

The Development and Application of Data Analytics Pipelines in High-throughput Experimentation

A dissertation submitted to The University of Manchester for the degree of
Master of Physics
in the Faculty of Science and Engineering

Year of submission
2023

Student ID
10294857

School of Natural Sciences
Tomasz Neska

Contents

Contents	2
Abstract	3
Acknowledgements	4
1 Introduction	5
1.1 Motivation	5
1.2 Aims and objectives	6
1.3 The Overview of the Report	7
2 Introduction to the Existing Architecture	8
2.1 Introduction	8
2.2 Data Structures	8
2.3 the API	10
2.4 The User Interface and Authentication	10
3 Methodology	12
3.1 Introduction	12
3.2 New Data Structures	13
3.3 Group Functionality	13
3.4 Jupyter Drivers and Notebook	14
4 Critical Evaluation	17
5 Conclusions and Future Work	18
5.1 Future Work	18
5.2 Conclusion	19
References	19
Appendices	23
A Libraries and technical details	23
B Testing and Continuous development cycle	23
C Glossary	23
D Jupyter Notebook	25

Abstract

This report presents the development of a data analytics pipeline for the analysis of experimental data in material science. The API framework was transformed into a functional dashboard with the ability to create customised groups for data collection and retrieval, and to search datasets by metadata. The pipeline includes an API and a user interface for data manipulation and authentication, utilizing the MongoDB database to store data in a tree structure. The user interface enables researchers to easily access and manipulate data, and provides security measures such as password hashing and JSON Web Tokens for authentication. The pipeline was successfully demonstrated on a virtual server. While there are opportunities for improvement, such as the implementation of a visual interface and standardisation of authentication, this project presents a suitable architecture for use in high-throughput experiments.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr Patrick Parkinson for the continuous support in my work and research. For his patience, motivation, enthusiasm and immense knowledge.

Besides my advisor, I would like to thank the rest of the research group at the OMS Lab: Dr Stephen Church, Nikesh Patel, Nawal Al-Amairi, Ruqaiya Al-Abri and Hoyeon Choi.

Additionally I would like to thank my friends for the support both technical and emotional: Tristan Broadhead, Euan Lacy and Zobia Hussain.

1 Introduction

1.1 Motivation

In 2009, Microsoft Research published "The Fourth Paradigm: data-intensive scientific discovery"[1], a book containing a series of essays outlining a new way of thinking about research and conducting scientific innovation. This approach focused on the idea that nature could be understood directly from data, and that even if this was not possible, the correct utilization of data would provide valuable insights. The goal was to make the data obtained from experiments work for researchers, enabling them to harness their curiosity and creativity, and save time by not having to organise their data. This approach would then lead to a more prosperous and effective scientific process.

The book presents the paradigm as an improvement on the way scientific research was conducted at the time of publication. In fact, the use of data has rapidly become a key part of scientific research, with innovations such as ROOT [2] and the Perovskite database [3] becoming widely used. The aim of this project is to leverage the power of data management in the field of material science, specifically by demonstrating its usefulness in analyzing nano-ring spectra as part of an analytics pipeline.

The use of data is essential for understanding how nature functions. Every key scientific observation can be expressed as a relationship between two or more variables, which is then tested by collecting data and using statistics to validate the relationship. This process of data collection and analysis is at the heart of the scientific method. Recently, the ability to perform large-scale experiments that generate massive amounts of data has made this process nearly automatic [4] [5]. As computing power increases, it becomes possible to generate multi-dimensional data plots that provide a clearer understanding of the system being studied. This allows researchers to make informed decisions and avoid pursuing uncertain approaches. Data-driven methods have already proven effective in fields such as meteorology [6], and their application is increasingly yielding positive results in a variety of disciplines, including outside of science, such as advertising [7]. Data management is also key in the scientific process itself, as it helps to ensure that studies and analyses are reproducible. Institutions such as the UK Reproducibility Network (UKRN) [8] use meta-research to understand the strengths and weaknesses of analysis approaches. If the approach outlined in this report is followed, it can be quickly and easily quantified and accessed.

These approaches are promising due to their role in the scientific process. To illustrate, consider a collection of books. Most people do not organize their bookshelves because they do not have the time or need to do so. However, if the owner decides to start lending out the books or wants to restrict access to certain books, the situation becomes more complicated. In this case, the bookshelf becomes a library and the books must follow the principles of data storage for the system to work properly. It is important to focus not only on *data custody*, but also on storing data without

redundancy, in an accessible and secure way.

There is currently no single standard way of storing scientific data. Some publishers have their own standards, and there are organizations [9] that handle data storage for research in specific fields. For example, ELIXIR is a large organization that works across Europe to develop a data pipeline that ensures proper storage and compliance with accessibility laws, both legal and technical. In the field of material science, projects have been implemented that use the FAIR data principles [10] as a foundation for data storage. One example is a perovskite solar cell database that allows users to upload data and offers interactive tools for data exploration and visualization as an incentive.

High-throughput experimentation is an area where data analytics pipelines excel, as they allow for the automation of repetitive tasks and therefore reduce the number of work hours required and improve efficiency. These pipelines are commonly used in pharmaceuticals [11] to test large quantities of chemical compounds and in the design and testing of circuits[12]. They are particularly well-suited to these tasks because they allow researchers to leverage the strengths of computers, such as their ability to automate and perform rapid parallel or serial tasks. For example, a data pipeline could be used to create optical spectra of objects. If the goal is to understand the optical properties of a particular material by producing multiple spectra using optical equipment, the pipeline could be used to move the data from the data acquisition equipment to a database, where it can be processed and visualized for the user. High-throughput experimentation is frequently used to expand the understanding of various material properties [13] and to identify the best growth conditions for nano-materials [14]. This is especially useful in material discovery, where the ability to summarize large data sets is key.

The aim of this project is to address the current gap in material sciences and evaluate the usefulness of data science methods in this field. The project focuses on applying tried and tested data science practices to material research and developing an architecture that enables researchers to fully leverage existing technology.

1.2 Aims and objectives

The main objective of this project is to design and implement a user-friendly interface that can be widely accessed to provide and retrieve scientific data. To achieve this, we will use the architecture developed in the summer of 2022 [15] to build a data analytics pipeline, consisting of a cloud-based database, an API, and an interface. This mirrors the existing architecture described in Section 2 and expands the user-facing part of the architecture. The interface will have two components: a data analysis dashboard that allows for an asynchronously updated notebook, and a demonstration using a virtual machine that shows the data journey from data acquisition in the lab to analysis, as depicted in Figure 1. The data analysis dashboard will contain important details and summaries of the data in the database, and will be used to demonstrate the value of this architecture in guiding

