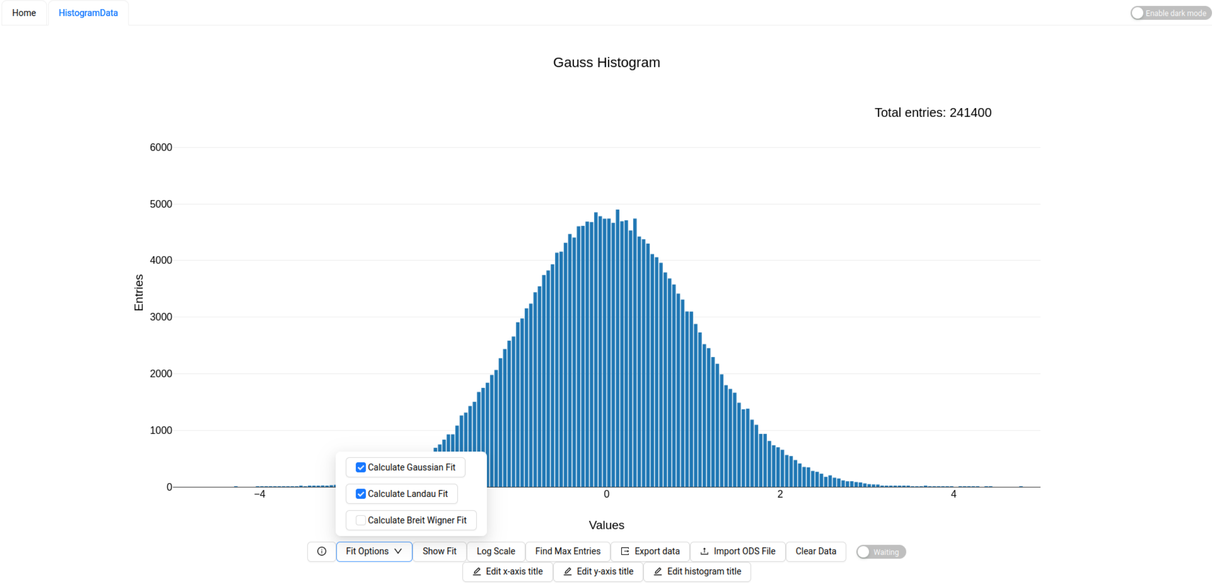

Figure 10: Fit selection dropdown with multiple selected options.

If a double gaussian distribution is detected then an additional choice will appear in the dropdown menu with three options: Get the double gaussian fit or any of the two component gaussians.

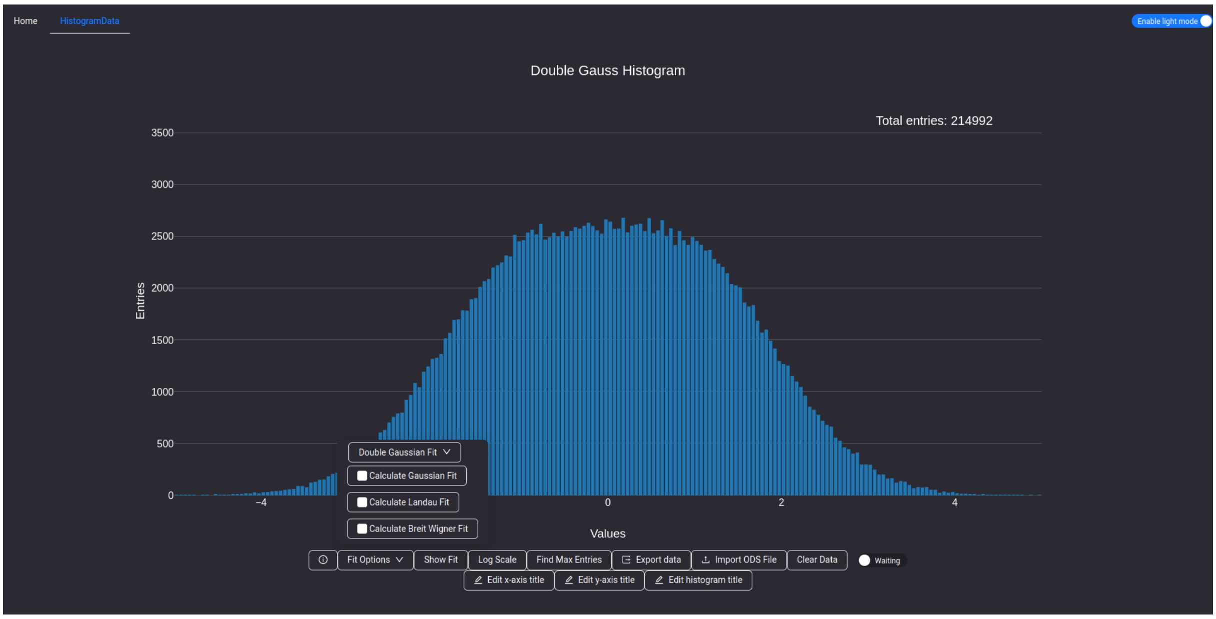

Figure 11: Fit selection dropdown of double gaussian distribution.

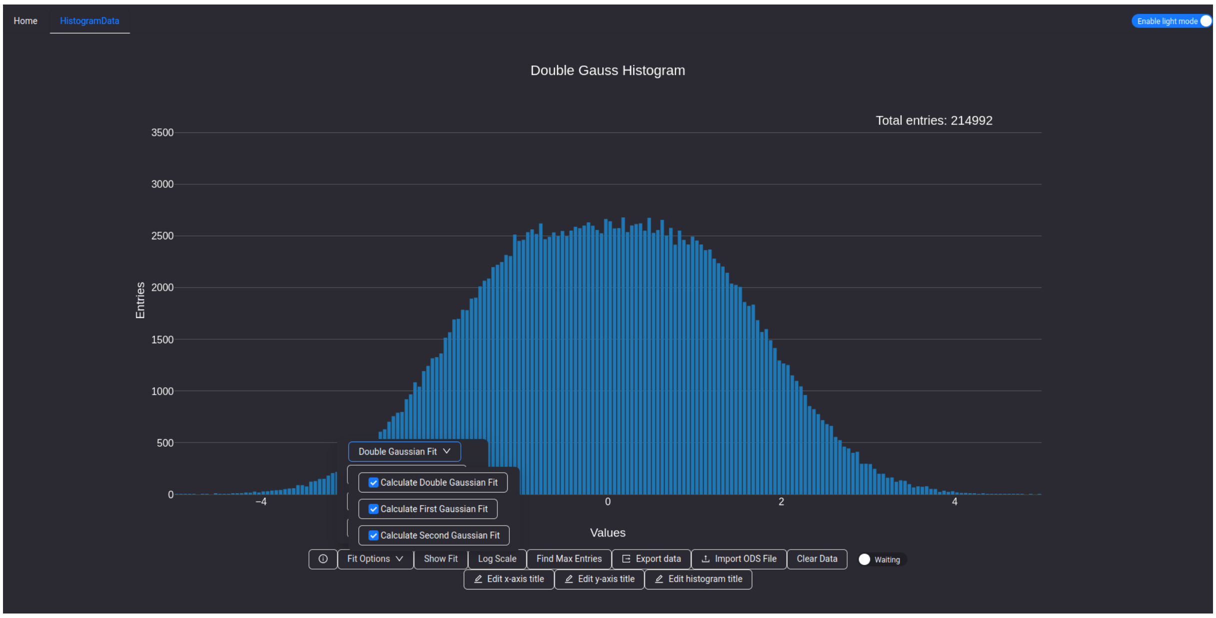

Figure 12: Fit selection dropdown with double gaussian options selected.

Show Fit Button Once a fit is selected, the 'Show Fit' button will become active. Clicking this button overlays the chosen fit curve(s) onto the histogram. Figure 13 illustrates the histogram after the Gaussian fit has been applied and displayed.

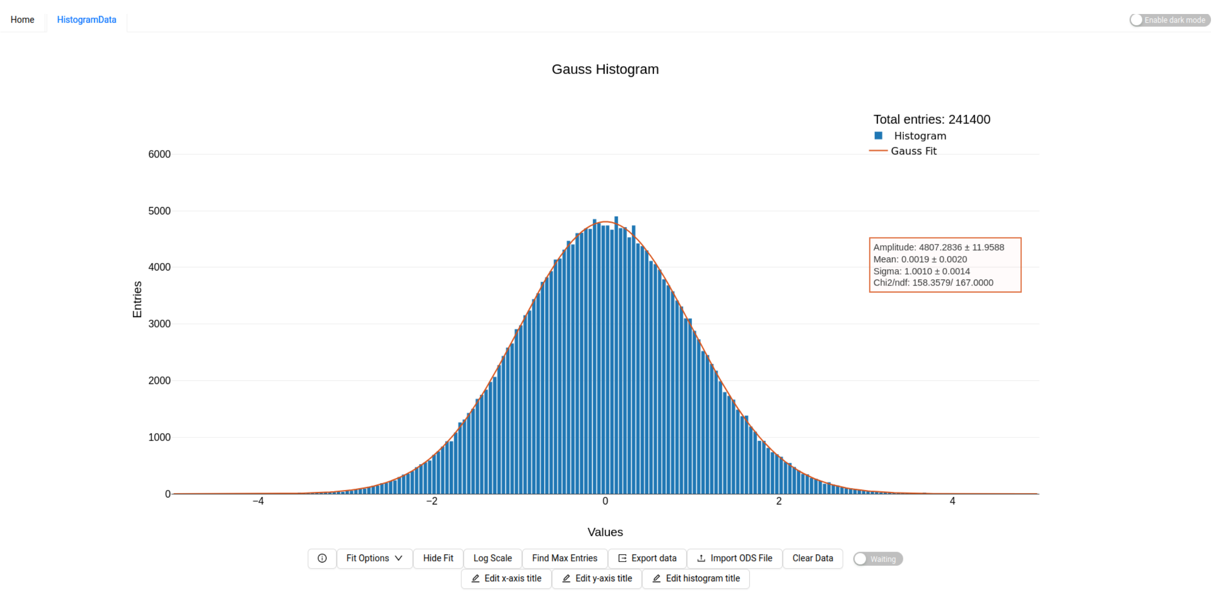

Figure 13: Histogram with Gaussian and Landau fit curve overlaid following the activation of the 'Show Fit' button.

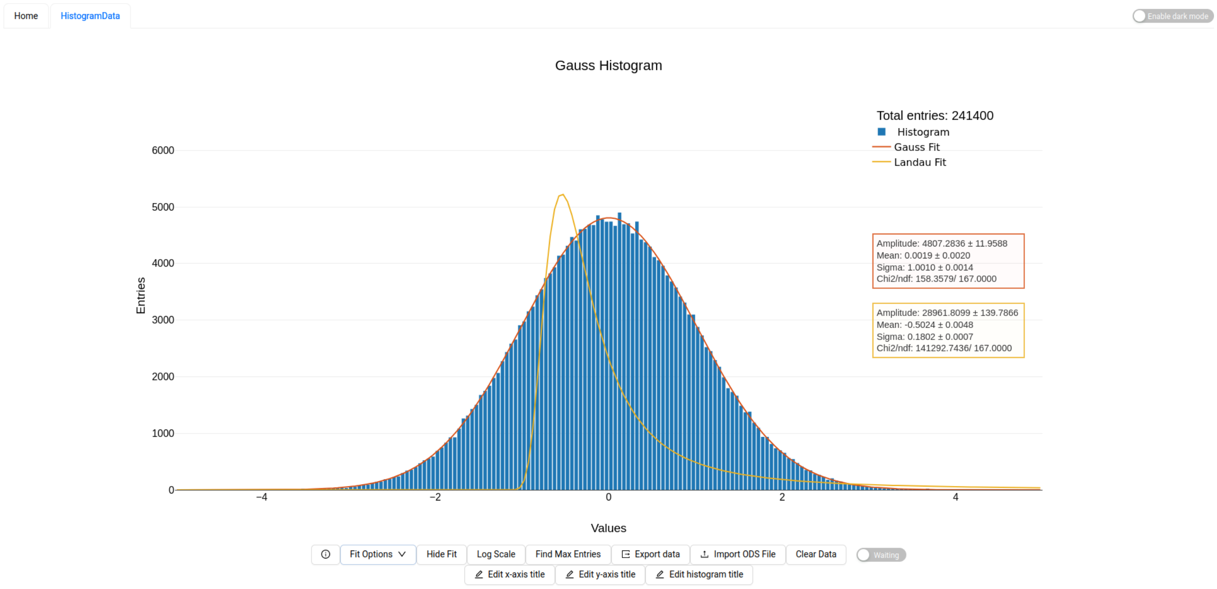

Figure 14: Histogram with the Gaussian fit curve overlaid following the activation of the 'Show Fit' button.

The user may also choose to hide from the plot the current set of data by clicking on the top right corner and selecting the corresponding data set to be removed.

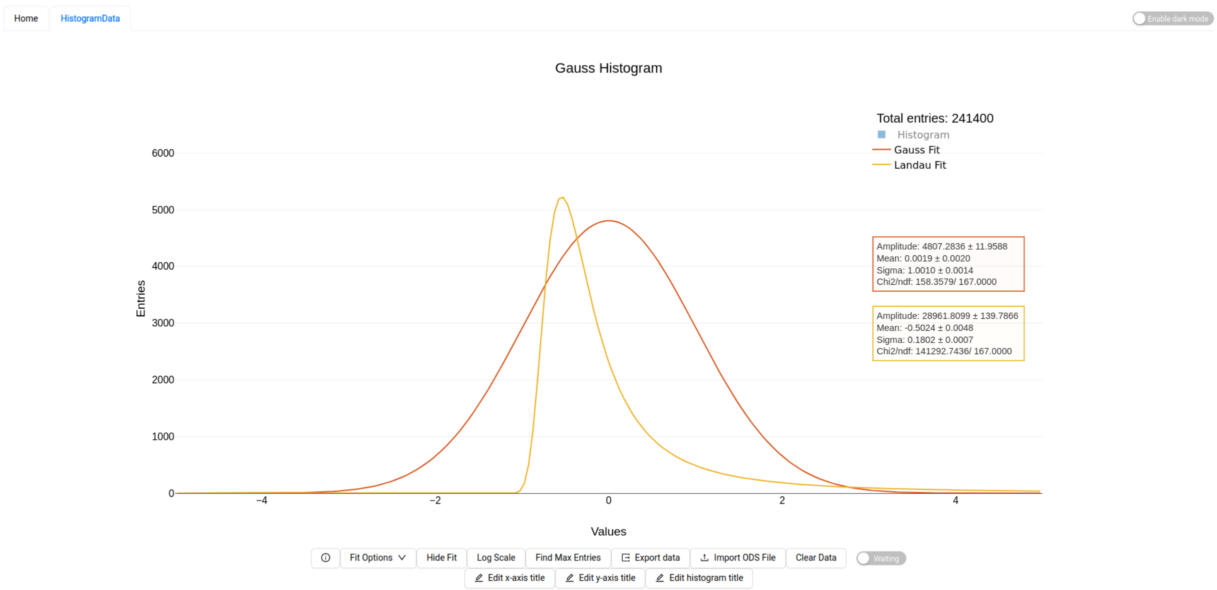

Figure 15: Histogram fit with deselected original data.

Figure 16 shows a double gaussian distribution where the user has selected to see the corresponding double gaussian fits.

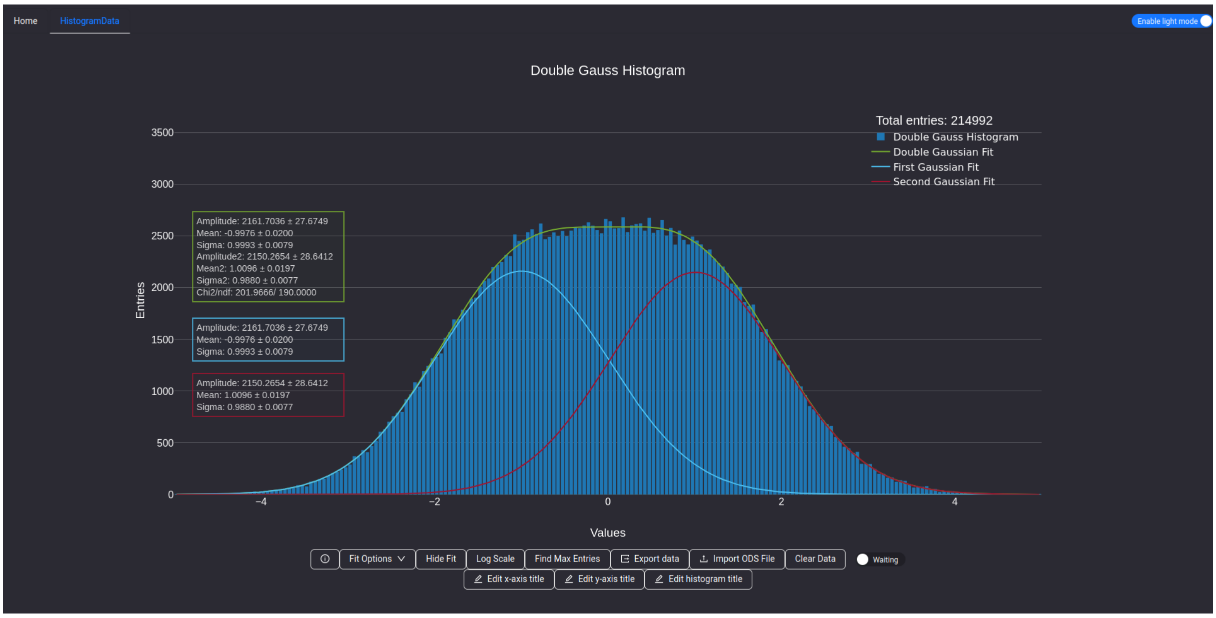

Figure 16: Representation of a double Gaussian distribution fit, showcasing the intricate interplay between the composite double Gaussian function and its constituent Gaussian components.