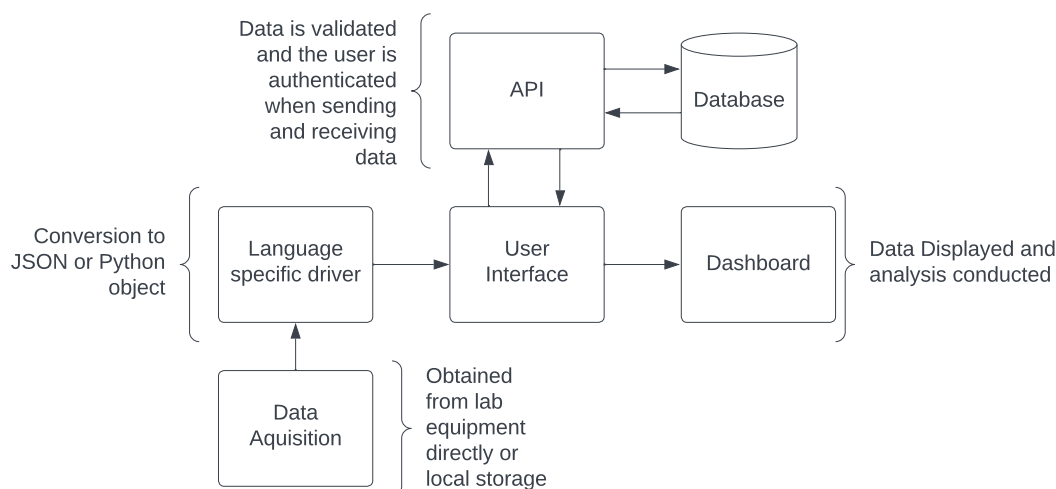


Fig. 1. The diagram describing the journey of the data across the architecture.

decision-making based on data by providing a simple tool to monitor the state of the data within a certain scope.

The project aims to demonstrate the usefulness of data management in material science research by designing and implementing a data analytics pipeline, including a cloud-based database, API, and user interface. The user interface consists of a data analysis dashboard and a virtual machine demonstration of the data journey from acquisition in the lab to analysis. In addition, the project aims to standardize metadata and allow users to declare their own metadata as key-value pairs, as well as allowing for the grouping of data using metadata and the creation of custom abstract objects within the existing architecture. Finally, the project aims to enable the ability to append and remove data custody rights for individual users and groups.

1.3 The Overview of the Report

In this report, I will explore the current architecture, including the API that resides on a server and the user-facing interface. I will then describe the improvements that were made to this existing architecture in order to achieve the aims outlined in Section 1.2. Next, I will explain how the success of the project was achieved and how it was evaluated. Finally, I will suggest future improvements to the architecture and compare the usefulness of the developed architecture to competing products.

2 Introduction to the Existing Architecture

2.1 Introduction

The original architecture for this project was created in the summer of 2022. This module was named "resdata" and was the core that allowed for the development of the analytics pipeline. The architecture was developed with the following functions in mind:

- Data insertion and retrieval
- Data visualization
- Authentication during data insertion/retrieval
- Management of read/write permissions for the data at the user level

During the course of this project, these functions were expanded upon. In this section, we will first describe how the existing architecture works and how it achieves its goals, in order to provide a clear understanding of the changes made to it. The data is stored using a database, and the existing architecture employs MongoDB [16]. This database has the advantage of easy handling of heterogeneous data, and it has an extensive library in Python, which allows for interfacing between the two.

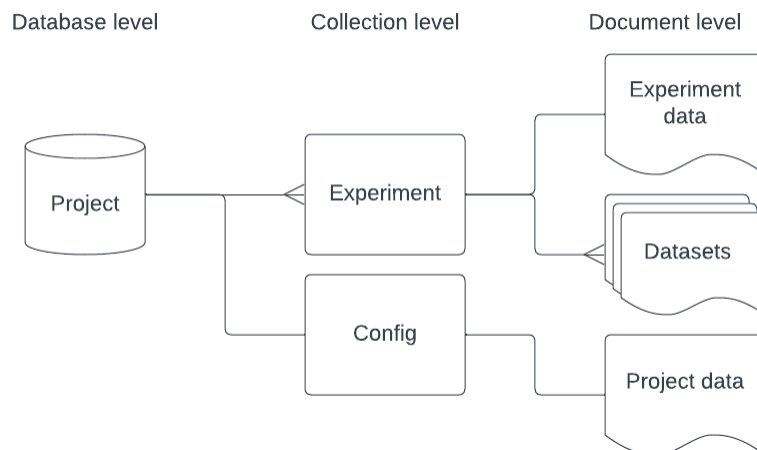


Fig. 2. The diagram describing how MongoDB stores data.

2.2 Data Structures

The essential part of any data pipeline is a robust abstraction that allows for structuring the data into something that can be manipulated and translated into database-compatible objects. The abstraction of the data structure for this pipeline was heavily influenced by the application of this

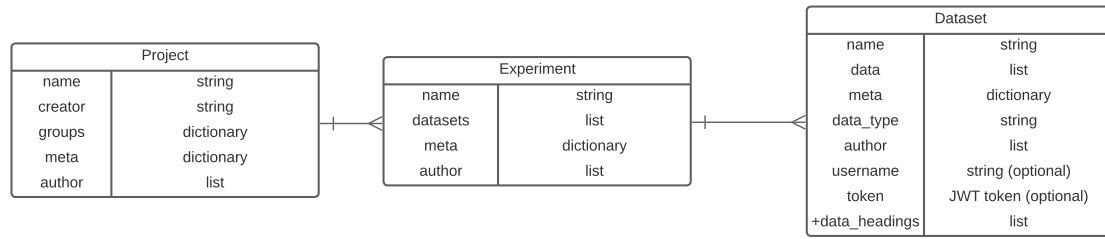


Fig. 3. The class diagram describing the data structure and the variables contained within them. Each variable has a variable type associated with it. The field added during this project are marked with a "+".

pipeline in research conducted on optical circuits such as nano-wires [17] and nano-rings [18]. This highlights the biggest challenge in designing optimal data structures: it is very difficult to design a reliable structure if the data is extremely heterogeneous.

To address this challenge, a tree data structure was implemented, inspired by ROOT [2] and the current storage practices of the OMS group. This tree is composed of three levels. Each node of the tree contains the information specific to itself and the links to the nodes below it. The first level is the project. This is envisioned as a user-specific place for collecting work done for a project. It is implemented as a database, with each project being its own database residing on a common server. The second level is an experiment node, implemented as a group of documents within a database. The final third level is a dataset, a single document. This forms the unit used in data insertion and retrieval. The structure and content of each node are shown in Figure 3.

It is important to note that each of the data structures is a Python object that utilizes the Pydantic library [19]. This library allows for the creation of Python objects that enforce variable data types. This is not a native function of Python, but it is essential in ensuring that the data input into those objects is restricted to exactly what is allowed. For example, numerical arrays cannot be processed and will raise errors if the entries in those arrays are not integers or float numbers. Therefore, this library is used to ensure data type validation. The data structure in Figure 3 is used in the interface, as handling of these objects is easier in Python.

This data then has to be converted into plain text to be sent as part of an HTTP call. This can be accomplished in multiple ways, but in order to ensure data readability and to ensure that the medium used for transferring the data is language-agnostic, it was decided that JavaScript Object Notation (JSON) was going to be utilised. JSON objects are readable by the user and, most importantly, are an easy way of sending complex structures using HTTP calls. The use of JSON also brings the added benefit of the Pydantic library allowing for the conversion into JSON objects using a single function call.

It is important to mention some key design choices made as part of this architecture. Specifically, the API only inserts data into the database at the level of a dataset. This was done to control the size of the data inserted into the database at once, minimizing the likelihood of bottle-necking and allowing for data streaming into the database. The data can be distributed across multiple experi-

ments and further into individual datasets. Large objects are fragmented such that a single dataset contains at most a single array of data along with the metadata. Bulk insertion is not the purpose of this architecture. The data that this architecture is designed for is large in number of instances but with a relatively small size of approximately 5 MB.

2.3 the API

The API contains a series of functions which contain a path. The functions utilise a request library [20] to call a server utilising HTTP calls. These calls were designed with the REST principles in mind [10]. Those principles state that data should be findable, accessible, interoperable and reusable. Those functions run asynchronously and are responsible for modifying the database. This database was implemented using a MongoDB library [16]. MongoDB was chosen due to the ability of implementing a database in Python while SQL is implemented in C and C++. Additionally, MongoDB allows for a *dynamic schema* while SQL requires a fixed *schema*. This is an advantage due to the variety of scientific data that this pipeline is meant to operate on. Additionally MongoDB is *open source* and hence doesn't require a license to be used which meant no additional cost.

The architecture contains three groups of functions: data manipulation, initialization, and user management. Firstly, data manipulation functions allow for the sending and retrieval of data. For example, the insertion of a dataset is done using an API function named *insert_single_dataset*, which is called in the interface by a function named *insert_dataset*. The interface also uses the same API call to insert entire experiments or projects.

Secondly, the API contains functions for initialising experiments and projects. This is necessary due to the way that MongoDB stores this data structure. Experiment and project nodes are not documents hence they don't exist until there is a document within them. Therefore, to store data on the level of those nodes, it is necessary to insert a document within them. For an experiment, a document is inserted along with other datasets, but it shares the name of the experiment. For a project a separate collection named *config* is created to store the project's data.

The last part of the API functionality encompasses the user authentication and creation functions. Authentication is conducted by a separate object named *User_Auth*, which handles token generation, password validation, and hashing. For example, user creation is done using the *create_user* function within the interface, which in turn uses the *User_Auth* object to run the *add_user* function.

2.4 The User Interface and Authentication

The interface is composed of an *interface* library and a Jupyter notebook. It contains an object that can be used to utilize the functionality of the API. The object allows users to conduct API calls in

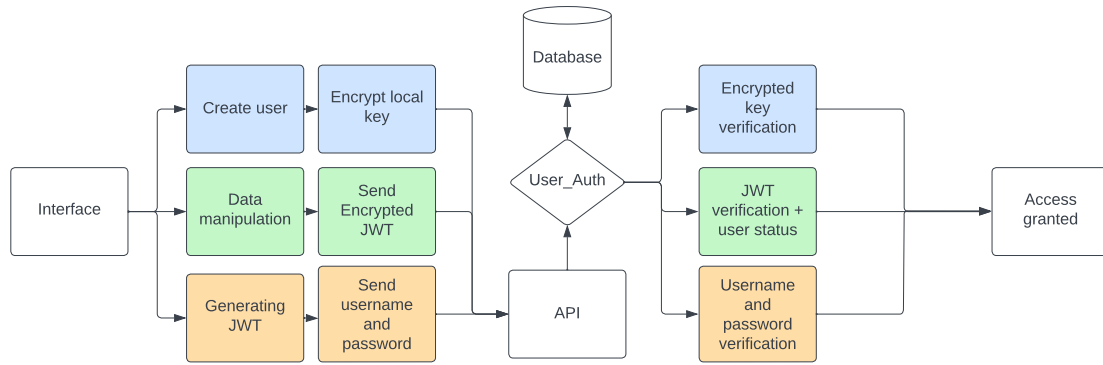


Fig. 4. The diagram describing the flow of data in the authentication process. These processes can be split into three categories: open endpoints (blue), data manipulation (green) and token generation (yellow).

a user-friendly way. Each function in the object performs one or more API calls. For example, the function responsible for returning a project uses multiple API calls to first authenticate the user, then find the names of the datasets and experiments to be returned. Finally, it recursively traverses the tree, populating the project object which is then returned as a Python object that can be easily manipulated. In this way, the API interface provides a convenient and user-friendly way to access and utilize the functionality of the API.

Authentication is conducted by a combination of the username and password approach and the usage of a JWT (JSON Web Token) [21]. The username and password is used to generate the token which is then subsequently used for any calls made by that user to authenticate the data access permission. This hybrid approach has been used specifically for this use case and the details of the improvement are mentioned in the final section. This approach follows the important requirements for sending data securely. Namely, only encrypted data is sent for the purpose of authentication. The interface and the API use different encryption algorithms and the database only stores encrypted data. Additionally the passwords are never saved.

The core of security is handled by functions within a class called *User_Auth*. This object is initialized when authentication is needed on the server side and uses its functions to perform the necessary actions for authentication. This includes checking if the user with the given credentials exists, hashing and authenticating the received token. The authentication journey for a user begins with the token generation function. The interface sends the username and encrypted password to the API. The password is then encrypted again using the *salt* that was generated during user creation. If the resulting hash matches the one stored in the database, a token is generated and the user's status is changed to "activated." This token expires after 30 minutes, and the user's status is then changed to "deactivated." All other functions use the generated token for authentication. The token is returned and stored within the interface for the duration of the session. Once the token expires using the same token will fail to authenticate, even if it is correct. It is important to note that the password hashes are compared using a *compare digest* function, which takes the same amount of time even if some characters match. This is done to prevent a *time-attack*. A time-attack is a method

of decoding a password by making multiple calls and timing how long it takes for the server to respond. If the server takes longer, it means the last character is correct. This allows for a more efficient brute-force approach, reducing the security of the authentication. By employing the *compare digest* function to compare the hashes, the security is made more robust.

The interface contains three types of functions used for operating on data. First are the *name* functions, which allow the user to view the nodes on the database that they have access to. These functions only return the names that the user is listed as an author for. The second type of function are the insert functions, which allow the user to insert data at the dataset, experiment or project level. This is achieved by passing the Python object used for storing the appropriate node and then using the API to first initialise the project, and experiment (if necessary), and then insert the datasets individually. The third type is a set of author management functions, which allow the user to add a user to a dataset and share it. This can be done recursively across entire experiments and projects. These functions were not functional in the previous architecture, but were tested and made operational during the course of this project, as the group functionality mentioned in Section 3.3 uses the same methods.

3 Methodology

3.1 Introduction

One of the major challenges of designing a data pipeline is the difficulty of packaging data in a form that can be easily transferred and analysed. This is because data is often highly heterogeneous - for example, a combination of spectral images and multidimensional spectra may require different types of storage. The goal is to provide an abstraction that standardises and homogenises the data for a specific use case. This is especially important because different people have different ways of storing data that work best for them. This can include using H5 files or different formats of JSON files. However, if the data is homogenised, it becomes easily understood by any party interested in it. These ideas are best conveyed by the FAIR principles of data storage [10]. Firstly, the data must be findable. This is essential because the purpose of the pipeline is to allow the user to make decisions about what to do with the data, and this is difficult if the data cannot be found. The second principle is accessibility. The user must know how to access the data, which can involve a clear user interface or visualization tools to give insight into what the data represents. The final principle is interoperability. This principle emphasizes the importance of attaching other data, such as metadata, to enable interoperability with applications that use the data to achieve a goal.