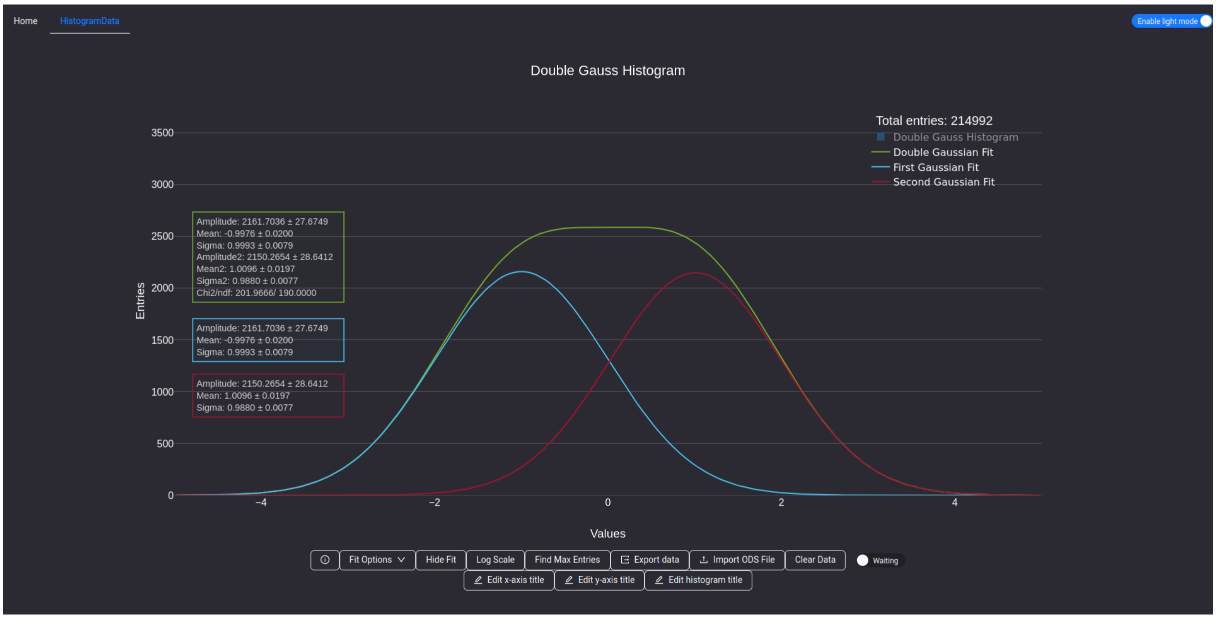

Figure 17: Representation of a double Gaussian distribution fit, showcasing the intricate interplay between the composite double Gaussian function and its constituent Gaussian components with primary histogram data deselected.

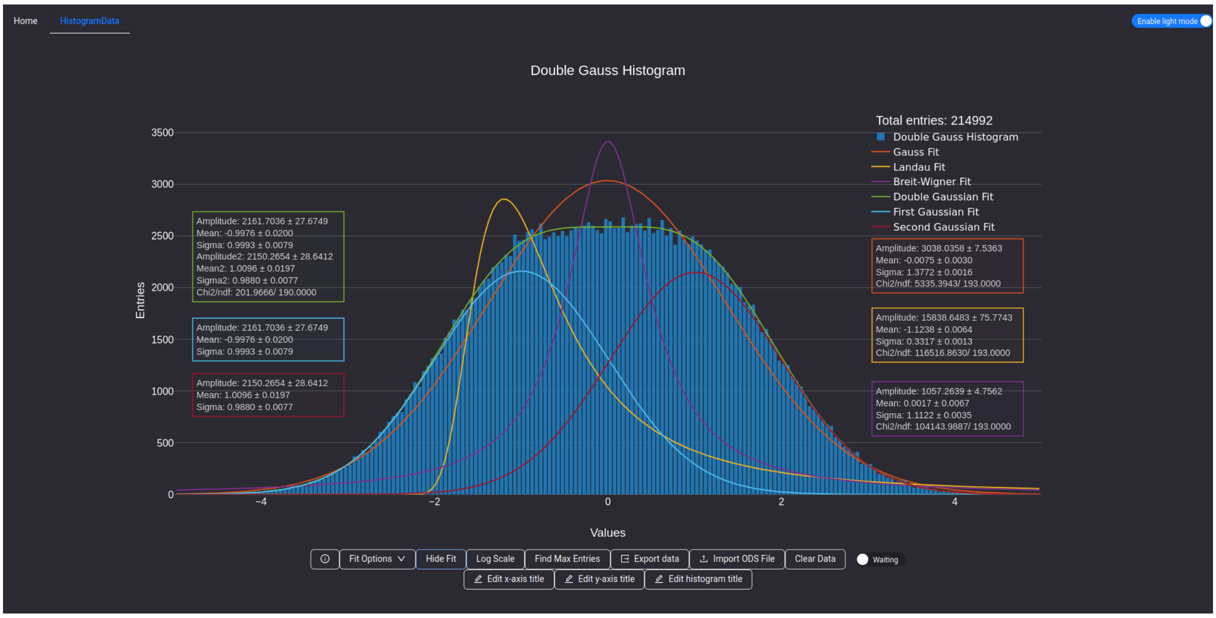

Figure 18: All fits selected in the histogram.

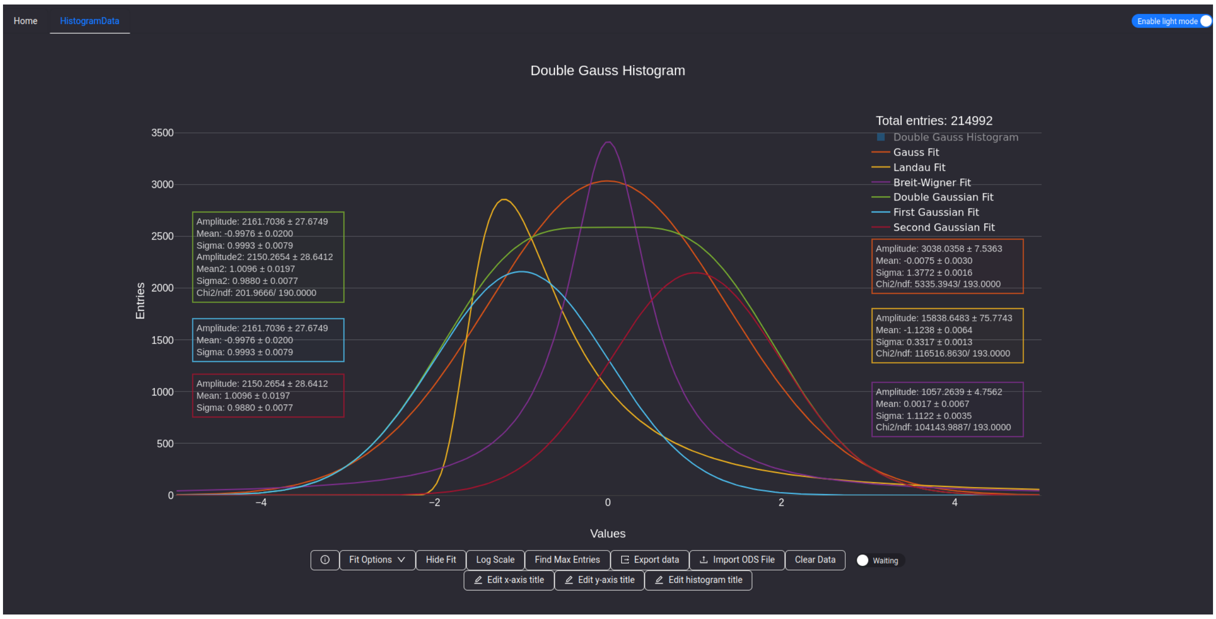

Figure 19: All fits selected in the histogram with primary data deselected.

Logarithmic and Linear Scale Toggle The scale toggle button allows users to switch between logarithmic and linear y-axis representations of the histogram.

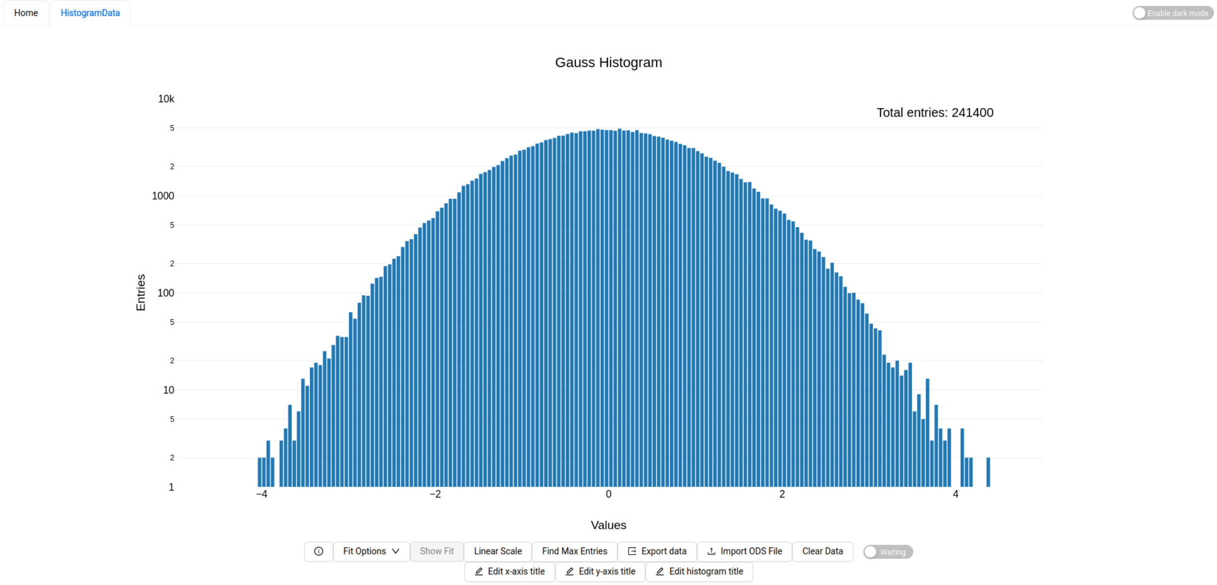

Figure 20: Histogram displaying data in logarithmic scale, enhancing the visibility of data points with smaller counts.

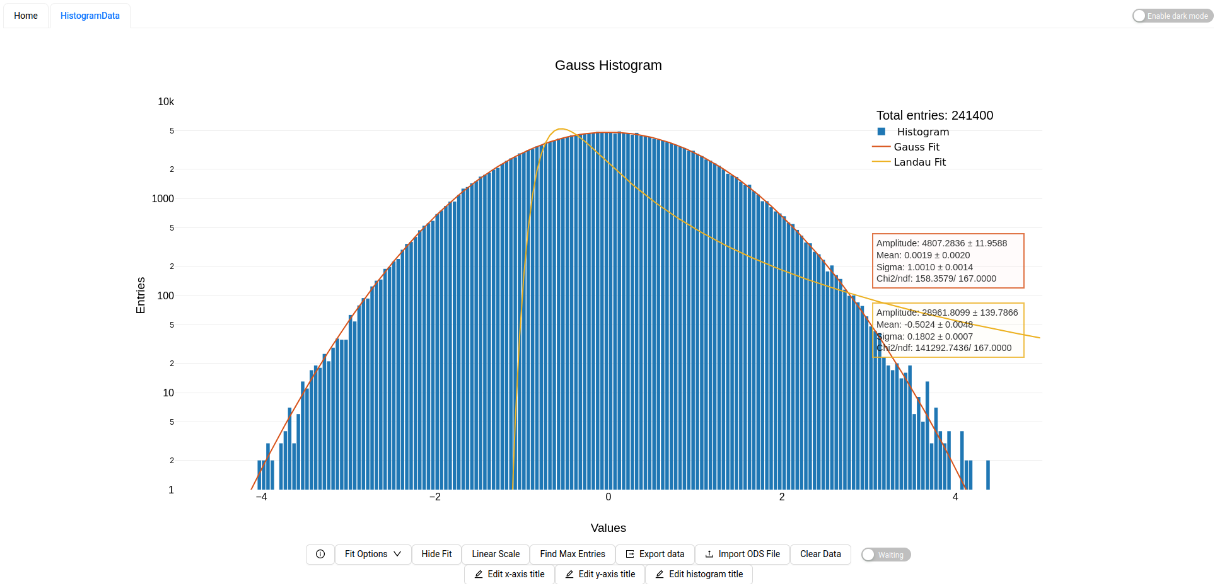

Figure 21: Logarithmic scale with selected fits.

Data Export and Import The 'Export data' button triggers a download of the current histogram data in ODS format

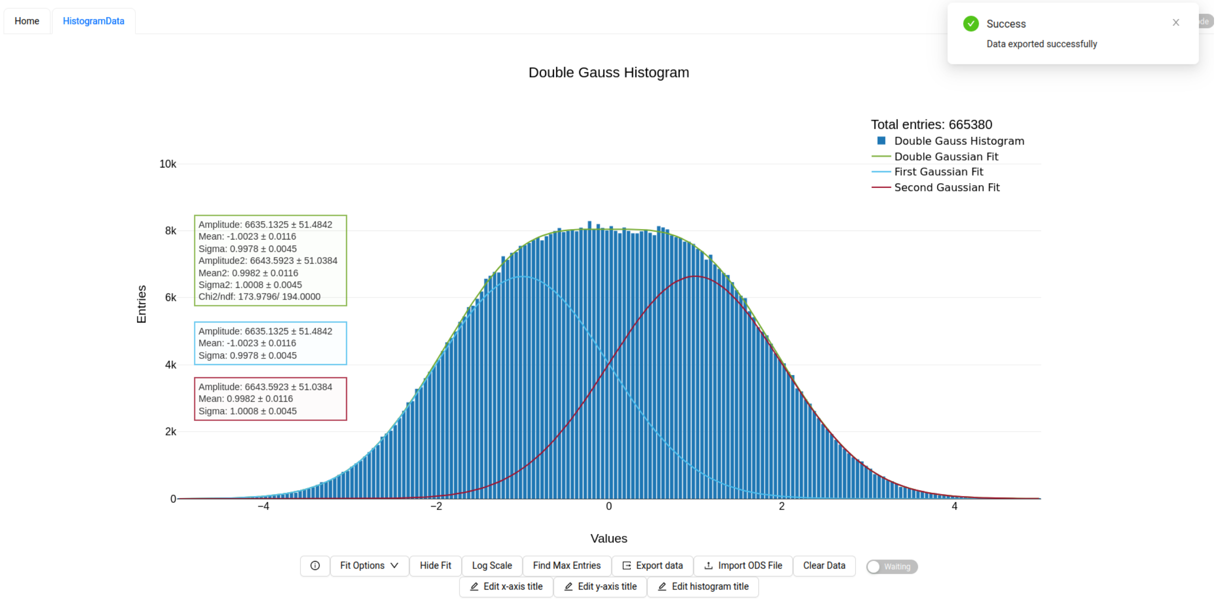

Figure 22: Success message when exporting data.

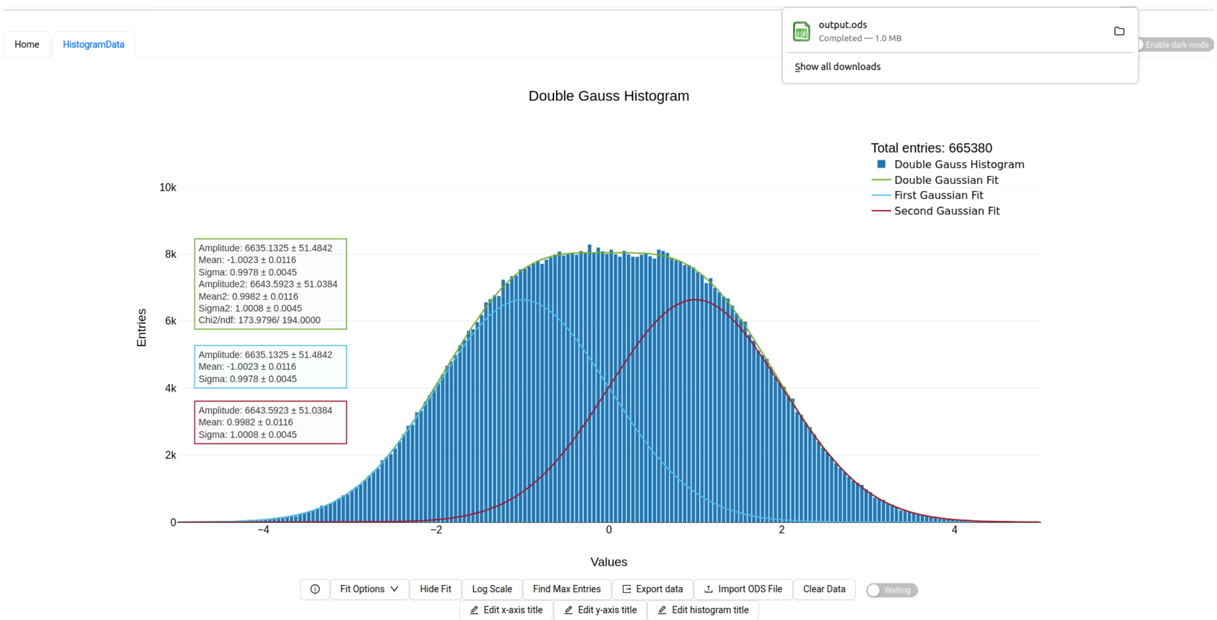

Figure 23: Ods file generated after exporting data.

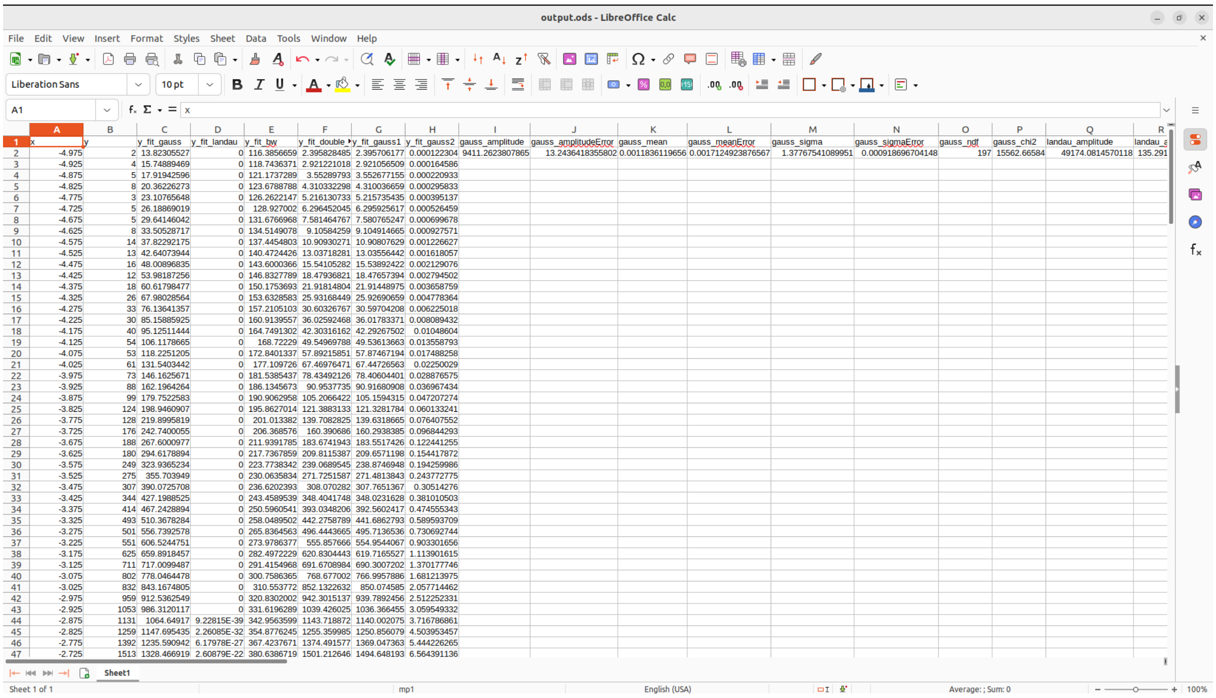

Figure 24: Content of the created ods file.