The complete list of functions that were designed during the duration of this project is listed in Table 1.

Function name	Module name	Function purpose
purge_function	API_server	Clears the database of all data
return_all_dataset_names_group	API_server	Returns the names of the datasets within a group
return_all_experiment_names_group	API_server	Returns the names of the experiments within a group
return_all_project_names_group	API_server	Returns the names of the projects within the group
add_group_to_dataset	API_server	Adds a dataset to a group
meta_search	API_server	Returns the names of datasets containing a meta data variable
return_hash	interface	Shake 256 hashing algorithm
purge_everything	interface	Clears the database of all data
experiment_search_meta	interface	Fetches the datasets matching the metadata variables
add_group_to_dataset	interface	Adds a dataset to a group
add_group_to_experiment	interface	Adds an experiment to a group
add_group_to_experiment_rec	interface	Adds an experiment and all datasets contained within it to a group
add_group_to_project	interface	Adds a project to a group
add_group_to_project_rec	interface	Adds a project and everything contained within it to a group
add_group_to_dataset_rec	interface	Adds a dataset and the things that contain it to a group
get_experiment_names_group	interface	Returns the names of the experiments within a group in a specified path
get_dataset_names_group	interface	Returns the names of the datasets within a group in a specified path
get_project_names_group	interface	Returns the names of the projects within the group
author_query	interface	Returns the list of data nodes containing the author/group specified
tree_print_group	interface	Prints the structure of a group in a readable manner
plot_from_dataset	jupyter_driver	Print the data from a dataset passed in
summarise_dimensions	jupyter_driver	Summarise the mean of the quantities in the datasets passed in

Table 1. The full list of functions added to the modules during this project. Functions contained within the module API_server are public API calls.

3.2 New Data Structures

The new data structure, "Ring," is a structure used to demonstrate the ability to apply the abstraction of the previous architecture to a more complex structure. The ring structure is modeled after the variables that a nano-ring would contain [22]. It is separated into two types of data: the dimension dataset and the spectrum dataset. The dimension dataset contains variables inherent to the ring, such as the ring's quality, pitch, diameter, and most importantly, its identification number. Due to the nature of it being an abstraction, the ring ID is restricted to being an integer. The spectrum dataset can be a number of different spectra and is used to reflect the fact that quantum rings are tested using an optical spectrum. This is then converted into a list of datasets linked by the same entry in the metadata. There are also new fields added to the existing data structures, as shown in Figure 3. The addition of "data_headings" was done to allow for printing the data correctly with the appropriate axis labeling.

3.3 Group Functionality

One of the major drawbacks of using the architecture described in Section 2 is the difficulty in recalling data. The only function that allows for data browsing is the *tree_print* function, which prints out the structure of the data with the names of the projects, experiments, and datasets. This allows the user to call the dataset by its name and path. To address this issue, two critical functions were created. The first function returns a nested list with the names of all the nodes accessible to the user, rather than just printing the structure. This allows for the creation of an automatically up-

dated dashboard by passing the dictionary. The second function is a metadata search, which allows for recalling every dataset linked by a common metadata variable. This is particularly useful for the new ring data structure, as all the spectra and internal variables are linked together by the same metadata variable.

Another improvement is the introduction of groups. Each dataset already has a list of authors whose permissions are checked, so it makes sense to extend this and create a parallel system of groups that takes the same slot as the author. This system allows the user to add datasets, experiments, or even entire projects to a group, creating a collection of nodes that can be recalled together. This customization allows for further structure to be added and enables analysis to be performed on certain sections of a larger dataset. Recalling this data together with the previous functions is useful for producing dashboards or conducting analysis.

A final function worth mentioning is the author query, which allows for recalling groups but also for recalling all datasets that contain a specified author. This transforms the cumbersome data recalling system into something that can be easily automated and customized by the user. This is particularly useful because metadata is user-defined and can contain any level of complexity while still maintaining linear lookup time, as long as it follows the key-value pair structure of the dictionary.

3.4 Jupyter Drivers and Notebook

The user-facing side of the API consists of the *interface* module, which stores the functions that the user has access to, and the Jupyter Notebook, where those functions are run. The Jupyter Notebook is divided into cells, which allow for code to be run in sections. The demonstration notebook shown in Appendix D provides the full notebook demonstrating how the pipeline developed can be utilized. For example, the notebook allows the user to pull data from a previously populated database for analysis, using the Ring object as an example. The analysis calculates the average values of the pitch, quality, and ring diameter in one cell, and in another cell, a single ring and all its attached data is pulled and printed, including the dimensions and plotted spectra. The data used in these examples contains random numbers for testing purposes, but once real data is pulled, the user can perform calculations to obtain the desired values from the experiment (see Figure 6). Additionally, the functions designed for adding authors to datasets allow the current user to share the analysed data with other users of the database, which is important for peer review in the scientific process. This data pipeline is organized in a way that makes it easy to share.

The key aspects of the implementation of the notebook should be viewed from the user's perspective. The module named *jupyter_driver* holds functions that can be applied to the data structure in order to present the data in a readable manner. This module contains two functions: *plot_from_dataset* and *summarise_dimensions*. These functions take in a dataset and return a print of the data contained within, ranging from a print of the mean values to a plot of the data. These functions demonstrate the types of analysis that can be performed on the dataset once it is retrieved (see Figure 7).

```

In [8]: # populate the database
no_of_rings = 5
no_of_experiments = 2

import testing as t
api.generate_token(username, password)
project_name = "project_test_"
experiment_name = "experiment_test"
ds_size = 2000
for i in range(0,4,1):
    t.generate_optics_project(file_name,[no_of_rings, no_of_experiments], project_name+str(i),
                             experiment_name, author_name=username, size_of_dataset=ds_size)
    project = t.load_file_project(file_name)
    api.insert_project(project)

# generate special project for analysis showcase
project_name = "special_project"
experiment_name = "special_experiment_test"
ds_size = 5000

search_value = 15.65
t.generate_optics_project_2(file_name, [1,1], project_name, experiment_name, username, ds_size, search_value)
project = t.load_file_project(file_name)
api.insert_project(project)

print("Database populated")
print("")
api.tree_print()

```

The data tree:

```

project_test_0
-->experiment_test 1
-->ring_no. 0
-->PL spectrum - ring no.0
-->TRPL spectrum - ring no.0
-->Lasing spectrum - ring no.0
-->ring_no. 1
-->PL spectrum - ring no.1
-->TRPL spectrum - ring no.1

```

Fig. 5. The Jupyter notebook cell used to populate the database with test data. This shows the insertion of data into the database along with the required authentication.

```

In [11]: # Pull single ring -> plot dimensions and all spectra
import sys
sys.path.insert(0, "/home/splitsky/Desktop/code_repositories/resdata/")
import interface as ui
import jupyter_driver as dr

api = ui.API_interface(path)
api.check_connection()
api.generate_token(username, password)
api.tree_print()
ring_id = 0
experiment_id = "experiment_test 1"
project_id = "project_test_0"
projects = api.experiment_search_meta(meta_search={"ring_id": ring_id}, experiment_id=experiment_id, project_id=project_id)
for entry in projects:
    print(entry.name)
for entry in projects:
    dr.plot_from_dataset(entry, entry.data_type, entry.name)