The 'Import ODS File' button allows users to upload and visualize previously saved histogram states.

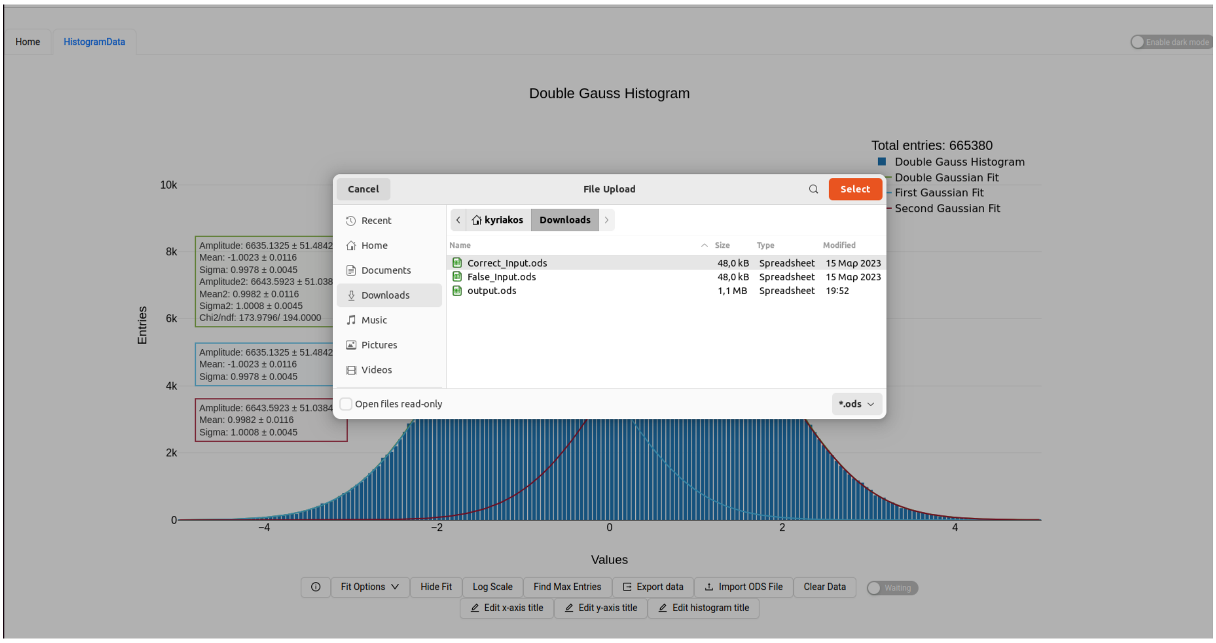

Figure 25: File search when trying to import from ods file.

If an ODS file with the wrong format is provided, the application will throw an error indicating exactly what went wrong.

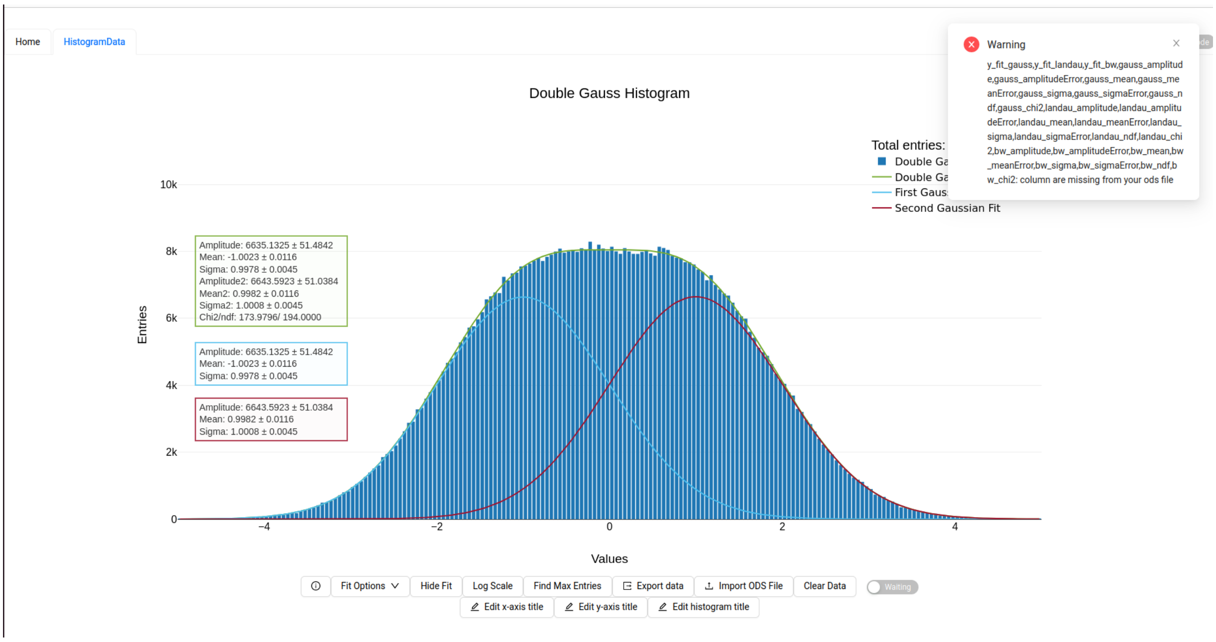

Figure 26: Error when providing a wrong ods file.

Clear Data and await/continue fetching These two buttons provide the utility of either clearing all existing data from both client and the server or momentarily freezing the current histogram snapshot.

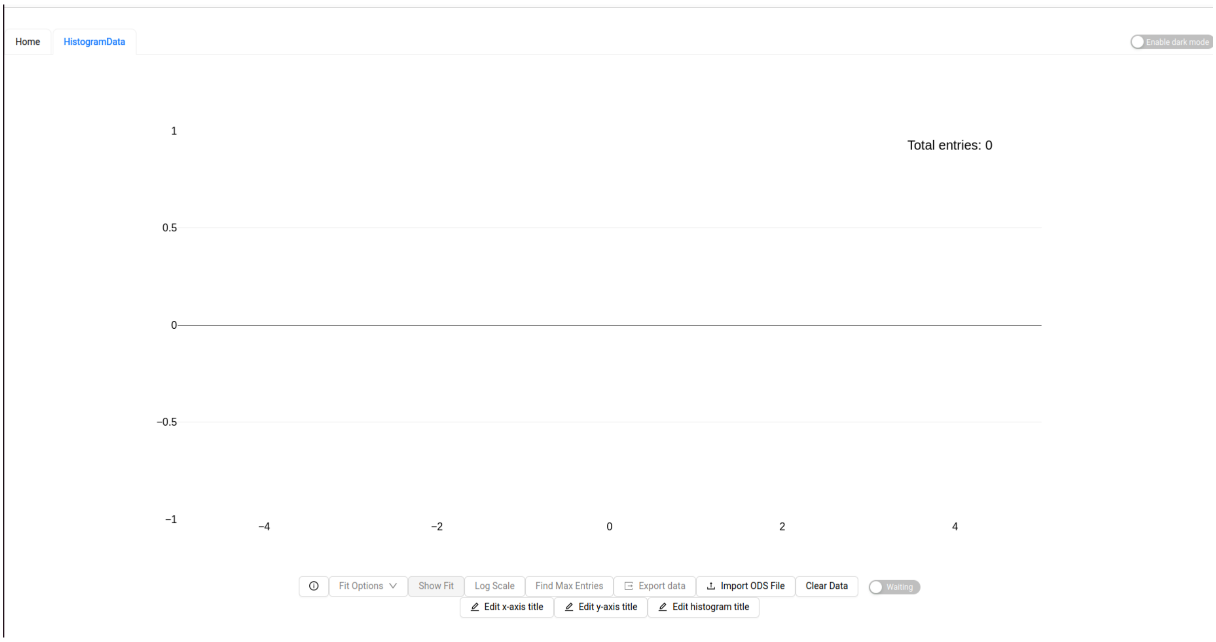

Figure 27: Clearing all existing data.

Adjusting title names The application allows the user to customize the titles of the histogram's axes and its overall title.

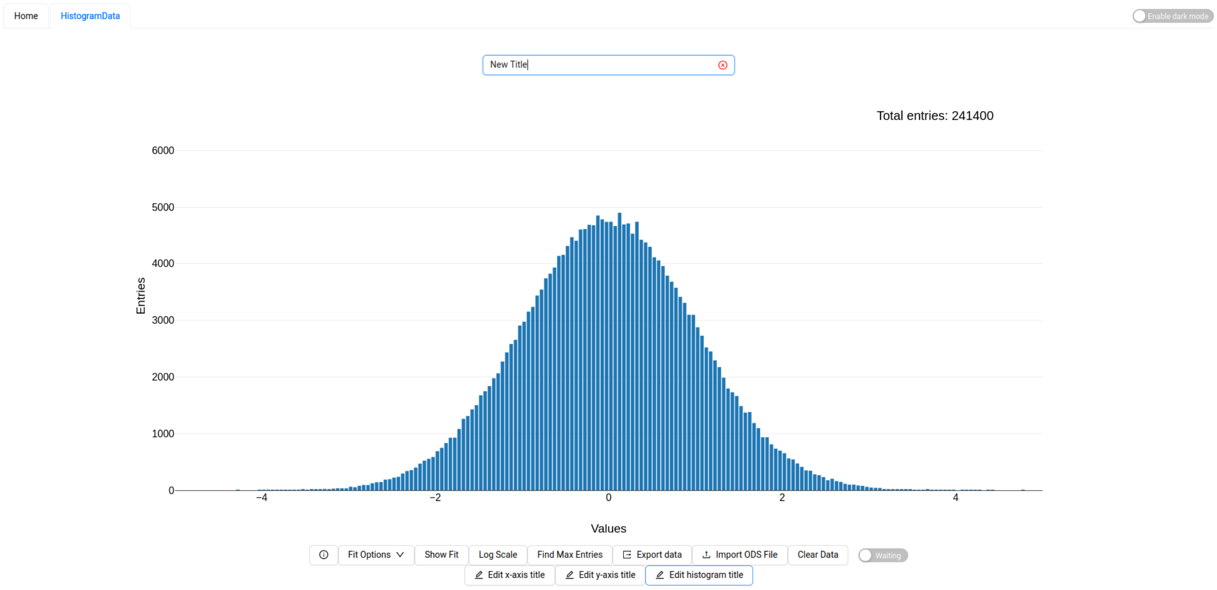

Figure 28: Editing histogram title.

3.3.14 A Closer Look into the Results

The analysis pipeline of our client-side application is meticulously designed to offer a comprehensive walkthrough of data as it traverses through various stages of processing, culminating in the visual representation and statistical analysis of the histogram data. This section aims to elucidate the data flow within the client and the mechanisms underlying the determination of the most suitable fit based on the provided dataset.

Data Flow and Visualization Upon receipt of the histogram data from the server — x and y values, fit parameters, and their respective errors — the client initiates a sequence of operations to render the histogram alongside the fit curves. The data undergoes normalization and scaling to match the plot dimensions, ensuring that the visualization accurately represents the underlying distribution.

Evaluating Fit Precision A critical aspect of our analysis involves evaluating the precision of various fits applied to the histogram data. This evaluation uses the chi-squared per degrees of freedom (χ^2/NDF) ratio, a statistical metric that offers insight into the goodness of fit. A χ^2/NDF ratio of $\sim$1 indicates a fit that most accurately describes the dataset.

Impact of Data Volume on Analysis Accuracy The reliability and accuracy of fit evaluations are intrinsically tied to the number of entries constituting the histogram. Initial datasets comprising a limited number of entries (e.g., 100 points) may yield preliminary insights; however, these early analyses lack the precision attainable with more substantial datasets. As the histogram accumulates more data (e.g., 100,000 points), the statistical significance of the fits improves.

4 Future Directions and Improvements

While the web-based application developed in this thesis serves as a robust tool for data analysis and visualization, there are several areas where enhancements could further augment its functionality, usability, and efficiency. These improvements not only aim to refine the current capabilities but also to extend the application's utility in addressing more complex data analysis challenges.

Advanced Data Analysis Features Integrating more sophisticated data analysis algorithms could provide users with deeper insights into their data. This includes the implementation of machine learning models for predictive analysis and anomaly detection, offering a deeper understanding of data patterns and behaviors.

User Interface and Experience Enhancements Although the client-side application uses an intuitive and responsive interface, further refinements of the user interface design could enhance user engagement and ease of use. This could involve the incorporation of customizable dashboards, interactive data filtering options etc.

Performance Optimization As data volumes continue to grow, optimizing the application's performance to handle larger datasets efficiently becomes crucial. This could involve refining the backend data processing algorithms, introducing the use of parallel computing techniques, or adopting more efficient data storage and retrieval mechanisms.

Security Enhancements With the increasing importance of data privacy and security, strengthening the application's security measures to protect user data is paramount. Implementing more robust authentication protocols and data encryption methods would safeguard against unauthorized access and data breaches.

Scalability and Deployment Enhancing the application's scalability to support a growing number of users and simultaneous data analysis tasks is another area for improvement. This could be achieved through cloud-based deployment strategies, containerization, and the use of scalable architecture patterns such as microservices. Additionally, an on-hardware implementation remains a viable option. Leveraging FPGA-based systems, such as Zynq SoCs, could provide dedicated hardware acceleration for data processing tasks. This approach can offload intensive computations from the server, resulting in lower latency and higher throughput. While this implementation was not feasible within the given project timeline, future work could explore the integration of hardware accelerators to further enhance the application's performance and efficiency.

5 Conclusion

This thesis has presented a detailed exploration of the design and implementation of a web-based application for the analysis and visualization of data derived from a random number generator, utilizing the ROOT library for data processing and Flask for server-client communication. From the server's meticulous construction for efficient data handling and analysis to the client's dynamic and interactive data visualization capabilities, we have navigated through the challenges of integrating Python, Flask, PyROOT, React, and TypeScript to build a system that not only meets the project's requirements but is also scalable and extendable for future development.

Server to Client Data Flow The server, designed with an object-oriented approach, generates, processes and fits histogram data, determining the data distribution through sophisticated statistical analysis. using Flask and WebSockets, processed data are then seamlessly transferred to the client, ensuring real-time data communication and interactive user engagement.

Client-Side Visualization and Analysis On the client side, React and TypeScript combination use facilitates a modular and efficient application structure, promoting reusability and maintainability. Through interactive components and dynamic data visualization with Plotly, users can explore the data in depth, adjust visualization parameters, and derive meaningful insights from the analysis.
In conclusion, this thesis has not only demonstrated the capability of integrating various technologies to develop a comprehensive data analysis and visualization tool but also highlighted the importance of thoughtful system design in creating scalable and extendable software applications. The journey from server to client encapsulates a holistic approach to solving complex data analysis challenges, paving the way for future innovation in the field.
As the volume of data continues to grow, the need for effective data quality monitoring systems becomes increasingly critical. This thesis contributes to this ongoing effort, providing a blueprint for developing applications that can adapt to and evolve with the ever-changing landscape of data analysis.

References

[1] True Random Number Generation and Evaluation Using Silicon Die Process Variations in FPGAs. `http://dx.doi.org/10.26268/heal.uoi.12113`

[2] Redman, T. C. (1996). Data Quality for the Information Age. Artech House.

[3] Kimball, R., Ross, M. (2008). The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling. Wiley.

[4] Wang, R. Y., Strong, D. M. (1996). Beyond Accuracy: What Data Quality Means to Data Consumers. Journal of Management Information Systems, 12(4), 5-34.

[5] Strong, D. M., Lee, Y. W., Wang, R. Y. (1997). Data Quality in Context. Communications of the ACM, 40(5), 103-110.

[6] Batini, C., Scannapieco, M. (2009). Data Quality: Concepts, Methodologies and Techniques. Springer.

[7] Ballou, D. P., Pazer, H. L. (1998). Modeling Data and Process Quality in Multi-input, Multi-output Information Systems. Management Science, 31(2), 150-162.

[8] Pipino, L. L., Lee, Y. W., Wang, R. Y. (2002). Data Quality Assessment. Communications of the ACM, 45(4), 211-218.

[9] English, L. P. (1999). Improving Data Warehouse and Business Information Quality. Wiley.

[10] Huang, K. T., Lee, Y. W., Wang, R. Y. (1999). Quality Information and Knowledge. Prentice Hall.

[11] Friedman, T. (2010). The Impact of Poor Data Quality on Businesses. Gartner Research.

[12] Chen, H., Chiang, R. H., Storey, V. C. (2014). Business Intelligence and Analytics: From Big Data to Big Impact. MIS Quarterly, 36(4), 1165-1188.

[13] Cheng, C., Liu, X. (2020). Artificial Intelligence in Data Quality Management: A Review. Journal of Data and Information Quality, 12(1), 1-27.

[14] Khatri, V., Brown, C. V. (2010). Designing Data Governance. Communications of the ACM, 53(1), 148-152.

[15] Bertolucci, J. (2013). Real-Time Data Quality Monitoring. InformationWeek Reports.

[16] Friedman, T. (2013). Data Quality as a Service. Gartner Research.

[17] Gentle, J. E. (2003). Random Number Generation and Monte Carlo Methods. Springer.

[18] Metropolis, N., Ulam, S. (1949). The Monte Carlo Method. Journal of the American Statistical Association, 44(247), 335-341.

[19] Ferguson, N., Schneier, B., Kohno, T. (2010). Cryptography Engineering: Design Principles and Practical Applications. Wiley.

[20] Knuth, D. E. (1997). The Art of Computer Programming, Volume 2: Seminumerical Algorithms. Addison-Wesley.

[21] Herrero-Collantes, M., Garcia-Escartin, J. C. (2017). Quantum Random Number Generators. Reviews of Modern Physics, 89(1), 015004.

[22] Menezes, A. J., van Oorschot, P. C., Vanstone, S. A. (1996). Handbook of Applied Cryptography. CRC Press.

[23] Brun, R., Rademakers, F. (1997). ROOT - An Object-Oriented Data Analysis Framework. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 389(1-2), 81-86.

[24] Antcheva, I., et al. (2009). ROOT - A C++ Framework for Petabyte Data Storage, Statistical Analysis and Visualization. Computer Physics Communications, 180(12), 2499-2512.

[25] Lutz, M. (2013). Learning Python. O'Reilly Media.

[26] McKinney, W. (2012). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. O'Reilly Media.

[27] Grinberg, M. (2018). Flask Web Development: Developing Web Applications with Python. O'Reilly Media.

[28] Facebook Inc. (2013). React: A JavaScript library for building user interfaces. Available at: https://reactjs.org

[29] Bierman, G., Abadi, M., Torgersen, M. (2014). Understanding TypeScript. In Proceedings of the 2014 ACM SIGPLAN Symposium on New Ideas, New Paradigms, and Reflections on Programming Software (pp. 1-13).

[30] Plotly Technologies Inc. (2021). Plotly: The front end for ML and data science models. Available at: https://plotly.com

[31] McAfee, A., Brynjolfsson, E. (2012). Big Data: The Management Revolution. Harvard Business Review, 90(10), 60-68.

[32] Chen, H., Chiang, R. H., Storey, V. C. (2012). Business Intelligence and Analytics: From Big Data to Big Impact. MIS Quarterly, 36(4), 1165-1188.

[33] Dhar, V. (2013). Data Science and Prediction. Communications of the ACM, 56(12), 64-73.

[34] Davenport, T. H., Kirby, J. (2018). Only Humans Need Apply: Winners and Losers in the Age of Smart Machines. Harper Business.

[35] Russell, S., Norvig, P. (2016). Artificial Intelligence: A Modern Approach. Pearson.

[36] Domingos, P. (2015). The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World. Basic Books.

[37] Marcus, G., Davis, E. (2018). Rebooting AI: Building Artificial Intelligence We Can Trust. Pantheon.

[38] Thompson, C. (2012). Remote Work Revolution: Succeeding from Anywhere. Harvard Business Review Press.

[39] Ford, M. (2020). Architects of Intelligence: The Truth About AI from the People Building It. Packt Publishing.

[40] Redecker, C., Punie, Y. (2012). The Future of Learning 2020: New Ways to Learn New Skills for Future Jobs. Institute for Prospective Technological Studies.

[41] Bashshur, R. L., Shannon, G. W., Krupinski, E. A. (2016). The Empirical Foundations of Telemedicine Interventions in Primary Care. Telemedicine and e-Health, 22(5), 342-375.

[42] Collins, A., Halverson, R. (2020). Rethinking Education in the Age of Technology: The Digital Revolution and Schooling in America. Teachers College Press.

[43] Van Rossum, G., Drake, F. L. (2009). Python 3 Reference Manual. CreateSpace.

[44] Flanagan, D. (2020). JavaScript: The Definitive Guide. O'Reilly Media.

[45] Fette, I., Melnikov, A. (2011). The WebSocket Protocol. IETF RFC 6455.

[46] Miguel Grinberg, "Flask-SocketIO Documentation," [Online]. `https://flask-socketio.readthedocs.io/en/latest/`.

[47] Duck DNS, "Duck DNS - Free Dynamic DNS," [Online]. `https://www.duckdns.org/`.

[48] React ,A JavaScript library for building user interfaces `https://legacy.reactjs.org/docs/hooks-intro.html`

[49] Beazley, D. M., Jones, B. K. (2013). *Python Cookbook* (3rd ed.)

[50] Harris, C. R., Millman, K. J., van der Walt, S. J., et al. (2020). Array programming with NumPy. *Nature*, 585(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2.

[51] van der Walt, S., Colbert, S. C., Varoquaux, G. (2011). The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science & Engineering*, 13(2), 22–30. https://doi.org/10.1109/MCSE.2011.37..

[52] Watanabe, P. (2021). *Effective TypeScript: 62 Specific Ways to Improve Your TypeScript*

[53] Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*.

[54] The NumPy Developers. (n.d.). *NumPy v1.22 Manual*. Retrieved from https://numpy.org/-doc/stable/.

[55] The ROOT Team. (n.d.). *ROOT User's Guide*. Retrieved from https://root.cern.ch.

[56] The PYROOT Team. (n.d.). *PyROOT: A Python Extension for ROOT*. Retrieved from https://root.cern/manual/python/.

[57] The Plotly Team. (n.d.). *Plotly JavaScript Open Source Graphing Library*. Retrieved from https://plotly.com/javascript/.

[58] The Ant Design Team. (n.d.). *Ant Design - A UI Design Language*. Retrieved from https://ant.design/.

6 Appendix

6.1 Server-side Code

Server-side main

```
import threading
import logging
from flask import Flask, jsonify, render_template
from flask_cors import CORS
from flask_socketio import SocketIO
import numpy as np
from Histogram import Histogram
from FitHistogram import FitHistogram
from Distribution_Analysis import DistributionAnalysis
import warnings

# Somewhere in your Flask application initialization code
warnings.filterwarnings("ignore", message="Fit data is empty", category=
    RuntimeWarning)


app = Flask(__name__)
CORS(app)
socketio = SocketIO(app, cors_allowed_origins="*")

# Set the logging level for eventlet (assuming you are using eventlet)
log = logging.getLogger('werkzeug')
log.setLevel(logging.ERROR)

# Define the histogram
histogram = Histogram('hist', 'continuous histogram', 200, -5, 5)

# Threading event for controlling the processing thread
stop_event = threading.Event()


# Function to continuously fill the histogram with random data
def continuous_histogram_filling():
    while not stop_event.is_set():
        # Generate random data points (you can replace this with your own
            generator)
        random_data_1 = np.random.normal(loc=-1, scale=1, size=50)
        random_data_2 = np.random.normal(loc=1, scale=1, size=50)

        random_data = np.concatenate((random_data_1, random_data_2))

        # random_data = np.random.normal(loc=0, scale=1, size=100)

        # Fill the histogram with new data
        histogram.fill(random_data)

        x, y = histogram.get_histogram_data()
        y_max = histogram.get_y_max()

        # find distribution
```

```
        distribution_obj = DistributionAnalysis(histogram)
        distribution = distribution_obj.find_distribution()

        distribution_type = {'gaus': 'Gauss', 'landau': 'Landau',
                             '[0] / ( (x - [1])**2 + 0.25 * [2]**2 )': 'Breit Wigner',
                             "[0]*TMath::Gaus(x, [1], [2]) + [3]*TMath::Gaus(x, [4], [5])": 'Double Gauss'}

        fit_results = {}

        for key in list(distribution_type.keys()):
            if key != "[0]*TMath::Gaus(x, [1], [2]) + [3]*TMath::Gaus(x, [4], [5])":
                generate_fits(fit_results, histogram, key)

        # if distribution is found to be a double gaussian then repeat the above process but for a double gaussian
        if distribution == "[0]*TMath::Gaus(x, [1], [2]) + [3]*TMath::Gaus(x, [4], [5])":
            double_gauss_amplitude, double_gauss_amplitude_error, double_gauss_mean, double_gauss_mean_error, \
                double_gauss_sigma, double_gauss_sigma_error, double_gauss_amplitude2, double_gauss_amplitude_error2, \
                double_gauss_mean2, double_gauss_mean_error2, double_gauss_sigma2, double_gauss_sigma_error2, \
                double_gauss_ndf, double_gauss_chi2, y_fit_double_gaussian, \
                y_fit_gauss1, y_fit_gauss2 = generate_double_fit()
        else:
            double_gauss_amplitude, double_gauss_amplitude_error, double_gauss_mean, double_gauss_mean_error, \
                double_gauss_sigma, double_gauss_sigma_error, double_gauss_amplitude2, double_gauss_amplitude_error2, \
                double_gauss_mean2, double_gauss_mean_error2, double_gauss_sigma2, double_gauss_sigma_error2, \
                double_gauss_ndf, double_gauss_chi2, y_fit_double_gaussian, y_fit_gauss1, \
                y_fit_gauss2 = 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, [], [], []

        if distribution in distribution_type:
            distribution = distribution_type[distribution]
        socketio.emit('update_histogram',
                      {'x': x, 'y': y, 'y_fit_gauss': fit_results['y_fit_gaus'],
                       'gauss_amplitude': fit_results['gaus_amplitude'],
                       'gauss_amplitude_error': fit_results['gaus_amplitude_error'],
                       'gauss_mean': fit_results['gaus_mean'],
                       'gauss_mean_error': fit_results['gaus_mean_error'],
                       'gauss_sigma': fit_results['gaus_sigma'],
                       'gauss_sigma_error': fit_results['gaus_sigma_error'],
```

```
                        'gauss_ndf': fit_results['gaus_ndf'],
                        'gauss_chi2': fit_results['gaus_chi2'], "y_fit_landau": fit_results['y_fit_landau'],
                        "landau_amplitude": fit_results['landau_amplitude'],
                        "landau_amplitude_error": fit_results['landau_amplitude_error'],
                        "landau_mean": fit_results['landau_mean'],
                        "landau_mean_error": fit_results['landau_mean_error'],
                        "landau_sigma": fit_results['landau_sigma'],
                        "landau_sigma_error": fit_results['landau_sigma_error'],
                        "landau_ndf": fit_results['landau_ndf'],
                        "landau_chi2": fit_results['landau_chi2'],
                        "y_fit_bw": fit_results['y_fit_[0] / ( (x - [1])**2 + 0.25 * [2]**2 )'],
                        "bw_amplitude": fit_results['[0] / ( (x - [1])**2 + 0.25 * [2]**2 )_amplitude'],
                        "bw_amplitude_error": fit_results['[0] / ( (x - [1])**2 + 0.25 * [2]**2 )_amplitude_error'],
                        "bw_mean": fit_results['[0] / ( (x - [1])**2 + 0.25 * [2]**2 )_mean'],
                        "bw_mean_error": fit_results['[0] / ( (x - [1])**2 + 0.25 * [2]**2 )_mean_error'],
                        "bw_sigma": fit_results['[0] / ( (x - [1])**2 + 0.25 * [2]**2 )_sigma'],
                        "bw_sigma_error": fit_results['[0] / ( (x - [1])**2 + 0.25 * [2]**2 )_sigma_error'],
                        "bw_ndf": fit_results['[0] / ( (x - [1])**2 + 0.25 * [2]**2 )_ndf'],
                        "bw_chi2": fit_results['[0] / ( (x - [1])**2 + 0.25 * [2]**2 )_chi2'],
                        "y_fit_double_gaussian": y_fit_double_gaussian, "double_gauss_amplitude": double_gauss_amplitude,
                        "double_gauss_amplitude_error": double_gauss_amplitude_error,
                        "double_gauss_mean": double_gauss_mean, "double_gauss_mean_error": double_gauss_mean_error,
                        "double_gauss_sigma": double_gauss_sigma, "double_gauss_sigma_error": double_gauss_sigma_error,
                        "double_gauss_amplitude2": double_gauss_amplitude2,
                        "double_gauss_amplitude_error2": double_gauss_amplitude_error2,
                        "double_gauss_mean2": double_gauss_mean2, "double_gauss_mean_error2": double_gauss_mean_error2,
                        "double_gauss_sigma2": double_gauss_sigma2,
                        "double_gauss_sigma_error2": double_gauss_sigma_error2, "double_gauss_ndf": double_gauss_ndf,
                        "double_gauss_chi2": double_gauss_chi2, "y_fit_gauss1": y_fit_gauss1,
                        "y_fit_gauss2": y_fit_gauss2,
                        'distribution_type': distribution, 'max_y_for_hist':
```

```
                                    y_max})

        # stop_event.wait(0.1)100


def generate_fits(fit_results, histogram, key):
    fit = FitHistogram(histogram, key)
    fit_results[f'{key}_amplitude'], fit_results[f'{key}_amplitude_error'],
         fit_results[f'{key}_mean'], fit_results[
        f'{key}_mean_error'], \
        fit_results[f'{key}_sigma'], fit_results[f'{key}_sigma_error'],
            fit_results[f'{key}_ndf'], fit_results[
        f'{key}_chi2'] = fit.get_fit_parameters()
    fit_results[f'y_fit_{key}'] = fit.get_y_fit()


def generate_double_fit():
    original_histogram_tf1_double_gauss = FitHistogram(histogram,
                                                        "[0]*TMath::Gaus(x,
                                                            [1], [2]) + [3]*
                                                            TMath::Gaus(x,
                                                            [4], [5])")
    original_histogram_tf1_double_gauss.set_params_for_double_gaussian()
    double_gauss_amplitude, double_gauss_amplitude_error, double_gauss_mean
        , double_gauss_mean_error, \
        double_gauss_sigma, double_gauss_sigma_error,
            double_gauss_amplitude2, double_gauss_amplitude_error2, \
        double_gauss_mean2, double_gauss_mean_error2, double_gauss_sigma2,
            double_gauss_sigma_error2, \
        double_gauss_ndf, double_gauss_chi2 =
            original_histogram_tf1_double_gauss.get_double_fit_parameters()
    y_fit_double_gaussian = original_histogram_tf1_double_gauss.get_y_fit()

    histogram_of_first_gaussian = Histogram('gauss1', 'gauss1', 200, -5, 5)
    histogram_of_second_gaussian = Histogram('gauss2', 'gauss2', 200, -5,
        5)

    # Get histogram data from the original histogram
    original_histogram_data = histogram.get_data()

    # Split the data into two halves
    half_data_points = len(original_histogram_data) // 2

    histogram_of_first_gaussian_tf1_gauss = FitHistogram(
        histogram_of_first_gaussian, 'gaus')
    histogram_of_first_gaussian_tf1_gauss.set_params_for_gaussian(
        double_gauss_amplitude, double_gauss_mean,
                                                            double_gauss_sigma
                                                                )
    histogram_of_first_gaussian.hist.FillRandom("gaus", half_data_points)
    y_fit_gauss1 = histogram_of_first_gaussian_tf1_gauss.get_y_fit()

    histogram_of_second_gaussian_tf1_gauss = FitHistogram(
        histogram_of_second_gaussian, 'gaus')
    histogram_of_second_gaussian_tf1_gauss.set_params_for_gaussian(
        double_gauss_amplitude2, double_gauss_mean2,
```

```python
                                                                                double_gauss_sigma2
                                                                                   )
    histogram_of_second_gaussian.hist.FillRandom("gaus", half_data_points)
    y_fit_gauss2 = histogram_of_second_gaussian_tf1_gauss.get_y_fit()

    return double_gauss_amplitude, double_gauss_amplitude_error,
       double_gauss_mean, double_gauss_mean_error, \
        double_gauss_sigma, double_gauss_sigma_error,
           double_gauss_amplitude2, double_gauss_amplitude_error2, \
        double_gauss_mean2, double_gauss_mean_error2, double_gauss_sigma2,
           double_gauss_sigma_error2, \
        double_gauss_ndf, double_gauss_chi2, y_fit_double_gaussian,
           y_fit_gauss1, y_fit_gauss2


# Route to serve the HTML page with JavaScript for WebSocket communication
@app.route('/')
def index():
    return render_template('index.html')


@socketio.on('client_command')
def handle_client_command(data):
    command = data.get('command')
    if command == 'stop':
        # Set the threading event to stop the processing thread
        stop_event.set()
    elif command == 'restart':
        # Clear the threading event to restart the processing thread
        stop_event.clear()
        # Restart the processing thread
        new_thread = threading.Thread(target=continuous_histogram_filling)
        new_thread.start()


@socketio.on('clear_data')
def clear_data(data):
    command = data.get('command')
    if command == 'Clear':
        histogram.hist.Reset()


processing_thread = threading.Thread(target=continuous_histogram_filling)
processing_thread.start()
# Run the Flask app with SocketIO
if __name__ == '__main__':
    socketio.run(app, host='0.0.0.0', port=49152)
```

Listing 1: Server-side main

Histogram class

```python
import ROOT
import numpy as np

"""
The Histogram class represents a histogram and provides methods for manipulating and analyzing it.
"""


class Histogram(ROOT.TH1F):
    """
           Constructor for the Histogram class.

           Parameters:
           - name: A string representing the name of the histogram.
           - title: A string representing the title of the histogram.
           - n_bins: An integer specifying the number of bins in the histogram.
           - x_min: The minimum value for the x-axis range of the histogram.
           - x_max: The maximum value for the x-axis range of the histogram.

           This constructor initializes the Histogram object by setting the histogram name, title, number of bins,
           and x-axis range. It creates a TH1F histogram object using the provided parameters, and initializes
           other attributes such as data and histogram reconstruction.
           """

    def __init__(self, name, title, n_bins, x_min, x_max):
        super().__init__(name, title, n_bins, x_min, x_max)
        ROOT.TH1.AddDirectory(False)
        ROOT.gROOT.SetBatch(True)
        self.name = name
        self.title = title
        self.n_bins = n_bins
        self.x_min = x_min
        self.x_max = x_max
        self.hist = ROOT.TH1F(self.name, self.title, self.n_bins, self.x_min, self.x_max)
        self.hist.SetDirectory(0)
        self.data = []

    def reconstruct_hist(self):
        """
        Reconstructs the histogram object.

        This method recreates the histogram object using the parameters provided during initialization.
        """
        self.hist = ROOT.TH1F(self.name, self.title, self.n_bins, self.x_min, self.x_max)

```

```python
    def fill(self, data):
        """
        Fill the histogram with data.

        Parameters:
        - data: A list of floating-point numbers representing the data points to fill the histogram with.

        This method fills the histogram with the given data points. It appends the data points to the data
        attribute and fills the histogram object using the TH1F.Fill() method.
        """
        for d in data:
            self.data.append(d)
            self.hist.Fill(d)

    def get_data(self):
        """
        Get the data points used to fill the histogram.

        Returns:
        - A list of floating-point numbers representing the data points used to fill the histogram.

        This method returns the data points that were used to fill the histogram.
        """
        return self.data

    def get_y_max(self):
        """
        Get the maximum y-axis value of the histogram.

        Returns:
        - The maximum value of the y-axis for the histogram.

        This method returns the maximum value of the y-axis for the histogram using the TH1F.GetMaximum() method.
        """
        return self.hist.GetMaximum()

    def scale_hist(self, this_y_max, other_y_max):
        """
        Scale the histogram based on maximum y-axis values.

        Parameters:
        - this_y_max: The maximum y-axis value of the current histogram.
        - other_y_max: The maximum y-axis value of the other histogram.

        Returns:
        - The scaled histogram object.

        This method scales the current histogram based on the maximum y-axis values of the current histogram
        and another histogram. It scales the histogram using the TH1F.Scale
```

```
        () method and returns the scaled
        histogram object.
        """
        return self.hist.Scale(other_y_max / this_y_max)

    def add(self, other_hist):
        """
        Add another histogram to the current histogram.

        Parameters:
        - other_hist: Another Histogram object to be added to the current histogram.

        This method adds another histogram to the current histogram using the TH1F.Add() method.
        """
        self.hist.Add(other_hist.hist)

    @staticmethod
    def sum_histograms(hist1, hist2):
        """
        Static method to sum two histograms.

        Parameters:
        - hist1: The first Histogram object.
        - hist2: The second Histogram object.

        Returns:
        - A new Histogram object representing the sum of hist1 and hist2.

        This static method creates a new histogram with the same binning as hist1 and hist2. It fills
        the new histogram with the sum of the bin contents from hist1 and hist2, and returns the new
        histogram object.
        """
        # Create a new histogram with the same binning as hist1 and hist2
        hist_sum = ROOT.TH1F(hist1.GetName() + "_plus_" + hist2.GetName(), hist1.GetTitle() + " + " + hist2.GetTitle(),
                             hist1.GetNbinsX(), hist1.GetXaxis().GetXmin(), hist2.GetXaxis().GetXmax())

        # Fill the new histogram with the sum of data from hist1 and hist2
        for i in range(1, hist_sum.GetNbinsX() + 1):
            bin_content = hist1.GetBinContent(i) + hist2.GetBinContent(i)
            hist_sum.SetBinContent(i, bin_content)

        return hist_sum

    def get_sum_histogram_data(self, other_hist):
        """
        Get the x and y values of the sum of two histograms.

        Parameters:
        - other_hist: Another Histogram object.