```

```

-->ring_no. 0
-->PL spectrum - ring no.0
-->TRPL spectrum - ring no.0
-->Lasing spectrum - ring no.0
ring_no. 0
PL spectrum - ring no.0
TRPL spectrum - ring no.0
Lasing spectrum - ring no.0
ring diameter : 0.9637907737377375
quality : 1
pitch : 0.18965777228893765
threshold : 0.6673953433290012
Printing 2D spectrum with error bars

```

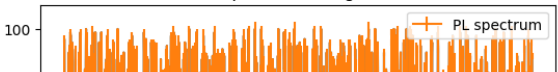


Fig. 6. The Jupyter notebook cell used to recall all datasets for a single ring and plot it.

```

In [12]: # Pull all rings from experiment and summarise dimensions
import jupyter_driver as dr
# pull full experiment
experiment = api.return_full_experiment(project_name="project_test_0", experiment_name="experiment_test 0")
# separate dimensions
datasets = experiment.children
desired_data = []
for ds in datasets:
    if ds.data_type == "dimensions":
        desired_data.append(ds)

# summarise the quantities
dr.summarise_dimensions(desired_data)

Average values for dimensions:
ring diameter : 0.7422914029193357 +/- 0.3187223925063638
quality : 6.4 +/- 3.3823069050575523
pitch : 0.4082606383008508 +/- 0.2555024488839914
threshold : 0.6063285670392031 +/- 0.28511286866429963

```

Fig. 7. The Jupyter notebook cell used to recall an experiment and summarise properties of the items contained within it.

The utility of this pipeline can best be described with an example of its use in investigating a new material. This pipeline can be thought of as a tool used by a researcher to automate their research process. For example, consider nano-materials grown using selective area epitaxy [22]. These materials are grown in large sheets which may contain hundreds of nano-sized objects, making it difficult to manage data manually for each object. The objects are grown under certain conditions, which can be described by a list of variables. The objects' optical properties are then tested using a laser, which generates a spectrum of intensity versus wavelength for the material. This can be performed in a high-throughput experiment. This process can produce a lot of data, as the laser setup can be automated to generate multiple spectra for a sheet of nano-materials. This is the scenario for which this project was designed. Each object (e.g. a nano-ring) has a list of characteristics, such as dimensions and growth parameters. The abstract "Ring" data structure allows for the storage of these parameters, along with the spectrum generated for the ring (linked using the ring identification number). The meta data can be modified to allow for a two-dimensional identification, using a coordinate system to identify the rings. The spectra can also be differentiated by meta data such as the date they were performed or the machine identification number.

In this process, the application of this pipeline would allow the researcher to conduct experiments and generate a data tree describing the nano-ring they are studying. The final utility of this structure would allow the researcher to create a visual representation of how the parameters change and analyse the relationship between two parameters like a specific growth parameter and a lasing threshold. This analysis would automatically update as more data is added to the database, providing the researcher with insight to guide their decisions based on the data they see. This is the strength of this approach – the ability to perform a small study and produce a dashboard that provides insights and allows the user to make informed decisions. Additionally, it would allow a different researcher to gain access to the selected datasets, or even the entire project, to perform a different form of analysis or review the analysis from the same data.

4 Critical Evaluation

The first most important part of ensuring that the pipeline works as intended is testing. This has been conducted using the Pytest library [23]. The details of the tests conducted are detailed in Appendix A. The tests pass, and the testing stage has been conducted after each function was implemented. This methodical approach allowed for the code to work at all times. The usage of a version controlled system has been essential in allowing for the development of a project of this scale. The demonstration was conducted on Monday, 5/12/2022. It was successful and allowed for the deployment of the API onto a virtual server. The server successfully responded to the calls made from the local interface, hence demonstrating the data journey as described in Section 1.2. The project was an overall success. The changes made to the functionality transformed it from an API framework into an operable data analytics pipeline. The addition of groups brought a substantial quality of life improvement and allowed for streamlining data collection and retrieval. Overall, the goals explained in Section 1.2 have been met.

However, the project still requires work and there are multiple points at which it could be improved. The first and most important improvement is changing the authentication into a true OAuth2 system, which employs only tokens and does not expose the hashed password to a man-in-the-middle attack. This has been mitigated by hashing the passwords using interface-specific salt, but this still leaves a lot to be desired in terms of the robustness in security. This vulnerability of the system was discovered during the demonstration.

An important aspect in data custody is the application of authentication with the purpose of ensuring data security. The choice to use an API means that there is an exposed back-end which can be called by non-desirable sources that are not the user. Implementation of authentication during data entry prevents the users from modifying the database or inserting things which they don't have authorisation to modify. However, it is also important to secure the connection between the interface of the pipeline and the API server that manages the calls made to the database, as this connection is public. In this case, it was addressed as seen in Figure 4.

The best way to understand the advantages of this architecture is to compare it to an old form of data organization. Namely, the use of the H5 data format [24]. While this approach is simpler and lacks structure and ease of integration, it has been used for decades. However, the biggest flaw with this approach is the lack of a central server. If the data is copied, it must be updated diligently, or the user will end up with multiple copies at different stages of completion. Additionally, the H5 file struggles when it needs to be combined with additional data for a meta-analysis, while the API approach allows for the data to be fragmented and recombined into any shape that's necessary. The user also needs to write their own drivers to convert the data into something accessible, and there are no rules for the form that the data is saved in. This lack of standardization leads to every user having their own structure, which in turn leads to the need for many different drivers. This is the final flaw of this approach: the H5 approach is not scalable. Once the user runs out of USB

drives or struggles to find the appropriate files, it becomes necessary to introduce a data architecture. The API approach skips this step completely, starting with a heterogeneous, easily accessible, and secure data pipeline. However, a big disadvantage of this system is the decentralised nature of the pipeline. If the sever is down due to a technical issue the data cannot be accessed.

Additionally, the form of data storage has been selected specifically to meet the needs of the pipeline. It must handle many datasets in a readable fashion while not sacrificing speed. SQL databases are incredibly fast and allow for very efficient lookup times. However, they require a lot of abstraction, which becomes increasingly complex as the data complexity rises. Additionally, SQL is best suited for structured data while MongoDB is better for a semi-structured or non-structured data like the data considered in this project. MongoDB allows for the storage of JSON files, which are readable and language agnostic. It also allows for the storage of multiple data types and doesn't require strict adherence to the table structure of an SQL database. This is why MongoDB is a better fit for this use case.

5 Conclusions and Future Work

5.1 Future Work

There are a few very important changes that would drastically improve this project. First and foremost is the implementation of a visual interface. While this was not a priority during development due to time constraints and the emphasis on functionality over appearance, it would simplify the operation of the pipeline and improve the user experience. Additionally, standardising authentication, as mentioned in Section 4, is the second most important improvement. Creating a system parallel to the one used for authors but for interface endpoints is the first step towards a more secure connection between the open endpoint and the desired interface. This also brings up another point that is often overlooked during early stages of development: making the system resistant to a malicious user. While the code currently only does a limited set of things, once the system expands, the functionality given to the user can break the entire structure if permissions are not managed properly. This can be seen with the purge function, which was implemented for development only. It was ensured that this function could be disabled by restricting the database client permissions during setup. However, keeping this function in a release version raises important concerns about maintaining the safety of the user's data if any more of these functions are implemented. This risk was minimized by restricting the scope of the user and raising appropriate exceptions during execution.

The final important issue to address in future development is the idea that the user can be just as detrimental to the data as badly tested code. This pipeline assumes that the person using it has good intentions towards the data they possess. However, this cannot be assumed once data is shared across multiple users, so it is very important to standardize permissions and implement

them as part of authentication to prevent one user from destroying another user's work. Additionally, the pipeline currently only handles numerical data types. This means that storage of images was not implemented, even though it is technically possible. Extending the pipeline to include image storage would allow for more complex analysis and provide additional functionality.

5.2 Conclusion

In conclusion, the project successfully transformed the existing API framework into a functional data analytics pipeline that can be used in the analysis of experimental data. The architecture was transformed into a usable dashboard with broad functionality, improving its ease of use. Throughout the development process, the application towards material science was prioritized and the principles of data management were followed. While there is still room for improvement and additional work is needed to make the pipeline more efficient, this project demonstrates that a data analytics pipeline can be implemented and used in high-throughput experiments. Overall, the project can be considered a success.

References

- [1] T. Hey, K. M. Tolle, S. Tansley, and A. Hey, *The Fourth Paradigm Data-intensive Scientific Discovery*. Microsoft Research, 2010 (cited on p. 5).
- [2] F. C. R. Rademakers, "Root-project/root: V6.18/02," *Zenodo*, 2019. [Online]. Available: <https://zenodo.org/record/3895860> (cited on pp. 5, 9).
- [3] T. J. Jacobsson, A. Hultqvist, A. García-Fernández, *et al.*, "An open-access database and analysis tool for perovskite solar cells based on the fair data principles," *Nature Energy*, vol. 7, pp. 107–115, 1 Dec. 2021, ISSN: 2058-7546. DOI: 10.1038/s41560-021-00941-3 (cited on p. 5).
- [4] N. Donratanapat, S. Samadi, J. Vidal, and S. S. Tabas, "A national scale big data analytics pipeline to assess the potential impacts of flooding on critical infrastructures and communities," *Environmental Modelling & Software*, vol. 133, p. 104828, Nov. 2020, ISSN: 13648152. DOI: 10.1016/j.envsoft.2020.104828 (cited on p. 5).

- [5] F. Idachaba and M. Rabiei, "Current technologies and the applications of data analytics for crude oil leak detection in surface pipelines," *Journal of Pipeline Science and Engineering*, vol. 1, pp. 436–451, 4 Dec. 2021, ISSN: 26671433. DOI: 10.1016/j.jpse.2021.10.001 (cited on p. 5).
- [6] M. Das and S. K. Ghosh, "Data-driven approaches for meteorological time series prediction: A comparative study of the state-of-the-art computational intelligence techniques," *Pattern Recognition Letters*, vol. 105, pp. 155–164, Apr. 2018, ISSN: 01678655. DOI: 10.1016/j.patrec.2017.08.009 (cited on p. 5).
- [7] S. Braverman, "Global review of data-driven marketing and advertising," *Journal of Direct, Data and Digital Marketing Practice*, vol. 16, pp. 181–183, 3 Jan. 2015, ISSN: 1746-0166. DOI: 10.1057/dddmp.2015.7 (cited on p. 5).
- [8] UKRN, *Uk reproducibiltiy network - webpage*, 2022. [Online]. Available: <https://www.ukrn.org/about/> (cited on p. 5).
- [9] UCAR/Unidata Program Center, "Integrated data viewer (idv) software," 2022. DOI: <http://doi.org/10.5065/D6H70CW6> (cited on p. 6).
- [10] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, *et al.*, "The fair guiding principles for scientific data management and stewardship," *Scientific Data*, vol. 3, p. 160 018, 1 Dec. 2016, ISSN: 2052-4463. DOI: 10.1038/sdata.2016.18 (cited on pp. 6, 10, 12).
- [11] S. M. Mennen, C. Alhambra, C. L. Allen, *et al.*, "The evolution of high-throughput experimentation in pharmaceutical development and perspectives on the future," *Organic Process Research & Development*, vol. 23, pp. 1213–1242, 6 2019. DOI: 10.1021/acs.oprd.9b00140 (cited on p. 6).
- [12] C. Akin, L. C. Feldman, C. Durand, *et al.*, "High-throughput electrical measurement and microfluidic sorting of semiconductor nanowires," *Lab on a*

Chip, vol. 16, pp. 2126–2134, 11 2016, ISSN: 1473-0197. DOI: 10 . 1039 / C6LC00217J (cited on p. 6).

- [13] S. Arab, M. Yao, C. Zhou, P. D. Dapkus, and S. B. Cronin, “Doping concentration dependence of the photoluminescence spectra of n-type gaas nanowires,” *Applied Physics Letters*, vol. 108, p. 182 106, 18 May 2016, ISSN: 0003-6951. DOI: 10 . 1063/1 . 4947504 (cited on p. 6).
- [14] N. Wang, Y. Cai, and R. Zhang, “Growth of nanowires,” *Materials Science and Engineering: R: Reports*, vol. 60, pp. 1–51, 1-6 Mar. 2008, ISSN: 0927796X. DOI: 10 . 1016/j . mser . 2008 . 01 . 001 (cited on p. 6).
- [15] T. Neska and P. Parkinson, *Resdata: Python api managing data storage using mongodb database*, 2022. [Online]. Available: [https : / / github . com / SplitSky/resdata](https://github.com/SplitSky/resdata) (cited on p. 6).
- [16] D. Mike, *Pymongo documentation*, 2022. [Online]. Available: [https : / / pymongo . readthedocs . io/en/stable/index.html#overview](https://pymongo.readthedocs.io/en/stable/index.html#overview) (cited on pp. 8, 10).
- [17] P. Yang, R. Yan, and M. Fardy, “Semiconductor nanowire: What’s next?” *Nano Letters*, vol. 10, pp. 1529–1536, 5 May 2010, ISSN: 1530-6984. DOI: 10 . 1021/ nl100665r (cited on p. 9).
- [18] W. W. Wong, Z. Su, N. Wang, C. Jagadish, and H. H. Tan, “Epitaxially grown inp micro-ring lasers,” *Nano Letters*, vol. 21, pp. 5681–5688, 13 Jul. 2021, ISSN: 1530-6984. DOI: 10 . 1021/acs . nanolett . 1c01411 (cited on p. 9).
- [19] C. S, *Pydantic documentation*, Sep. 2022. [Online]. Available: [https : / / pypi . org/project/pydantic/](https://pypi.org/project/pydantic/) (cited on p. 9).
- [20] R. K, *Requests: A simple, yet elegant http library*, 2022. [Online]. Available: <https://github.com/psf/requests> (cited on p. 10).