```

```
        Returns:
        - Two lists containing the x-axis values and y-axis values of the sum of the two histograms.

        This method gets the sum of two histograms by adding their bin contents. It returns the x-axis values
        and y-axis values of the sum histogram as two separate lists.
        """
        # Get the sum of the two histograms
        sum_hist = self.hist.Clone()
        sum_hist.Add(other_hist.hist)

        # Get the x and y values of the sum histogram
        x = np.array([sum_hist.GetBinCenter(i) for i in range(1, sum_hist.GetNbinsX() + 1)])
        y = np.array([sum_hist.GetBinContent(i) for i in range(1, sum_hist.GetNbinsX() + 1)])
        return x.tolist(), y.tolist()

    def get_histogram_data(self):
        """
        Get the x and y values of the histogram.

        Returns:
        - Two lists containing the x-axis values and y-axis values of the histogram.

        This method gets the x-axis values and y-axis values of the histogram as two separate lists.
        """
        x = np.array([self.hist.GetBinCenter(i) for i in range(1, self.hist.GetNbinsX() + 1)])
        y = np.array([self.hist.GetBinContent(i) for i in range(1, self.hist.GetNbinsX() + 1)])
        return x.tolist(), y.tolist()
```

Listing 2: Histogram class

Fit Histogram Class

```python
import ROOT
import numpy as np

"""
The FitHistogram class represents a histogram fitting object and provides methods for fitting and retrieving fit parameters.
"""


class FitHistogram:
    _instances = {}

    def __new__(cls, histogram, distribution):
        # Check if an instance with the given histogram and distribution already exists
        instance_key = f"{id(histogram)}_{distribution}"
        if instance_key not in cls._instances:
            instance = super(FitHistogram, cls).__new__(cls)
            cls._instances[instance_key] = instance
            instance.hist_obj = histogram
            instance.hist = histogram.hist
            instance.distribution = distribution
            instance.fit_func = ROOT.TF1("f1", instance.distribution, instance.hist_obj.x_min, instance.hist_obj.x_max)
            instance.hist.Fit(instance.fit_func, "Q")
        else:
            # If the instance exists, update the fit with the new histogram
            cls._instances[instance_key].update_fit(histogram)
        return cls._instances[instance_key]

    def update_fit(self, new_histogram):
        """
        Update the fit with a new histogram.

        Parameters:
        - new_histogram: The new histogram to update the fit.
        """
        self.hist_obj = new_histogram
        self.hist = new_histogram.hist
        self.fit_func.SetRange(new_histogram.x_min, new_histogram.x_max)
        self.hist.Fit(self.fit_func, "Q")

    def get_fit_parameters(self):
        """
       Get the fit parameters of the histogram.

       Returns:
       - Tuple containing the fit parameters of the histogram: amplitude, amplitude error, mean, mean error,
         sigma, sigma error, NDF (number of degrees of freedom), and chi-square.

       This method retrieves the fit parameters of the histogram, including the amplitude, mean, sigma,
```

```
        amplitude error, mean error, sigma error, NDF, and chi-square values, and returns them as a tuple.
        """
        amplitude = self.fit_func.GetParameter(0)
        amplitude_error = self.fit_func.GetParError(0)
        mean = self.fit_func.GetParameter(1)
        mean_error = self.fit_func.GetParError(1)
        sigma = self.fit_func.GetParameter(2)
        sigma_error = self.fit_func.GetParError(2)
        ndf = self.fit_func.GetNDF()
        chi2 = self.fit_func.GetChisquare()
        return amplitude, amplitude_error, mean, mean_error, sigma, sigma_error, ndf, chi2

    def get_double_fit_parameters(self):
        """
        Get the fit parameters of the double Gaussian fit.

        Returns:
        - Tuple containing the fit parameters of the double Gaussian fit: amplitude, amplitude error,
          mean, mean error, sigma, sigma error, amplitude2, amplitude2 error, mean2, mean2 error,
          sigma2, sigma2 error, NDF (number of degrees of freedom), and chi-square.

        This method retrieves the fit parameters of the double Gaussian fit, including the parameters of
        the first and second Gaussian components. It returns these parameters as a tuple.
        """
        amplitude = self.fit_func.GetParameter(0)
        mean = self.fit_func.GetParameter(1)
        sigma = self.fit_func.GetParameter(2)
        amplitude2 = self.fit_func.GetParameter(3)
        mean2 = self.fit_func.GetParameter(4)
        sigma2 = self.fit_func.GetParameter(5)
        amplitude_error = self.fit_func.GetParError(0)
        mean_error = self.fit_func.GetParError(1)
        sigma_error = self.fit_func.GetParError(2)
        amplitude2_error = self.fit_func.GetParError(3)
        mean2_error = self.fit_func.GetParError(4)
        sigma2_error = self.fit_func.GetParError(5)
        ndf = self.fit_func.GetNDF()
        chi2 = self.fit_func.GetChisquare()
        return amplitude, amplitude_error, mean, mean_error, sigma, sigma_error, amplitude2, amplitude2_error, mean2, \
            mean2_error, sigma2, sigma2_error, ndf, chi2

    def get_y_fit(self):
        """
        Get the y-axis values of the fitted curve.

        Returns:
        - A list of y-axis values representing the fitted curve.

```

```
        This method generates the y-axis values of the fitted curve by evaluating the fit function at each
        x-axis value of the histogram bins. It returns the y-axis values as a list.
        """
        fit_hist = ROOT.TH1F("fit_hist", "Fit", self.hist_obj.n_bins, self.hist_obj.x_min,
                             self.hist_obj.x_max)
        for i in range(1, fit_hist.GetNbinsX() + 1):
            x = fit_hist.GetBinCenter(i)
            y = self.fit_func.Eval(x)
            fit_hist.SetBinContent(i, y)
        fit_hist.SetDirectory(0)
        y_fit = np.array([fit_hist.GetBinContent(i) for i in range(1, fit_hist.GetNbinsX() + 1)])
        return y_fit.tolist()

    def set_params_for_gaussian(self, amplitude, mean, sigma):
        """
        Set the fit parameters for a Gaussian distribution.

        Parameters:
        - amplitude: The amplitude parameter for the Gaussian distribution.
        - mean: The mean parameter for the Gaussian distribution.
        - sigma: The sigma parameter for the Gaussian distribution.

        This method sets the fit parameters for a Gaussian distribution by setting the corresponding parameters
        in the fit function. It then performs the fitting using the updated parameters.
        """
        self.fit_func.SetParameters(amplitude, mean, sigma)
        self.hist.Fit(self.fit_func, "Q")

    def set_params_for_double_gaussian(self):
        """
        Set the fit parameters for a double Gaussian distribution.

        This method sets the fit parameters for a double Gaussian distribution by setting the corresponding parameters
        in the fit function. It initializes the parameters using the histogram properties (maximum, mean, and standard
        deviation). It then performs the fitting using the updated parameters.
        """
        self.fit_func.SetParameters(self.hist.GetMaximum() / 2, self.hist.GetMean() - self.hist.GetStdDev(),
                                    self.hist.GetStdDev() / 2,
                                    self.hist.GetMaximum() / 2, self.hist.GetMean() + self.hist.GetStdDev(),
                                    self.hist.GetStdDev() / 2)
        self.hist.Fit(self.fit_func, "Q")
```

Listing 3: Fit Histogram Class

Distribution analysis class

```python
import numpy as np
from FitHistogram import FitHistogram

"""
This class, DistributionFinder, is responsible for finding the best-fitting distribution function
for a given histogram data. It performs fits using different distribution functions and compares
goodness-of-fit statistics to determine the most suitable distribution function for the data.
"""


class DistributionAnalysis:
    def __init__(self, hist):
        """
        Constructor for the DistributionFinder class.

        Parameters:
        - hist: An instance of the Histogram class representing a histogram.

        This constructor initializes the DistributionFinder object by setting the histogram object,
        accessing the histogram data, and defining a list of distribution functions to be used for fitting.
        """
        self.hist_obj = hist
        self.hist = hist.hist

        self.distributions = [
            'gaus',  # Gaussian
            'landau',  # Landau
            '[0] / ( (x - [1])**2 + 0.25 * [2]**2 )',  # Breit-Wigner
            "[0]*TMath::Gaus(x, [1], [2]) + [3]*TMath::Gaus(x, [4], [5])",  # Double Gaussian
        ]

    def find_distribution(self):
        """
        Find the best-fitting distribution function for the histogram data.

        Returns:
        - Tuple containing the best-fitting distribution function and an array of goodness-of-fit statistics.

        This method fits the histogram data to each distribution function in self.distributions and compares
        the goodness-of-fit statistics. It returns the best-fitting distribution function and an array of
        goodness-of-fit statistics.
        """
        result_array = np.array([])

        # Fit the data to each distribution and compare goodness-of-fit
```

```python
            statistics
        for dist in self.distributions:
            f1 = FitHistogram(self.hist_obj, dist)
            func = f1.fit_func
            if dist == "[0]*TMath::Gaus(x, [1], [2]) + [3]*TMath::Gaus(x, [4], [5])":
                f1.set_params_for_double_gaussian()

            if func.GetNDF() != 0:
                chi2_ndf_ratio = func.GetChisquare() / func.GetNDF()
            result_array = np.append(result_array, chi2_ndf_ratio)

            idx = (np.abs(result_array - 1)).argmin()
            # if self.distributions[idx] in self.distributions_type:
            #     self.distributions[idx] = self.distributions_type[self.distributions[idx]]
        if result_array[0] and abs(result_array[0] - result_array[-1]) <= abs(result_array[-1] * 0.1):
            idx = 0

        return self.distributions[idx]
```

Listing 4: Distribution analysis class

Update duckdns script

```python
import requests
import time

# Your Duck DNS domain and token
domain = "dqm.duckdns.org"
token = "cb55d3f7-d0ac-431d-bea2-07b7bc22b153"

# Initialize a variable to store the current IP address
current_ip = None

while True:
    try:
        # Get your current public IP
        response = requests.get("https://api.ipify.org?format=json")
        new_ip = response.text.strip()

        # Check if the IP has changed
        if new_ip != current_ip:
            print(f"IP has changed to {new_ip}. Updating Duck DNS...")
            # Send a request to Duck DNS API to update the IP
            response = requests.get(f"https://www.duckdns.org/update?domains={domain}&token={token}")
            print(response.text)

            # Update the current IP
            current_ip = new_ip
        else:
            print('IP have not changed')

        # Sleep for 20 minutes
        time.sleep(1200)

    except Exception as e:
        print(f"An error occurred: {e}")
```

Listing 5: Update duckdns script

6.2 Client-side Code

6.2.1 Main render and initialization

Index file

```
import ReactDOM from "react-dom/client";
import "./index.css";
import App from "./App";

const root = ReactDOM.createRoot(
  document.getElementById("root") as HTMLElement
);
root.render(<App />);
```

Listing 6: Index file

Index style

```
body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', '
      Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  background-color: white;
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}
```

Listing 7: Index style

Main app handler

```
import React, { useState } from "react";
import HistogramData from "./components/histogramData";
import styles from "./AppStyles.module.css";
import Home from "./components/home";
import { Switch } from "antd";
import style from "./components/histogramData.module.css";
import { Tabs } from "antd";
import { adjustColors } from "./Shared/helperFunctions";

function App() {
  const [darkMode, setDarkMode] = useState<boolean>(false);
  const items = [];

  const toggleMode = () => {
    setDarkMode(!darkMode);
  };

  adjustColors(darkMode);

  items.push(
    {
      key: "1",
      label: "Home",
      children: <Home />,
    },
    {
      key: "2",
      label: "HistogramData",
      children: <HistogramData darkMode={darkMode} />,
    }
  );

  return (
    <>
      <div className={styles.nav}>
        <Tabs
          type="card"
          items={items}
          tabBarExtraContent={
            <Switch
              defaultChecked
              checked={darkMode}
              checkedChildren="Enable light mode"
              unCheckedChildren="Enable dark mode"
              onChange={toggleMode}
              className={`${style.ant_btn} ${style.darkmode}`}
            />
          }
        />
      </div>
    </>
  );
}
```

```
export default App;
```

Listing 8: Main app handler

Main app handler styles

```
.nav{
    margin-top: 2rem;
    /* display:flex;
    align-items: center; */
}

:global(.ant-tabs-tab){
    color:var(--color) !important;;
    background-color: var(--background-color) !important;
    transition: all 2s ease !important;
}
```

Listing 9: Main app handler styles

Global module declarator

```
declare module "*.module.css";
declare module "*.module.scss";
```

Listing 10: Global module declarator

Plotly declarator

```
declare module 'react-plotly.js';
```

Listing 11: Plotly declarator

6.2.2 Main components

Histogram Data page

```
import io from "socket.io-client";
import React, { useState, useRef, useEffect } from "react";
import Plot from "react-plotly.js";
import style from "./histogramData.module.css";
import "./histogramData.css";

import {
  DistributionsBooleans,
  DistributionsBooleansKey,
  HistogramDataTypes,
  HistogramShowBooleans,
  HistogramTitles,
  Layout,
} from "../Types/Types";
import { RenderSuccess, RenderWarning } from "../Warnings/Warnings";
import { WarningMessageForServer } from "../Warnings/warningMessages";
import ButtonBarForHistogram from "../Utils/buttonBarForHistogram";
import { histogramLayout } from "./histogramLayout";
import { defaultPlotValues, enableFit } from "../Shared/helperFunctions";

//Main component of the application. Responsible for the histogram tab. Handles and structures
//the data responsible for the histogram graph.

interface IProps {
  darkMode: boolean;
}

const HistogramData = (props: IProps) => {
  // State variables
  const [plotData, setPlotData] = useState<HistogramDataTypes>({
    // Initialize the plot data with empty arrays and default values
    ...defaultPlotValues,
  });
  const [titles, setTitles] = useState<HistogramTitles>({
    // Default names for x,y axis and title
    xAxisTitle: "Values",
    yAxisTitle: "Entries",
    histogramTitle: `${plotData.distribution_type} Histogram`,
  });
  const tempTitle = useRef<string>("");
  const [histogramShowBooleans, setHistogramShowBooleans] =
    //state responsible for all the booleans regarding micro changes in the UI
    useState<HistogramShowBooleans>({
      showResults: false,
      showFit: false,
      logScale: false,
      enableFit: false,
      isModalOpen: false,
      showMaxEntries: false,
      fetchData: true,
```

```
      loading: true,
    });
  const [distributionsBooleans, setDistributionsBooleans] =
    //state responsible for rendering the data of the corresponding
       distribution
    useState<DistributionsBooleans>({
      gauss: false,
      landau: false,
      bw: false,
      double_gaussian: false,
      gauss1: false,
      gauss2: false,
    });
  const [layout, setLayout] = useState<Layout>({
    //standard layout values
    title: "",
    xaxis: { title: "", range: [0, 0] },
    yaxis: { title: "", range: [0, 0] },
    modebar: { orientation: "" },
    font: { family: "", size: 0 },
    width: 0,
    height: 0,
    legend: {
      x: 0,
      y: 0,
      traceorder: "",
      font: { family: "", size: 0, color: "" },
    },
    annotations: [],
  });

  const renderMessage = useRef(true);
  const [screenWidth, setScreenWidth] = useState<number>(1920);
  const [screenHeight, setScreenHeight] = useState<number>(1080);
  const currentIp = useRef(null);

  let dataForPlot: unknown = [];
  let resizeTimer: NodeJS.Timeout;
  let fetchIpForTheFirstTime: boolean = true;
  // let renderMessage: boolean = true;

  const get_default_fit_for_key = (key: DistributionsBooleansKey) => {
    //depending on the distibutions clears the customData,
    //enable fit and trigger the corresponding boolean for rendering
    enableFit(histogramShowBooleans, setHistogramShowBooleans);
    setDistributionsBooleans({
      ...distributionsBooleans,
      [key]: !distributionsBooleans[key],
    });
  };

  const handleResize = () => {
    //Adjusts plot size depending on the users window size
    clearTimeout(resizeTimer);
    resizeTimer = setTimeout(() => {
      setScreenWidth(
```

```
        window.innerWidth ||
          document.documentElement.clientWidth ||
          document.body.clientWidth
      );
      setScreenHeight(
        window.innerHeight ||
          document.documentElement.clientHeight ||
          document.body.clientHeight
      );
    }, 100);
  };

  window.addEventListener("resize", handleResize);

  const getCurrentIp = async () => {
    const response = await fetch("https://api.ipify.org?format=json");
    const result = await response.json();
    currentIp.current = result.ip;
  };

  // useEffect(() => {
  //   setTitles({
  //     ...titles,
  //     histogramTitle: tempTitle.current,
  //   });
  // }, [tempTitle.current]);

  useEffect(() => {
    handleResize();
    const intervalId = setInterval(getCurrentIp, 60 * 20 * 1000);
  }, []);

  useEffect(() => {
    fetchIp().then(() => {
      try {
        setHistogramShowBooleans({ ...histogramShowBooleans, loading: false
            });
        // Establish WebSocket connection
        const socket = io(`http://127.0.0.1:49152`);
        // Listen for updates from the server
        socket.on("update_histogram", (data) => {
          setPlotData({
            ...plotData,
            x: data.x,
            y: data.y,
            distribution_type: data.distribution_type,
            y_fit_gauss: data.y_fit_gauss,
            y_fit_landau: data.y_fit_landau,
            y_fit_bw: data.y_fit_bw,
            y_fit_double_gaussian: data.y_fit_double_gaussian,
            y_fit_gauss1: data.y_fit_gauss1,
            y_fit_gauss2: data.y_fit_gauss2,
            gauss_amplitude: data.gauss_amplitude,
            gauss_amplitudeError: data.gauss_amplitude_error,
            gauss_mean: data.gauss_mean,
            gauss_meanError: data.gauss_mean_error,
```

```
            gauss_sigma: data.gauss_sigma,
            gauss_sigmaError: data.gauss_sigma_error,
            gauss_ndf: data.gauss_ndf,
            gauss_chi2: data.gauss_chi2,
            landau_amplitude: data.landau_amplitude,
            landau_amplitudeError: data.landau_amplitude_error,
            landau_mean: data.landau_mean,
            landau_meanError: data.landau_mean_error,
            landau_sigma: data.landau_sigma,
            landau_sigmaError: data.landau_sigma_error,
            landau_ndf: data.landau_ndf,
            landau_chi2: data.landau_chi2,
            bw_amplitude: data.bw_amplitude,
            bw_amplitudeError: data.bw_amplitude_error,
            bw_mean: data.bw_mean,
            bw_meanError: data.bw_mean_error,
            bw_sigma: data.bw_sigma,
            bw_sigmaError: data.bw_sigma_error,
            bw_ndf: data.bw_ndf,
            bw_chi2: data.bw_chi2,
            max_y_for_hist: data.max_y_for_hist,
            double_gauss_amplitude: data.double_gauss_amplitude,
            double_gauss_amplitudeError: data.double_gauss_amplitude_error,
            double_gauss_mean: data.double_gauss_mean,
            double_gauss_meanError: data.double_gauss_mean_error,
            double_gauss_sigma: data.double_gauss_sigma,
            double_gauss_sigmaError: data.double_gauss_sigma_error,
            double_gauss_amplitude2: data.double_gauss_amplitude2,
            double_gauss_amplitudeError2: data.
               double_gauss_amplitude_error2,
            double_gauss_mean2: data.double_gauss_mean2,
            double_gauss_meanError2: data.double_gauss_mean_error2,
            double_gauss_sigma2: data.double_gauss_sigma2,
            double_gauss_sigmaError2: data.double_gauss_sigma_error2,
            double_gauss_ndf: data.double_gauss_ndf,
            double_gauss_chi2: data.double_gauss_chi2,
          });
          tempTitle.current = `${data.distribution_type} Histogram`;
          console.log(`${data.distribution_type} Histogram`);
          if (renderMessage.current) {
            setTitles({
              ...titles,
              histogramTitle: `${data.distribution_type} Histogram`,
            });
            RenderSuccess(
              `Loaded ${data.distribution_type} distribution`,
              true
            );
            renderMessage.current = false;
          }
        });
        // Clean up the socket connection when component unmounts

        return () => {
          socket.disconnect();
        };
```

```
      } catch {
        RenderWarning(WarningMessageForServer);
      }
    });
  }, [renderMessage.current]);

  useEffect(() => {
    // if all distribution booleans are false, disable the show fit button
    if (
      !distributionsBooleans.gauss &&
      !distributionsBooleans.landau &&
      !distributionsBooleans.bw &&
      !distributionsBooleans.double_gaussian &&
      !distributionsBooleans.gauss1 &&
      !distributionsBooleans.gauss2
    ) {
      setHistogramShowBooleans({ ...histogramShowBooleans, enableFit: false
          });
    }
  }, [distributionsBooleans]);

  const fetchIp = async () => {
    return;
    if (fetchIpForTheFirstTime) {
      await getCurrentIp();
      fetchIpForTheFirstTime = false;
    }
  };

  useEffect(() => {
    //if something of the dependency array changes then make the changes in
        the layout format
    setLayout(
      histogramLayout(
        histogramShowBooleans.showFit,
        props.darkMode,
        distributionsBooleans,
        histogramShowBooleans,
        plotData,
        layout,
        titles,
        screenWidth,
        screenHeight
      )
    );
  }, [
    plotData,
    histogramShowBooleans.logScale,
    histogramShowBooleans.showMaxEntries,
    titles,
    distributionsBooleans,
    screenWidth,
    screenHeight,
    histogramShowBooleans.showFit,
    props.darkMode,
  ]);
```

```

  if (histogramShowBooleans.showFit) {
    dataForPlot = [
      {
        x: plotData.x,
        y: plotData.y,
        type: "bar",
        name: `${plotData.distribution_type} Histogram`,
      },
      distributionsBooleans.gauss
        ? {
            x: plotData.x,
            y: plotData.y_fit_gauss,
            type: "scatter",
            mode: "lines",
            name: "Gauss Fit",
            line: {
              color: "D95319",
            },
          }
        : {},
      distributionsBooleans.landau
        ? {
            x: plotData.x,
            y: plotData.y_fit_landau,
            type: "scatter",
            mode: "lines",
            name: "Landau Fit",
            line: {
              color: "EDB120",
            },
          }
        : {},
      distributionsBooleans.bw
        ? {
            x: plotData.x,
            y: plotData.y_fit_bw,
            type: "scatter",
            mode: "lines",
            name: "Breit-Wigner Fit",
            line: {
              color: "7E2F8E",
            },
          }
        : {},
      distributionsBooleans.double_gaussian
        ? {
            x: plotData.x,
            y: plotData.y_fit_double_gaussian,
            type: "scatter",
            mode: "lines",
            name: "Double Gaussian Fit",
            line: {
              color: "77AC30",
            },
          }
```

```
          : {},
        distributionsBooleans.gauss1
          ? {
              x: plotData.x,
              y: plotData.y_fit_gauss1,
              type: "scatter",
              mode: "lines",
              name: "First Gaussian Fit",
              line: {
                color: "4DBEEE",
              },
            }
          : {},
        distributionsBooleans.gauss2
          ? {
              x: plotData.x,
              y: plotData.y_fit_gauss2,
              type: "scatter",
              mode: "lines",
              name: "Second Gaussian Fit",
              line: {
                color: "A2142F",
              },
            }
          : {},
      ];
    } else {
      dataForPlot = [
        {
          x: plotData.x,
          y: plotData.y,
          type: "bar",
          name: `${plotData.distribution_type} Histogram`,
        },
      ];
    }
    const config = {
      displayModeBar: false,
    };

    return (
      <div className={style.histogram_container}>
        <Plot
          data={dataForPlot}
          layout={layout}
          config={config}
          className={style.plot_container}
        />
        <ButtonBarForHistogram
          plotData={plotData}
          setPlotData={setPlotData}
          histogramShowBooleans={histogramShowBooleans}
          setHistogramShowBooleans={setHistogramShowBooleans}
          distributionsBooleans={distributionsBooleans}
          setDistributionsBooleans={setDistributionsBooleans}
          titles={titles}
```

```
        setTitles={setTitles}
        get_default_fit_for_key={get_default_fit_for_key}
        layout={layout}
        setLayout={setLayout}
        darkMode={props.darkMode}
        currentIp={currentIp}
        renderMessage={renderMessage}
      />
    </div>
  );
};

export default HistogramData;
```