- [21] S. E. Peyrott, *Json web token*, Accessed on 8/12/2022, 2022. [Online]. Available: <https://jwt.io/introduction> (cited on p. 11).
- [22] N. Wang, X. Yuan, X. Zhang, *et al.*, “Shape engineering of inorganic nanostructures by selective area epitaxy,” *ACS Nano*, vol. 13, pp. 7261–7269, 6 Jun. 2019, ISSN: 1936-0851. DOI: 10.1021/acsnano.9b02985 (cited on pp. 13, 16).
- [23] H. Krekel, B. Oliveira, and R. Pfannschmidt, *Pytest 7.2.0*, Oct. 2022. [Online]. Available: <https://pypi.org/project/pytest/> (cited on p. 17).
- [24] G. Heber, *Hdf5 1.13.4-1 documentation*, 2022. [Online]. Available: <https://docs.hdfgroup.org/hdf5/develop/> (cited on p. 17).

Appendices

A Libraries and technical details

This architecture uses libraries to allow for its functionality. The libraries and their purpose are listed in Table 2

Library name	Library function
Pydantic	Objects that validate data contained within them. Allows to specify the type of variable to be enforced
Pymongo	Interface used for communication between the API and the database
FastAPI	Tools used to write functions callable by HTTP
requests	Library for making HTTP calls using Python
typing	Library containing data types
hashlib	Library containing hashing and security algorithms. Used in authentication
matplotlib	Data plotting and display
Numpy	Numerical calculations and statistical functions
datetime	Library used for operating on date and time variables
jose	Tools for managing and generating JWT tokens
secrets	Contains functions concerned with data security. Used in Authentication

Table 2. Table summarising the libraries used.

B Testing and Continuous development cycle

The testing has been conducting using the Pytest library. All tests have been successful as of May 16, 2024. The tests and their purpose are listed below. The testing functions are within a file named testing_interface.py.

Test number	Test purpose
0	Check the connection of the API to the database
1	Verify that a user is created
2	Insert a project using previously created user
3	Insert a project. Then return it and compare to the one inserted
4	Authenticate a user and return a JWT token
5	Tree print the data associated with a user after authentication
6	Adding authors and verifies an author was added successfully
7	Tests that the generate_optics_project generated the Ring object successfully
8	Verifies that the search by meta variable works
9	Tests the group feature and adding authors to the group
10	Tests the author query function
11	Testing the add author to group functions on the level of the database, experiment and project

Table 3. Table summarising the tests conducted.

C Glossary

List of words that are defined for the need of this report:

- data pipeline - A set of processes used to automate the movement and transformation of data between a source system and a target repository.
- pull/fetch - To request and receive data from a data repository
- Node - A point in a tree structure where data is contained.
- API - Application Programming Interface - In the context of APIs, the word Application refers to any software with a distinct function. Interface can be thought of as a contract of service between two applications. It refers to a piece of code operating as a handler between two nodes of a data journey.
- JSON - JavaScript Object Notation - A type of data storage by the use of an often nested dictionary.
- Version control - It's a system of software that helps the developer track changes in code over time. As a developer edits code, the version control takes a snapshot of the files. It then saves the snapshot permanently so it can be recalled later if needed.
- salt - Random input which is used as an additional input to a one-way function that hashes the data.
- data custody - It refers to the process of having the legal right and authentic control over particular set of data elements which are then authorized for storage and use by any particular custodian of that data.
- data redundancy - The practice of keeping data in two or more places within a database or data storage system.
- dynamic schema - A changeable representation of data.
- schema - A representation of a plan or theory in the form of an outline or model.
- open source - Denoting software for which the original source code is made freely available and may be redistributed and modified.
- data journey - The data journey represents the key stages of the data process. The journey is not necessarily linear; it is intended to represent the different steps and activities that could be undertaken to produce meaningful information from data.
- compare digest - A method of comparing two arrays of bits. This function uses an approach designed to prevent timing analysis by avoiding content-based short circuiting behaviour. This makes it appropriate for cryptography.

D Jupyter Notebook

The entire testing notebook used to demonstrate the functionality of the pipeline:

```
In [6]: ## Sign in and Authentication
username = 'test_user_main'
password = 'some_password'
email = 'email@test_email.com'
full_name = 'Tomasz Neska'
path = 'http://18.200.229.75:8080/'
file_name = 'test.json'
# declaration of variables used

In [7]: # initialisation cell
import sys
sys.path.insert(0, "/home/splitsky/Desktop/code_repositories/resdata/")
import interface as ui
api = ui.API_interface(path)
api.check_connection()
#purge call - comment out if necessary -> just for development
#api.purge_everything()

Out[7]: True

In [8]: # make user and populate the database with data
result = api.create_user(username_in=username, password_in=password, email=email, full_name=full_name)
print("User created " + str(result))

User created True
```

Fig. 8. Cell 1: The definition of the variables. Cell 2: Initialisation cell which declares the objects used. Cell 3: User creation.

```
In [8]: # populate the database
no_of_rings = 5
no_of_experiments = 2

import testing as t
api.generate_token(username, password)
project_name = "project_test_"
experiment_name = "experiment_test"
ds_size = 2000
for i in range(0,4,1):
    t.generate_optics_project(file_name,[no_of_rings, no_of_experiments], project_name+str(i),
                             experiment_name, author_name=username, size_of_dataset=ds_size)
    project = t.load_file_project(file_name)
    api.insert_project(project)

# generate special project for analysis showcase
project_name = "special_project"
experiment_name = "special_experiment_test"
ds_size = 5000

search_value = 15.65
t.generate_optics_project_2(file_name, [1,1], project_name, experiment_name, username, ds_size, search_value)
project = t.load_file_project(file_name)
api.insert_project(project)

print("Database populated")
print("")
api.tree_print()
```

Fig. 9. Cell 4: Populates the database.