Listing 12: Histogram Data component

Histogram Data page styles

```
.histogram_container{
    justify-content: center;
    text-align: center;
}

.darkmode{
    display:flex;
    margin-left: auto;
    justify-content: flex-end;

}

.spinner{
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    display: flex;
    flex-direction: column;
    align-items: center;
    font-size: 30px;
}

.fetchingMessage{
    color: var(--color) !important;
}
```

Listing 13: Histogram Data component scss styles

```
.modebar-btn {
    width:auto;
    bottom: -4rem;
    left:-5rem
}

.icon {
    font-size: 1.5rem;
}

```

```
.g-xaxis .g-title {
    pointer-events: none;
  }
```

Listing 14: Histogram Data css styles

Histogram Layout component

```
import {
  DistributionsBooleans,
  HistogramDataTypes,
  HistogramShowBooleans,
  Layout,
} from "../Types/Types";
import { histogramGetAllLabels } from "./histogramLabel";

export const histogramLayout = (
  showFit: boolean,
  darkMode: boolean,
  distributionsBooleans: DistributionsBooleans,
  histogramShowBooleans: HistogramShowBooleans,
  plotData: HistogramDataTypes,
  layout: Layout,
  titles: {
    xAxisTitle: string;
    yAxisTitle: string;
    histogramTitle: string;
  },
  screenWidth: number,
  screenHeight: number
) => {
  const [
    fitGaussLabel,
    fitLandauLabel,
    fitBwLabel,
    fitDoubleGaussLabel,
    fitGauss1Label,
    fitGauss2Label,
  ] = histogramGetAllLabels(plotData);
  let totalEntries = 0;
  for (let i = 0; i < plotData.y.length; i++) {
    totalEntries = totalEntries + plotData.y[i];
  }

  const annotationForGauss = distributionsBooleans.gauss
    ? {
        x: 0.98,
        y: 0.59,
        xref: "paper",
        yref: "paper",
        text: `${fitGaussLabel}`,
        align: "left",
        showarrow: false,
        font: {
          size: 4 + (screenWidth * 0.0005 + screenHeight * 0.02) / 2,
          color: darkMode ? "FFFFFF" : "#000000",
        },
        bgcolor: "rgba(217,83,25,0.02)",
        borderpad: 4,
        bordercolor: "#D95319",
        borderwidth: 2,
        width: screenWidth * 0.12,
```

```
        opacity: 0.8,
      }
    : { showarrow: false, text: "" };
  const annotationForLandau = distributionsBooleans.landau
    ? {
        x: 0.98,
        y: 0.41,
        xref: "paper",
        yref: "paper",
        text: `${fitLandauLabel}`,
        align: "left",
        showarrow: false,
        font: {
          size: 4 + (screenWidth * 0.0005 + screenHeight * 0.02) / 2,
          color: darkMode ? "FFFFFF" : "#000000",
        },
        bgcolor: "rgba(237,177,32,0.02)",
        borderpad: 4,
        bordercolor: "#EDB120",
        borderwidth: 2,
        width: screenWidth * 0.12,
        opacity: 0.8,
      }
    : { showarrow: false, text: "" };

  const annotationForBw = distributionsBooleans.bw
    ? {
        x: 0.98,
        y: 0.15,
        xref: "paper",
        yref: "paper",
        text: `${fitBwLabel}`,
        align: "left",
        showarrow: false,
        font: {
          size: 4 + (screenWidth * 0.0005 + screenHeight * 0.02) / 2,
          color: darkMode ? "FFFFFF" : "#000000",
        },
        bgcolor: "rgba(126,47,142,0.02)",
        borderpad: 4,
        bordercolor: "#7E2F8E",
        borderwidth: 2,
        width: screenWidth * 0.12,
        opacity: 0.8,
      }
    : { showarrow: false, text: "" };

  const annotationForDoubleGauss = distributionsBooleans.double_gaussian
    ? {
        x: 0.02,
        y: 0.73,
        xref: "paper",
        yref: "paper",
        text: `${fitDoubleGaussLabel}`,
        align: "left",
        showarrow: false,
```

```
        font: {
          size: 4 + (screenWidth * 0.0005 + screenHeight * 0.02) / 2,
          color: darkMode ? "FFFFFF" : "#000000",
        },
        bgcolor: "rgba(119,172,48, 0.02)",
        borderpad: 4,
        bordercolor: "#77AC30",
        borderwidth: 2,
        width: screenWidth * 0.12,
        opacity: 0.8,
      }
    : { showarrow: false, text: "" };

  const annotationForGauss1 = distributionsBooleans.gauss1
    ? {
        x: 0.02,
        y: 0.4,
        xref: "paper",
        yref: "paper",
        text: `${fitGauss1Label}`,
        align: "left",
        showarrow: false,
        font: {
          size: 4 + (screenWidth * 0.0005 + screenHeight * 0.02) / 2,
          color: darkMode ? "FFFFFF" : "#000000",
        },
        bgcolor: "rgba(77,190,238, 0.02)",
        borderpad: 4,
        bordercolor: "#4DBEEE",
        borderwidth: 2,
        width: screenWidth * 0.12,
        opacity: 0.8,
      }
    : { showarrow: false, text: "" };

  const annotationForGauss2 = distributionsBooleans.gauss2
    ? {
        x: 0.02,
        y: 0.2,
        xref: "paper",
        yref: "paper",
        text: `${fitGauss2Label}`,
        align: "left",
        showarrow: false,
        font: {
          size: 4 + (screenWidth * 0.0005 + screenHeight * 0.02) / 2,
          color: darkMode ? "FFFFFF" : "#000000",
        },
        bgcolor: "rgba(162,20,47,0.02)",
        borderpad: 4,
        bordercolor: "#A2142F",
        borderwidth: 2,
        width: screenWidth * 0.12,
        opacity: 0.8,
      }
    : { showarrow: false, text: "" };
```

```

const maxEntries = layout.annotations.filter((annotation) =>
  annotation.text?.includes("Max entries")
);

const annotationForMaxEntries = histogramShowBooleans.showMaxEntries
  ? {
      x: plotData.x[plotData.y.indexOf(plotData.max_y_for_hist)],
      y: histogramShowBooleans.logScale
        ? Math.log10(plotData.max_y_for_hist)
        : plotData.max_y_for_hist,
      xref: "x",
      yref: "y",
      text: darkMode
        ? `<span style='color:white'>Max entries: ${
            plotData.max_y_for_hist
          }<br>At value: ${plotData.x[
            plotData.y.indexOf(plotData.max_y_for_hist)
          ].toFixed(4)}</span>`
        : `<span style='color:black'>Max entries: ${
            plotData.max_y_for_hist
          }<br>At value: ${plotData.x[
            plotData.y.indexOf(plotData.max_y_for_hist)
          ].toFixed(4)}</span>`,
      align: "left",
      showarrow: true,
      arrowhead: 14,
      arrowcolor: darkMode ? "white" : "black",
      ax: -100,
      ay: 0,
      font: { size: 16, color: "#000000" },
    }
  : { showarrow: false, text: "" };

const defaultAnnotations = {
  x: 0.95,
  y: 0.988,
  xref: "paper",
  yref: "paper",
  text: darkMode
    ? `<span style='color:white;'>Total entries: ${totalEntries}</span> `
    : `<span style='color:black;'>Total entries: ${totalEntries}</span> `,
  align: "left",
  showarrow: false,
  font: {
    size: 20,
    color: darkMode ? "FFFFFF" : "#000000",
  },
};

const layoutReturn = {
  title: titles.histogramTitle,
  xaxis: {
    title: titles.xAxisTitle,
    range: [-5, 5],
```

```
    },
    yaxis: histogramShowBooleans.logScale
      ? {
          type: "log",
          range: [0, plotData.max_y_for_hist.toString().length],
          title: titles.yAxisTitle,
          gridcolor: darkMode ? "rgba(255,255,255,0.2)" : undefined,
        }
      : {
          title: titles.yAxisTitle,
          gridcolor: darkMode ? "rgba(255,255,255,0.2)" : undefined,
          range: [0, plotData.max_y_for_hist * 1.4],
        },
    modebar: {
      orientation: "v",
    },
    font: {
      family: "Arial, sans-serif",
      size: 16,
      color: darkMode ? "white" : "black",
    },
    plot_bgcolor: "rgba (0,0,0,0)", // Set the plot background color
    paper_bgcolor: "rgba (0,0,0,0)",
    width: screenWidth * 0.8,
    height: screenHeight * 0.8,
    legend: {
      x: 0.8,
      y: 0.95,
      traceorder: "normal",
      font: {
        family: "sans-serif",
        size: 16,
        color: darkMode ? "FFFFFF" : "#000000",
      },
    },
    annotations:
      showFit &&
      (distributionsBooleans.gauss ||
        distributionsBooleans.landau ||
        distributionsBooleans.bw ||
        distributionsBooleans.double_gaussian ||
        distributionsBooleans.gauss1 ||
        distributionsBooleans.gauss2)
        ? [
            annotationForMaxEntries,
            annotationForGauss,
            annotationForLandau,
            annotationForBw,
            annotationForDoubleGauss,
            defaultAnnotations,
            annotationForGauss1,
            annotationForGauss2,
          ]
        : [annotationForMaxEntries, defaultAnnotations],
  };
  return layoutReturn;
```

```
};
```

Listing 15: Histogram Layout component

Histogram Label Component

```
import { HistogramDataTypes } from "../Types/Types";

export const histogramGetAllLabels = (plotData: HistogramDataTypes) => {
  const fitGaussLabel = `Amplitude: ${plotData.gauss_amplitude.toFixed(
    4
  )} ± ${plotData.gauss_amplitudeError.toFixed(
    4
  )}<br>Mean: ${plotData.gauss_mean.toFixed(
    4
  )} ± ${plotData.gauss_meanError.toFixed(
    4
  )}<br>Sigma: ${plotData.gauss_sigma.toFixed(
    4
  )} ± ${plotData.gauss_sigmaError.toFixed(
    4
  )}<br>Chi2/ndf: ${plotData.gauss_chi2.toFixed(
    4
  )}/ ${plotData.gauss_ndf.toFixed(4)}`;

  const fitLandauLabel = `Amplitude: ${plotData.landau_amplitude.toFixed(
    4
  )} ± ${plotData.landau_amplitudeError.toFixed(
    4
  )}<br>Mean: ${plotData.landau_mean.toFixed(
    4
  )} ± ${plotData.landau_meanError.toFixed(
    4
  )}<br>Sigma: ${plotData.landau_sigma.toFixed(
    4
  )} ± ${plotData.landau_sigmaError.toFixed(
    4
  )}<br>Chi2/ndf: ${plotData.landau_chi2.toFixed(
    4
  )}/ ${plotData.landau_ndf.toFixed(4)}`;

  const fitBwLabel = `Amplitude: ${plotData.bw_amplitude.toFixed(
    4
  )} ± ${plotData.bw_amplitudeError.toFixed(
    4
  )}<br>Mean: ${plotData.bw_mean.toFixed(4)} ± ${plotData.bw_meanError.
     toFixed(
    4
  )}<br>Sigma: ${plotData.bw_sigma.toFixed(
    4
  )} ± ${plotData.bw_sigmaError.toFixed(
    4
  )}<br>Chi2/ndf: ${plotData.bw_chi2.toFixed(4)}/ ${plotData.bw_ndf.toFixed
     (
    4
  )}`;

  const fitDoubleGaussianLabel = `Amplitude: ${plotData.
     double_gauss_amplitude.toFixed(
    4
```

```
  )} ± ${plotData.double_gauss_amplitudeError.toFixed(
    4
  )}<br>Mean: ${plotData.double_gauss_mean.toFixed(
    4
  )} ± ${plotData.double_gauss_meanError.toFixed(
    4
  )}<br>Sigma: ${plotData.double_gauss_sigma.toFixed(
    4
  )} ± ${plotData.double_gauss_sigmaError.toFixed(
    4
  )}<br>Amplitude2: ${plotData.double_gauss_amplitude2.toFixed(
    4
  )} ± ${plotData.double_gauss_amplitudeError2.toFixed(
    4
  )}<br>Mean2: ${plotData.double_gauss_mean2.toFixed(
    4
  )} ± ${plotData.double_gauss_meanError2.toFixed(
    4
  )}<br>Sigma2: ${plotData.double_gauss_sigma2.toFixed(
    4
  )} ± ${plotData.double_gauss_sigmaError2.toFixed(
    4
  )}<br>Chi2/ndf: ${plotData.double_gauss_chi2.toFixed(
    4
  )}/ ${plotData.double_gauss_ndf.toFixed(4)}`;

  const fitGauss1Label = `Amplitude: ${plotData.double_gauss_amplitude.toFixed(
    4
  )} ± ${plotData.double_gauss_amplitudeError.toFixed(
    4
  )}<br>Mean: ${plotData.double_gauss_mean.toFixed(
    4
  )} ± ${plotData.double_gauss_meanError.toFixed(
    4
  )}<br>Sigma: ${plotData.double_gauss_sigma.toFixed(
    4
  )} ± ${plotData.double_gauss_sigmaError.toFixed(4)}`;

  const fitGauss2Label = `Amplitude: ${plotData.double_gauss_amplitude2.toFixed(
    4
  )} ± ${plotData.double_gauss_amplitudeError2.toFixed(
    4
  )}<br>Mean: ${plotData.double_gauss_mean2.toFixed(
    4
  )} ± ${plotData.double_gauss_meanError2.toFixed(
    4
  )}<br>Sigma: ${plotData.double_gauss_sigma2.toFixed(
    4
  )} ± ${plotData.double_gauss_sigmaError2.toFixed(4)}`;

  return [
    fitGaussLabel,
    fitLandauLabel,
    fitBwLabel,
```

```
    fitDoubleGaussianLabel,
    fitGauss1Label,
    fitGauss2Label,
  ];
};
```