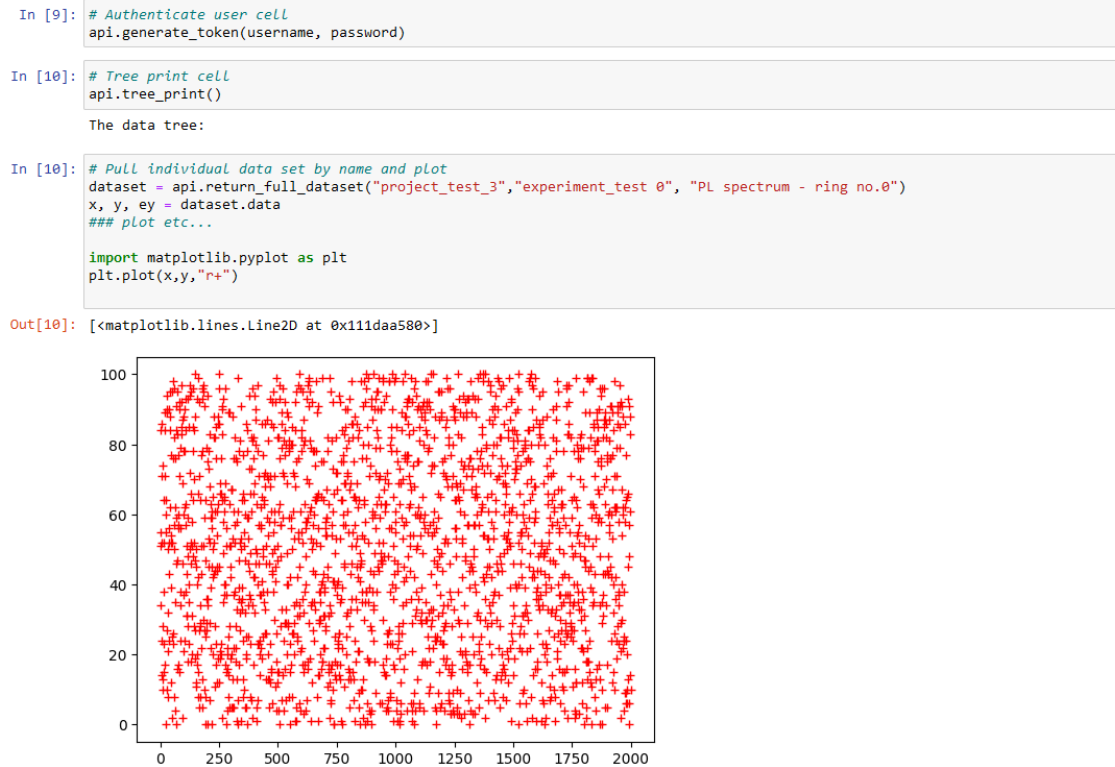


Fig. 10. Cell 5: Token generation. Cell 6: Tree print - data visualisation. Cell 7: Pulls individual dataset and plots the data.

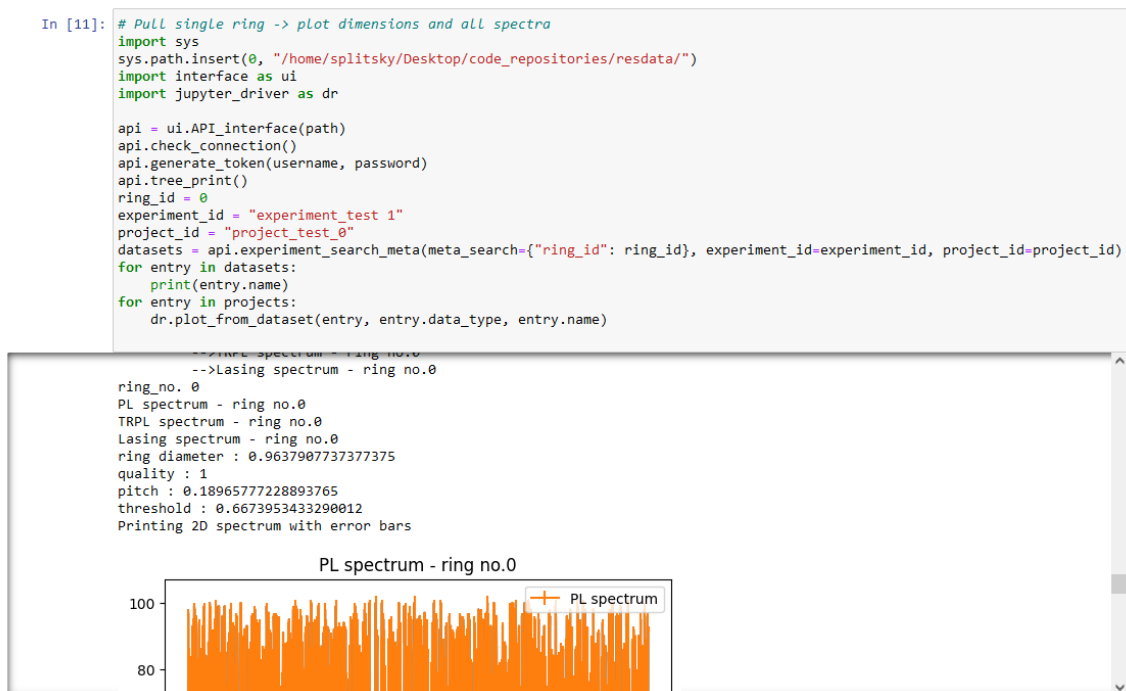


Fig. 11. Cell 8: Pulls single ring and summarises variables and plots spectra.

```

In [12]: # Pull all rings from experiment and summarise dimensions
import jupyter_driver as dr
# pull full experiment
experiment = api.return_full_experiment(project_name="project_test_0", experiment_name="experiment_test 0")
# separate dimensions
datasets = experiment.children
desired_data = []
for ds in datasets:
    if ds.data_type == "dimensions":
        desired_data.append(ds)

# summarise the quantities
dr.summarise_dimensions(desired_data)

Average values for dimensions:
ring diameter : 0.7422914029193357 +/- 0.3187223925063638
quality : 6.4 +/- 3.3823069050575523
pitch : 0.4082606383008508 +/- 0.2555024488839914
threshold : 0.6063285670392031 +/- 0.28511286866429963

In [13]: # Create a group and pull it together to summarise ring dimensions

group_name = "very_important_things"
group_permission = "write" # or read is recognised
# project_id, author_name, author_permission, group_name
# the function prints true or false depending whether it succeeded on adding the group to the project/experiment/dataset
# _rec versions of the function add to groups recursively

print(api.add_group_to_project_rec("project_test_0", username, group_permission, group_name))
print(api.add_group_to_experiment_rec("project_test_1", "experiment_test 0", username, group_permission, group_name))
print(api.add_group_to_dataset_rec(group_permission, username, group_name, "project_test_2", "experiment_test 1", "TRPL spec")
<
True
True
True

```

Fig. 12. Cell 9: Pulls all ring datasets connected by the same ring id. Cell 10: Creates a group and adds a dataset, an experiment, a project.

```

In [14]: # Pull group together to summarise TRPL spectra from the datasets

# use function which can be used with the group name or author name
structure = api.author_query(group_name)

api.tree_print_group(group_name)

```

```

project_test_0
-> experiment_test 1
--> ring_no. 0
--> PL spectrum - ring no.0
--> TRPL spectrum - ring no.0
--> Lasing spectrum - ring no.0
--> ring_no. 1
--> PL spectrum - ring no.1
--> TRPL spectrum - ring no.1
--> Lasing spectrum - ring no.1
--> ring_no. 2
--> PL spectrum - ring no.2
--> TRPL spectrum - ring no.2
--> Lasing spectrum - ring no.2
--> ring_no. 3
--> PL spectrum - ring no.3
--> TRPL spectrum - ring no.3
--> Lasing spectrum - ring no.3
--> ring_no. 4
--> PL spectrum - ring no.4
--> TRPL spectrum - ring no.4
--> Lasing spectrum - ring no.4
-> experiment_test 0
--> ring_no. 0
--> PL spectrum - ring no.0
--> TRPL spectrum - ring no.0
--> Lasing spectrum - ring no.0
--> ring_no. 1
--> PL spectrum - ring no.1
--> TRPL spectrum - ring no.1
--> Lasing spectrum - ring no.1
--> ring_no. 2
--> PL spectrum - ring no.2
--> TRPL spectrum - ring no.2
--> Lasing spectrum - ring no.2
--> ring_no. 3
--> PL spectrum - ring no.3
--> TRPL spectrum - ring no.3
--> Lasing spectrum - ring no.3
--> ring_no. 4
--> PL spectrum - ring no.4
--> TRPL spectrum - ring no.4
--> Lasing spectrum - ring no.4

```

Fig. 13. Cell 11: Pulls the nodes connected within a group.