Listing 16: Histogram Label Component

Home Page

```
import React from "react";
import "./HomePage.css";

const HomePage: React.FC = () => {
  return (
    <div className="home-page">
      <div className="section">
        <h1>Π    Μ      Σ     Σ    Η      Τ


        </h1>
      </div>
      <div className="section">
        <h2Τ  > Φ   </h2>
        <h2Σ  > Θ   Ε      </h2>
        <h2Π      > Ι     </h2>
      </div>
      <div className="section">
        <h3ΜΕΤΑΠΤΥΧΙΑΚΗ> ΔΙΠΛΩΜΑΤΙΚΗΕΡΓΑΣΙΑΜΔΕ  (..)</h3>
      </div>
      <div className="section">
        <hr className="line" />
        <h4>
          Design and Implementation of a Zynq-Based Data Quality Monitoring
          System Remotely Accessible
        </h4>
        <hr className="line" />
      </div>
      <div className="section">
        <h5Κ    > Π   </h5>
      </div>
    </div>
  );
};

export default HomePage;
```

Listing 17: Home Page

Home Page Styles

```
:root {
    --background-color: white;
    --color: black;
  }

  body {
    background-color: var(--background-color);
    color: var(--color);
  }

  .home-page {
    text-align: center;
    padding: 20px;
  }

  .section {
    margin-bottom: 20px;
  }

  h1, h2, h3, h4, h5 {
    color: var(--color);
  }

  h1 {
    font-size: 2.5em;
    font-weight: bold;
  }

  h2 {
    font-size: 1.75em;
    font-weight: bold;
  }

  h3 {
    font-size: 1.5em;
    font-weight: bold;
    margin: 30px 0;
  }

  h4 {
    font-size: 1.25em;
    font-weight: bold;
  }

  h5 {
    font-size: 1em;
    font-weight: bold;
  }

  .line {
    width: 100%;
    border: 1px solid var(--color);
  }
```

Listing 18: Home Page Styles

6.2.3 Shared functionalities

General Helper Functions

```
import {
  HistogramEditBooleans,
  HistogramShowBooleans,
  HistogramTitles,
} from "../Types/Types";

export const generateArrayInRange = (
  arrayLength: number,
  minValue: number,
  maxValue: number
) => {
  return Array.from(
    { length: arrayLength },
    (_, i) => minValue + ((maxValue - minValue) * i) / (arrayLength - 1)
  );
};

export const enableFit = (
  // enables the fit button
  histogramShowBooleans: HistogramShowBooleans,
  setHistogramShowBooleans: (
    value: React.SetStateAction<HistogramShowBooleans>
  ) => void
) => {
  setHistogramShowBooleans({
    ...histogramShowBooleans,
    enableFit: true,
  });
};

export const handleInputBlur = (
  setInputValue: (value: React.SetStateAction<string>) => void,
  setEdits: (value: React.SetStateAction<HistogramEditBooleans>) => void,
  edits: HistogramEditBooleans,
  key: string
) => {
  setInputValue("");
  setEdits({ ...edits, [key]: false });
};

export const handleInputSubmit = (
  setInputValue: (value: React.SetStateAction<string>) => void,
  setTitles: (value: React.SetStateAction<HistogramTitles>) => void,
  setEdits: (value: React.SetStateAction<HistogramEditBooleans>) => void,
  inputValue: string,
  titles: HistogramTitles,
  edits: HistogramEditBooleans,
  editKey: string
) => {
  setEdits({ ...edits, [editKey]: false });
  setTitles({
    ...titles,
```

```
    [editKey.replace(/^edit(.)(.*)$/, (_, firstLetter, restOfWord) =>
      firstLetter === "H"
        ? firstLetter.toLowerCase() + restOfWord
        : firstLetter.toLowerCase() +
          restOfWord.charAt(0).toUpperCase() +
          restOfWord.slice(1)
    )]: inputValue,
  });
  setInputValue("");
};

export const adjustColors = (darkMode: boolean) => {
  if (darkMode) {
    document.body.style.backgroundColor = "#2B2A33";
    document.documentElement.style.setProperty("--background-color", "#2B2A33");
    document.documentElement.style.setProperty("--color", "white");
  } else {
    document.body.style.backgroundColor = "white";
    document.documentElement.style.setProperty("--background-color", "white");
    document.documentElement.style.setProperty("--color", "black");
  }
};

export const defaultPlotValues = {
  x: [],
  y: [],
  y_fit_gauss: [],
  y_fit_landau: [],
  y_fit_bw: [],
  y_fit_double_gaussian: [],
  y_fit_gauss1: [],
  y_fit_gauss2: [],
  gauss_amplitude: 0,
  gauss_amplitudeError: 0,
  gauss_mean: 0,
  gauss_meanError: 0,
  gauss_sigma: 0,
  gauss_sigmaError: 0,
  gauss_ndf: 0,
  gauss_chi2: 0,
  landau_amplitude: 0,
  landau_amplitudeError: 0,
  landau_mean: 0,
  landau_meanError: 0,
  landau_sigma: 0,
  landau_sigmaError: 0,
  landau_ndf: 0,
  landau_chi2: 0,
  bw_amplitude: 0,
  bw_amplitudeError: 0,
  bw_mean: 0,
  bw_meanError: 0,
  bw_sigma: 0,
  bw_sigmaError: 0,
```

```
  bw_ndf: 0,
  bw_chi2: 0,
  double_gauss_amplitude: 0,
  double_gauss_amplitudeError: 0,
  double_gauss_mean: 0,
  double_gauss_meanError: 0,
  double_gauss_sigma: 0,
  double_gauss_sigmaError: 0,
  double_gauss_amplitude2: 0,
  double_gauss_amplitudeError2: 0,
  double_gauss_mean2: 0,
  double_gauss_meanError2: 0,
  double_gauss_sigma2: 0,
  double_gauss_sigmaError2: 0,
  double_gauss_ndf: 0,
  double_gauss_chi2: 0,
  limit_for_axis: 0,
  max_y_for_hist: 0,
  distribution_type: "",
};
```

Listing 19: General Helper Functions

Info Tools Function

```
import { Button, Tooltip } from "antd";
import { InfoCircleOutlined } from "@ant-design/icons";
import style from "../Utils/buttonBarForHistogram.module.css";

export const InfoToolForHistogram = ({ message }: { message: string }) => {
  return (
    <Tooltip title={message} className={style.ant_btn}>
      <Button>
        <InfoCircleOutlined />
      </Button>
    </Tooltip>
  );
};
```

Listing 20: Info Tools Function

Info Tools Text

```
export const InfoToolHistogramText = 'This is a histogram generated from
   PYROOT using data from a random number generator. The standard number of
    bins used are 200. All the calculations regarding the histogram are
   done in PYROOT, this page is only responsible for rendering the final
   results'
```

Listing 21: Info Tools Text

ODS Export Function

```
import * as XLSX from "xlsx";
import { HistogramDataTypes, HistogramTitles } from "../Types/Types";
import { SuccsesMessageForODSExport } from "../Warnings/warningMessages";
import { RenderSuccess } from "../Warnings/Warnings";

const ODSExport = (plotData: HistogramDataTypes, titles: HistogramTitles)
    => {
  //structures the data in a way that can be exported to an ods file
  const plotDataKeys = Object.keys(plotData);
  const titlesKeys = Object.keys(titles);
  let header = [];
  header.push(...plotDataKeys);
  header.push(...titlesKeys);
  const isItDoubleGaussian = plotData.y_fit_double_gaussian.length !== 0;
  if (!isItDoubleGaussian) {
    const valuesToRemove = [
      "y_fit_double_gaussian",
      "y_fit_gauss1",
      "y_fit_gauss2",
      "double_gauss_amplitude",
      "double_gauss_amplitudeError",
      "double_gauss_mean",
      "double_gauss_meanError",
      "double_gauss_sigma",
      "double_gauss_sigmaError",
      "double_gauss_amplitude2",
      "double_gauss_amplitudeError2",
      "double_gauss_mean2",
      "double_gauss_meanError2",
      "double_gauss_sigma2",
      "double_gauss_sigmaError2",
      "double_gauss_ndf",
      "double_gauss_chi2",
    ];
    header = header.filter((value) => !valuesToRemove.includes(value));
  }
  const data = [
    header,
    ...plotData?.x.map((_, i) => {
      return [
        plotData.x[i],
        plotData.y[i],
        plotData.y_fit_gauss[i],
        plotData.y_fit_landau[i],
        plotData.y_fit_bw[i],
        isItDoubleGaussian ? plotData.y_fit_double_gaussian[i] : "remove",
        isItDoubleGaussian ? plotData.y_fit_gauss1[i] : "remove",
        isItDoubleGaussian ? plotData.y_fit_gauss2[i] : "remove",
        i === 0 ? plotData.gauss_amplitude : "",
        i === 0 ? plotData.gauss_amplitudeError : "",
        i === 0 ? plotData.gauss_mean : "",
        i === 0 ? plotData.gauss_meanError : "",
        i === 0 ? plotData.gauss_sigma : "",
        i === 0 ? plotData.gauss_sigmaError : "",
```

```
        i === 0 ? plotData.gauss_ndf : "",
        i === 0 ? plotData.gauss_chi2 : "",
        i === 0 ? plotData.landau_amplitude : "",
        i === 0 ? plotData.landau_amplitudeError : "",
        i === 0 ? plotData.landau_mean : "",
        i === 0 ? plotData.landau_meanError : "",
        i === 0 ? plotData.landau_sigma : "",
        i === 0 ? plotData.landau_sigmaError : "",
        i === 0 ? plotData.landau_ndf : "",
        i === 0 ? plotData.landau_chi2 : "",
        i === 0 ? plotData.bw_amplitude : "",
        i === 0 ? plotData.bw_amplitudeError : "",
        i === 0 ? plotData.bw_mean : "",
        i === 0 ? plotData.bw_meanError : "",
        i === 0 ? plotData.bw_sigma : "",
        i === 0 ? plotData.bw_sigmaError : "",
        i === 0 ? plotData.bw_ndf : "",
        i === 0 ? plotData.bw_chi2 : "",
        isItDoubleGaussian
          ? i === 0
            ? plotData.double_gauss_amplitude
            : ""
          : "remove",
        isItDoubleGaussian
          ? i === 0
            ? plotData.double_gauss_amplitudeError
            : ""
          : "remove",
        isItDoubleGaussian
          ? i === 0
            ? plotData.double_gauss_mean
            : ""
          : "remove",
        isItDoubleGaussian
          ? i === 0
            ? plotData.double_gauss_meanError
            : ""
          : "remove",
        isItDoubleGaussian
          ? i === 0
            ? plotData.double_gauss_sigma
            : ""
          : "remove",
        isItDoubleGaussian
          ? i === 0
            ? plotData.double_gauss_sigmaError
            : ""
          : "remove",
        isItDoubleGaussian
          ? i === 0
            ? plotData.double_gauss_amplitude2
            : ""
          : "remove",
        isItDoubleGaussian
          ? i === 0
            ? plotData.double_gauss_amplitudeError2
```

```
              : ""
            : "remove",
          isItDoubleGaussian
            ? i === 0
              ? plotData.double_gauss_mean2
              : ""
            : "remove",
          isItDoubleGaussian
            ? i === 0
              ? plotData.double_gauss_meanError2
              : ""
            : "remove",
          isItDoubleGaussian
            ? i === 0
              ? plotData.double_gauss_sigma2
              : ""
            : "remove",
          isItDoubleGaussian
            ? i === 0
              ? plotData.double_gauss_sigmaError2
              : ""
            : "remove",
          isItDoubleGaussian
            ? i === 0
              ? plotData.double_gauss_ndf
              : ""
            : "remove",
          isItDoubleGaussian
            ? i === 0
              ? plotData.double_gauss_chi2
              : ""
            : "remove",
          i === 0 ? plotData.limit_for_axis : "",
          i === 0 ? plotData.max_y_for_hist : "",
          i === 0 ? plotData.distribution_type : "",
          i === 0 ? titles.xAxisTitle : "",
          i === 0 ? titles.yAxisTitle : "",
          i === 0 ? titles.histogramTitle : "",
        ].filter((value) => value !== "remove");
      }),
  ];
  const worksheetName = "Sheet1";
  const wb = XLSX.utils.book_new();
  const worksheet = XLSX.utils.aoa_to_sheet(data);
  XLSX.utils.book_append_sheet(wb, worksheet, worksheetName);
  XLSX.writeFile(wb, "output.ods");
  RenderSuccess(SuccsesMessageForODSExport, true);
};

export default ODSExport;
```

Listing 22: ODS Export Function

ODS Import Function

```
import { Button, Upload } from "antd";
import * as XLSX from "xlsx";
import { HistogramDataTypes, HistogramTitles } from "../Types/Types";
import { UploadOutlined } from "@ant-design/icons";
import { RenderSuccess, RenderWarning } from "../Warnings/Warnings";
import React from "react";
import style from "../Utils/buttonBarForHistogram.module.css";
import { defaultPlotValues } from "./helperFunctions";

interface IProps {
  plotData: HistogramDataTypes;
  setPlotData: React.Dispatch<React.SetStateAction<HistogramDataTypes>>;
  setTitles: React.Dispatch<React.SetStateAction<HistogramTitles>>;
}

const OdsImport = React.memo((props: IProps) => {
  //parser of an ods that searches for specific values and throws error if
  //some specific fields are missing
  const readFile = (file: File) => {
    const reader = new FileReader();
    reader.onload = (event) => {
      const data = new Uint8Array(event.target?.result as ArrayBuffer);
      const workbook = XLSX.read(data, { type: "array" });
      const worksheet = workbook.Sheets[workbook.SheetNames[0]];
      const aoa = XLSX.utils.sheet_to_json(worksheet, {
        header: 1,
        blankrows: false,
      }) as [string[], ...any[][]];
      const [headers, ...plotDataValues] = aoa;
      const plotDataKeys = Object.keys(props.plotData).filter(
        (header) =>
          !header.includes("double") &&
          !header.includes("y_fit_gauss1") &&
          !header.includes("y_fit_gauss2")
      );
      const plotDataKeysPresent = plotDataKeys.every((key) =>
        headers.includes(key)
      );
      if (!plotDataKeysPresent) {
        const missingKeys = plotDataKeys.filter(
          (key) => !headers.includes(key)
        );
        return RenderWarning(
          `${missingKeys}: column are missing from your ods file`
        );
      }
      let temp_plotData: HistogramDataTypes = {
        ...defaultPlotValues,
      };
      let temp_titles: HistogramTitles = {
        xAxisTitle: "",
        yAxisTitle: "",
        histogramTitle: "",
      };
```

```
      for (let i = 0; i < headers.length - 3; i++) {
        if (headers[i] === "x") {
          temp_plotData = {
            ...temp_plotData,
            x: plotDataValues.map((innerArray) => innerArray[0]),
          };
        } else if (headers[i] === "y") {
          temp_plotData = {
            ...temp_plotData,
            y: plotDataValues.map((innerArray) => innerArray[1]),
          };
        } else if (headers[i] === "y_fit_gauss") {
          temp_plotData = {
            ...temp_plotData,
            y_fit_gauss: plotDataValues.map((innerArray) => innerArray[2]),
          };
        } else if (headers[i] === "y_fit_landau") {
          temp_plotData = {
            ...temp_plotData,
            y_fit_landau: plotDataValues.map((innerArray) => innerArray[3]),
          };
        } else if (headers[i] === "y_fit_bw") {
          temp_plotData = {
            ...temp_plotData,
            y_fit_bw: plotDataValues.map((innerArray) => innerArray[4]),
          };
        } else if (headers[i] === "y_fit_double_gaussian") {
          temp_plotData = {
            ...temp_plotData,
            y_fit_double_gaussian: plotDataValues.map(
              (innerArray) => innerArray[5]
            ),
          };
        } else if (headers[i] === "y_fit_gauss1") {
          temp_plotData = {
            ...temp_plotData,
            y_fit_gauss1: plotDataValues.map((innerArray) => innerArray[6]),
          };
        } else if (headers[i] === "y_fit_gauss2") {
          temp_plotData = {
            ...temp_plotData,
            y_fit_gauss2: plotDataValues.map((innerArray) => innerArray[7]),
          };
        } else {
          temp_plotData = {
            ...temp_plotData,
            [headers[i]]: plotDataValues[0][i],
          };
        }
      }
      for (let i = headers.length - 3; i < headers.length; i++) {
        if (headers[i]) {
          temp_titles = { ...temp_titles, [headers[i]]: plotDataValues[0][i
```

```
          ] };
        } else {
          temp_titles = { ...temp_titles, [headers[i]]: "" };
        }
      }
      props.setPlotData(temp_plotData);
      props.setTitles(temp_titles);
      return RenderSuccess("Data has been successfully imported", true);
    };
    reader.readAsArrayBuffer(file);
  };

  return (
    <Upload
      accept=".ods"
      showUploadList={false}
      beforeUpload={(file) => {
        readFile(file);
        return false; // prevent upload
      }}
    >
      <Button className={style.ant_btn} icon={<UploadOutlined />}>
        Import ODS File
      </Button>
    </Upload>
  );
});

export default OdsImport;
```

Listing 23: ODS Import Function

6.2.4 Util functions

Button Bar For Histogram

```
import { Button, Dropdown, Menu, Tooltip, Switch, Checkbox } from "antd";
import { DownOutlined, ExportOutlined, EditOutlined } from "@ant-design/
    icons";
import { InfoToolForHistogram } from "../Shared/infoTools";
import { InfoToolHistogramText } from "../Shared/infoToolsText";
import ODSExport from "../Shared/odsExport";
import {
  DistributionsBooleans,
  DistributionsBooleansKey,
  HistogramDataTypes,
  HistogramEditBooleans,
  HistogramShowBooleans,
  HistogramTitles,
  Layout,
} from "../Types/Types";
import { useEffect, useRef, useState } from "react";
import FindTitleToEdit from "./findTitleToEdit";
import {
  SucessMessageForBwFit,
  SucessMessageForDefaultFit,
  SucessMessageForDoubleGauss,
  SucessMessageForDoubleGaussChild1,
  SucessMessageForDoubleGaussChild2,
  SucessMessageForLandauFit,
} from "../Warnings/warningMessages";
import { RenderSuccess } from "../Warnings/Warnings";
import style from "../Utils/buttonBarForHistogram.module.css";
import OdsImport from "../Shared/odsImport";
import { io, Socket } from "socket.io-client";

interface IProps {
  histogramShowBooleans: HistogramShowBooleans;
  setHistogramShowBooleans: React.Dispatch<
    React.SetStateAction<HistogramShowBooleans>
  >;
  distributionsBooleans: DistributionsBooleans;
  setDistributionsBooleans: React.Dispatch<
    React.SetStateAction<DistributionsBooleans>
  >;
  plotData: HistogramDataTypes;
  setPlotData: React.Dispatch<React.SetStateAction<HistogramDataTypes>>;
  titles: HistogramTitles;
  setTitles: React.Dispatch<React.SetStateAction<HistogramTitles>>;
  get_default_fit_for_key: (key: DistributionsBooleansKey) => void;
  layout: Layout;
  setLayout: React.Dispatch<React.SetStateAction<Layout>>;
  darkMode: boolean;
  currentIp: React.MutableRefObject<null>;
  renderMessage: React.MutableRefObject<boolean>;
}

//Component that contains all the buttons below the histogram graph
```

```

const ButtonBarForHistogram = (props: IProps) => {
  const [edits, setEdits] = useState<HistogramEditBooleans>({
    editXaxisTitle: false,
    editYaxisTitle: false,
    editHistogramTitle: false,
  });
  const socketRef = useRef<Socket | null>(null);

  useEffect(() => {
    // Create a new socket instance
    const socket = io(`http://127.0.0.1:49152`);

    // Store the socket instance in the ref
    socketRef.current = socket;

    // Clean up the socket connection when the component unmounts
    return () => {
      if (socketRef.current) {
        socketRef.current.disconnect();
      }
    };
  }, []);

  const clearData = () => {
    props.renderMessage.current = true;
    props.setPlotData({
      ...props.plotData,
      x: [],
      y: [],
      y_fit_gauss: [],
      y_fit_landau: [],
      y_fit_double_gaussian: [],
      gauss_amplitude: 0,
      gauss_amplitudeError: 0,
      gauss_mean: 0,
      gauss_meanError: 0,
      gauss_sigma: 0,
      gauss_sigmaError: 0,
      gauss_ndf: 0,
      gauss_chi2: 0,
      landau_amplitude: 0,
      landau_amplitudeError: 0,
      landau_mean: 0,
      landau_meanError: 0,
      landau_sigma: 0,
      landau_sigmaError: 0,
      landau_ndf: 0,
      landau_chi2: 0,
      bw_amplitude: 0,
      bw_amplitudeError: 0,
      bw_mean: 0,
      bw_meanError: 0,
      bw_sigma: 0,
      bw_sigmaError: 0,
      bw_ndf: 0,
```

```
      bw_chi2: 0,
      limit_for_axis: 0,
      max_y_for_hist: 0,
    });
    props.setTitles({
      xAxisTitle: "Values",
      yAxisTitle: "Entries",
      histogramTitle: "",
    });
    props.setDistributionsBooleans({
      gauss: false,
      landau: false,
      bw: false,
      double_gaussian: false,
      gauss1: false,
      gauss2: false,
    });
    props.setHistogramShowBooleans({
      ...props.histogramShowBooleans,
      showResults: false,
      showFit: false,
      logScale: false,
      enableFit: false,
      isModalOpen: false,
      showMaxEntries: false,
    });
    if (socketRef.current) {
      socketRef.current.emit("clear_data", { command: "Clear" });
    }
  };

  const editXaxis = () => {
    setEdits({ ...edits, editXaxisTitle: true });
  };

  const editYaxis = () => {
    setEdits({ ...edits, editYaxisTitle: true });
  };

  const editTitle = () => {
    setEdits({ ...edits, editHistogramTitle: true });
  };

  const toggleSwitchChange = (checked: boolean) => {
    if (!socketRef.current) return;

    props.setHistogramShowBooleans({
      ...props.histogramShowBooleans,
      fetchData: checked,
    });

    if (checked) {
      socketRef.current.emit("client_command", { command: "restart" });
    } else {
      socketRef.current.emit("client_command", { command: "stop" });
    }
```

```
};

const toggleFit = () => {
  props.setHistogramShowBooleans({
    ...props.histogramShowBooleans,
    showFit: !props.histogramShowBooleans.showFit,
  });
};

const toggleLogScale = () => {
  props.setHistogramShowBooleans({
    ...props.histogramShowBooleans,
    logScale: !props.histogramShowBooleans.logScale,
  });
};

const handleInnerButtonClick = (event: any) => {
  event.stopPropagation();
};

const menuForDoubleGaussian = (
  <Menu className={style.ant_btn}>
    <Menu.Item key="1">
      <Button
        className={style.ant_btn}
        onClick={(event) => {
          handleInnerButtonClick(event);
          props.get_default_fit_for_key("double_gaussian");
          RenderSuccess(
            SucessMessageForDoubleGauss,
            !props.distributionsBooleans.bw
          );
        }}
        icon={
          <Checkbox
            checked={props.distributionsBooleans.double_gaussian}
            className={style.checkbox}
          />
        }
      >
        Calculate Double Gaussian Fit
      </Button>
    </Menu.Item>
    <Menu.Item key="10">
      <Button
        className={style.ant_btn}
        onClick={(event) => {
          handleInnerButtonClick(event);
          props.get_default_fit_for_key("gauss1");
          RenderSuccess(
            SucessMessageForDoubleGaussChild1,
            !props.distributionsBooleans.gauss1
          );
        }}
        icon={
          <Checkbox
```

```
              checked={props.distributionsBooleans.gauss1}
              className={style.checkbox}
            />
          }
        >
          Calculate First Gaussian Fit
        </Button>
      </Menu.Item>
      <Menu.Item key="11">
        <Button
          className={style.ant_btn}
          onClick={(event) => {
            handleInnerButtonClick(event);
            props.get_default_fit_for_key("gauss2");
            RenderSuccess(
              SucessMessageForDoubleGaussChild2,
              !props.distributionsBooleans.gauss2
            );
          }}
          icon={
            <Checkbox
              checked={props.distributionsBooleans.gauss2}
              className={style.checkbox}
            />
          }
        >
          Calculate Second Gaussian Fit
        </Button>
      </Menu.Item>
    </Menu>
  );

  const menu = (
    <Menu className={style.ant_btn}>
      {props.plotData.distribution_type === "Double Gauss" ? (
        <Dropdown
          className={style.ant_btn}
          overlay={menuForDoubleGaussian}
          disabled={props.plotData.x.length === 0}
        >
          <Button style={{ marginLeft: "0.9rem" }} className={style.ant_btn
             }>
            Double Gaussian Fit <DownOutlined />
          </Button>
        </Dropdown>
      ) : null}
      <Menu.Item key="2">
        <Button
          className={style.ant_btn}
          onClick={(event) => {
            handleInnerButtonClick(event);
            props.get_default_fit_for_key("gauss");
            RenderSuccess(
              SucessMessageForDefaultFit,
              !props.distributionsBooleans.gauss
            );
```

```
          }}
          icon={
            <Checkbox
              checked={props.distributionsBooleans.gauss}
              className={style.checkbox}
            />
          }
        >
          Calculate Gaussian Fit
        </Button>
      </Menu.Item>
      {/* <Menu.Item key="2">
        <Button onClick={showModal} className={style.customfitbutton}>
          Calculate Gaussian Fit
          <br /> For Custom Bin Input
        </Button>
      </Menu.Item> */}
      <Menu.Item key="3">
        <Button
          className={style.ant_btn}
          onClick={(event) => {
            handleInnerButtonClick(event);
            props.get_default_fit_for_key("landau");
            RenderSuccess(
              SucessMessageForLandauFit,
              !props.distributionsBooleans.landau
            );
          }}
          icon={
            <Checkbox
              checked={props.distributionsBooleans.landau}
              className={style.checkbox}
            />
          }
        >
          Calculate Landau Fit
        </Button>
      </Menu.Item>
      <Menu.Item key="4">
        <Button
          className={style.ant_btn}
          onClick={(event) => {
            handleInnerButtonClick(event);
            props.get_default_fit_for_key("bw");
            RenderSuccess(
              SucessMessageForBwFit,
              !props.distributionsBooleans.bw
            );
          }}
          icon={
            <Checkbox
              checked={props.distributionsBooleans.bw}
              className={style.checkbox}
            />
          }
        >
```

```
          Calculate Breit Wigner Fit
        </Button>
      </Menu.Item>
    </Menu>
  );

  return (
    <div>
      <InfoToolForHistogram message={InfoToolHistogramText} />
      <Dropdown
        overlay={menu}
        disabled={props.plotData.x.length === 0}
        className={style.ant_btn}
      >
        <Button className={style.ant_btn}>
          Fit Options <DownOutlined />
        </Button>
      </Dropdown>
      <Tooltip
        className={style.ant_btn}
        title={
          !props.histogramShowBooleans.enableFit
            ? "Please select a Fit from the Fit options first"
            : undefined
        }
      >
        <Button
          className={style.ant_btn}
          onClick={toggleFit}
          disabled={!props.histogramShowBooleans.enableFit}
        >
          {props.histogramShowBooleans.showFit &&
          props.histogramShowBooleans.enableFit
            ? "Hide Fit"
            : "Show Fit"}
        </Button>
      </Tooltip>
      <Button
        disabled={props.plotData.x.length === 0}
        onClick={toggleLogScale}
        className={style.ant_btn}
      >
        {props.histogramShowBooleans.logScale ? "Linear Scale" : "Log Scale
           "}
      </Button>
      <Button
        className={style.ant_btn}
        disabled={props.plotData.x.length === 0}
        onClick={() =>
          props.setHistogramShowBooleans({
            ...props.histogramShowBooleans,
            showMaxEntries: !props.histogramShowBooleans.showMaxEntries,
          })
        }
      >
        {props.histogramShowBooleans.showMaxEntries
```

```
            ? "Disable Max Entries"
            : "Find Max Entries"}
        </Button>
        {props.plotData ? (
          <Button
            className={style.ant_btn}
            disabled={props.plotData.x.length === 0}
            onClick={() => {
              ODSExport(props.plotData, props.titles);
            }}
            icon={<ExportOutlined />}
          >
            Export data
          </Button>
        ) : null}
        <OdsImport
          plotData={props.plotData}
          setPlotData={props.setPlotData}
          setTitles={props.setTitles}
        />
        <Button onClick={clearData} className={style.ant_btn}>
          Clear Data
        </Button>
        <Switch
          defaultChecked
          checked={props.histogramShowBooleans.fetchData}
          checkedChildren="Fetching"
          unCheckedChildren="Waiting"
          onChange={toggleSwitchChange}
          className={`${style.switch}`}
        />
        <div>
          <Button
            onClick={editXaxis}
            className={style.ant_btn}
            icon={<EditOutlined />}
          >
            Edit x-axis title
          </Button>
          <Button
            onClick={editYaxis}
            className={style.ant_btn}
            icon={<EditOutlined />}
          >
            Edit y-axis title
          </Button>
          <Button
            onClick={editTitle}
            className={style.ant_btn}
            icon={<EditOutlined />}
          >
            Edit histogram title
          </Button>
        </div>
        <div></div>
        <FindTitleToEdit
```

```
          edits={edits}
          setEdits={setEdits}
          titles={props.titles}
          setTitles={props.setTitles}
        />
      </div>
    );
};

export default ButtonBarForHistogram;
```

Listing 24: Button Bar For Histogram

Button Bar For Histogram Styles

```
.customfitbutton{
    height:auto;
}

.inputforx{
    top: -6.5rem;
    width:25rem
}

.inputfory{
    transform: rotate(-90deg);
    width:25rem;
    top: -50vh;
    left: -38.5vw
}

.inputforhistogramtitle{
    top:-84.5vh;
    width:25rem
}


.cross{
    border-color: red;
   color: red;
}

.switch{
    margin-left:1rem
}

.checkbox{
    margin-right:0.2rem;
}

.ant_btn{
    color:var(--color) !important;;
    background-color: var(--background-color) !important;
    transition: all 2s ease;
}

body{
```

```
    color:var(--color) !important;;
    background-color: var(--background-color) !important;
    transition: all 2s ease;
}

:disabled {
    color:grey!important;
}
```

Listing 25: Button Bar For Histogram Styles

Edit Titles Function

```
import { Input } from "antd";
import { CloseCircleOutlined } from "@ant-design/icons";
import { useEffect, useRef } from "react";
import { InputRef } from "antd/lib/input";
import style from "./buttonBarForHistogram.module.css";
import { HistogramEditBooleans, HistogramTitles } from "../Types/Types";
import { handleInputBlur, handleInputSubmit } from "../Shared/
    helperFunctions";

interface IProps {
  editKey: string;
  inputValue: string;
  setInputValue: React.Dispatch<React.SetStateAction<string>>;
  edits: HistogramEditBooleans;
  setEdits: React.Dispatch<React.SetStateAction<HistogramEditBooleans>>;
  titles: HistogramTitles;
  setTitles: React.Dispatch<React.SetStateAction<HistogramTitles>>;
}

const EditSelectedTitle = (props: IProps) => {
  const inputRef = useRef<InputRef>(null);

  useEffect(() => {
    if (inputRef.current) {
      inputRef.current.focus();
    }
  }, [props.edits]);

  return (
    <Input
      className={
        props.editKey === "editXaxisTitle"
          ? style.inputforx
          : props.editKey === "editYaxisTitle"
          ? style.inputfory
          : style.inputforhistogramtitle
      }
      value={props.inputValue}
      onChange={(e) => props.setInputValue(e.target.value)}
      onBlur={() =>
        handleInputBlur(
          props.setInputValue,
          props.setEdits,
          props.edits,
          props.editKey
        )
      }
      onPressEnter={() =>
        handleInputSubmit(
          props.setInputValue,
          props.setTitles,
          props.setEdits,
          props.inputValue,
          props.titles,
```

```
            props.edits,
            props.editKey
          )
        }
        ref={inputRef}
        suffix={
          <CloseCircleOutlined
            className={style.cross}
            onClick={() =>
              handleInputBlur(
                props.setInputValue,
                props.setEdits,
                props.edits,
                props.editKey
              )
            }
          />
        }
      />
    );
};

export default EditSelectedTitle;
```

Listing 26: Edit Titles Function

Find Title To Edit Function

```
import { useState } from "react";
import { HistogramEditBooleans, HistogramTitles } from "../Types/Types";
import EditSelectedTitle from "./editsForAllTitles";

interface IProps {
  edits: HistogramEditBooleans;
  setEdits: React.Dispatch<React.SetStateAction<HistogramEditBooleans>>;
  titles: HistogramTitles;
  setTitles: React.Dispatch<React.SetStateAction<HistogramTitles>>;
}

const FindTitleToEdit = (props: IProps) => {
  const [inputValue, setInputValue] = useState("");

  const [editKey] = Object.entries(props.edits).find(
    ([key, value]) => value === true
  ) ?? [""];
  if (editKey !== "") {
    return (
      <EditSelectedTitle
        editKey={editKey}
        inputValue={inputValue}
        setInputValue={setInputValue}
        edits={props.edits}
        setEdits={props.setEdits}
        titles={props.titles}
        setTitles={props.setTitles}
      />
    );
  }
  return null;
};

export default FindTitleToEdit;
```

Listing 27: Find Title To Edit Function

6.2.5 Types

Typescript Types For App

```
//Contains all the types necessary data structure for the project

export interface HistogramDataTypes {
  x: number[];
  y: number[];
  y_fit_gauss: number[];
  y_fit_landau: number[];
  y_fit_bw: number[];
  y_fit_double_gaussian: number[];
  y_fit_gauss1: number[];
  y_fit_gauss2: number[];
  gauss_amplitude: number;
  gauss_amplitudeError: number;
  gauss_mean: number;
  gauss_meanError: number;
  gauss_sigma: number;
  gauss_sigmaError: number;
  gauss_ndf: number;
  gauss_chi2: number;
  landau_amplitude: number;
  landau_amplitudeError: number;
  landau_mean: number;
  landau_meanError: number;
  landau_sigma: number;
  landau_sigmaError: number;
  landau_ndf: number;
  landau_chi2: number;
  bw_amplitude: number;
  bw_amplitudeError: number;
  bw_mean: number;
  bw_meanError: number;
  bw_sigma: number;
  bw_sigmaError: number;
  bw_ndf: number;
  bw_chi2: number;
  double_gauss_amplitude: number;
  double_gauss_amplitudeError: number;
  double_gauss_mean: number;
  double_gauss_meanError: number;
  double_gauss_sigma: number;
  double_gauss_sigmaError: number;
  double_gauss_amplitude2: number;
  double_gauss_amplitudeError2: number;
  double_gauss_mean2: number;
  double_gauss_meanError2: number;
  double_gauss_sigma2: number;
  double_gauss_sigmaError2: number;
  double_gauss_ndf: number;
  double_gauss_chi2: number;
  limit_for_axis: number;
  max_y_for_hist: number;
  distribution_type: string;
```

```
}

export interface HistogramTitles {
  xAxisTitle: string;
  yAxisTitle: string;
  histogramTitle: string;
}

export interface HistogramEditBooleans {
  editXaxisTitle: boolean;
  editYaxisTitle: boolean;
  editHistogramTitle: boolean;
}

export interface HistogramShowBooleans {
  showResults: boolean;
  showFit: boolean;
  logScale: boolean;
  enableFit: boolean;
  isModalOpen: boolean;
  showMaxEntries: boolean;
  fetchData: boolean;
  loading: boolean;
}

export interface DistributionsBooleans {
  gauss: boolean;
  landau: boolean;
  bw: boolean;
  double_gaussian: boolean;
  gauss1: boolean;
  gauss2: boolean;
}

export type DistributionsBooleansKey =
  | "gauss"
  | "landau"
  | "bw"
  | "double_gaussian"
  | "gauss1"
  | "gauss2";

interface annotations {
  x?: number;
  y?: number;
  xref?: string;
  yref?: string;
  type?: string;
  text?: string;
  align?: string;
  showarrow?: boolean;
  arrowhead?: number;
  arrowcolor?: string;
  ax?: number;
  ay?: number;
  font?: { size: number; color: string };
```

```
}

export interface Layout {
  title: string;
  xaxis: { title: string; range: number[] };
  yaxis: { title: string; range: number[]; type?: string; gridcolor?:
     string };
  modebar: { orientation: string };
  font: { family: string; size: number };
  width: number;
  height: number;
  legend: {
    x: number;
    y: number;
    traceorder: string;
    font: { family: string; size: number; color: string };
  };
  annotations: annotations[];
}
```

Listing 28: Typescript Types For App

6.2.6 Warnings

Warning Messages

```
//contains reusable messages

export const SucessMessageForDefaultFit = "Default Fit for Gaus
    distribution generated from ROOT used"

export const SucessMessageForLandauFit = "Default Fit for Landau
    distribution generated from ROOT used"

export const SucessMessageForBwFit = "Default Fit for Breit Wigner
    distribution generated from ROOT used"

export const SuccessMessageForCustomFit = `Custom Fit used. Number of
    points used: `

export const SuccsesMessageForODSExport = 'Data exported successfully'

export const WarningMessageForServer = 'Server is not live at the moment'
```

Listing 29: Warning Messages

Warning Messages Handler

```
import { notification } from "antd";

//Contanins reusable warnings/success messages

export const RenderSuccess = (message: string, shouldRender: boolean) => {
  if (shouldRender) {
    notification.success({
      message: "Success",
      description: message,
      placement: "topRight",
      duration: 3,
    });
  }

  return null;
};

export const RenderWarning = (message: string) => {
  notification.error({
    message: "Warning",
    description: message,
    placement: "topRight",
    duration: 3,
  });

  return null;
};
```

Listing 30: Warning Messages Handler