ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

PHYLOViZ Web Platform

A Modular and Web-Based Tool for Phylogenetic Analysis

André Jesus
André Páscoa
Nyckollas Brandão

Supervisors: Cátia Vaz, ISEL
Alexandre P. Francisco, IST

Final report written for Project and Seminary
BSc in Computer Science and Computer Engineering

June 2023

Instituto Superior de Engenharia de Lisboa

PHYLOViZ Web Platform

A Modular and Web-Based Tool for Phylogenetic Analysis

48089 André Filipe Pina Páscoa

48280 André Filipe do Pilar de Jesus

48287 Nyckollas Brandão

Supervisors: Cátia Vaz, ISEL

Alexandre P. Francisco, IST

Final report written for Project and Seminary
BSc in Computer Science and Computer Engineering

June 2023

Abstract

Modern biomedical research has witnessed a remarkable increase in the production and computational analysis of large datasets, leading to an urgent need of tools for data integration, processing and visualization, as well as the need of sharing standardized analytical techniques. Phylogenetic analysis plays a crucial role in biomedical research, enabling the understanding of the evolution of bacterial and viral epidemics, including their origin, evolution, and resistance to various treatments.

Several approaches to support phylogenetic analysis have been proposed, varying from standalone tools to web applications that include tools and/or algorithms for executing the common analysis tasks for these kind of data. PHYLOViZ and PHYLOViZ Online provide phylogenetic analysis processing, namely inference methods, visualization and integration of epidemiological and ancillary data. Nevertheless, features in both PHYLOViZ versions are not the same, and the web version is not modular, making it difficult to extend and maintain.

To address these limitations, we present PHYLOViZ Web Platform, a web-based tool for phylogenetic analysis, with a modular architecture. This platform unifies both versions of PHYLOViZ and supports structured and customized workflows for data processing and analyses tasks. It promotes the reproducibility of previous phylogenetic analyses through its management of projects, and supports integration of recently developed and future tools, remaining flexible, extensible, and maintainable. The platform supports large scale analyses by relying on a workflow system that enables the distribution of parallel computations on cloud and HPC environments. Its modular architecture allows seamless integration of new methods, tools, and customized workflows.

Powered by modern open-source technologies, PHYLOViZ Web Platform is designed to be maintained and extended in the future. It enables users to access and perform phylogenetic analyses from anywhere with an internet connection, eliminating the need for software installation or access to HPC resources.

Keywords: phylogenetic analysis; data processing and visualization; modular architecture; data-centric workflows.

Resumo

A investigação biomédica moderna testemunhou um aumento notável na produção e análise computacional de grandes conjuntos de dados, levando a uma necessidade urgente de ferramentas para integração, processamento e visualização de dados, bem como à necessidade de compartilhar técnicas analíticas padronizadas. A análise filogenética desempenha um papel crucial na investigação biomédica, permitindo a compreensão da evolução de epidemias bacterianas e virais, com o objetivo de determinar sua origem, evolução e resistência aos vários tratamentos em estudo.

Várias abordagens para suportar análise filogenética têm sido propostas, variando de ferramentas autónomas a aplicações web que incluem ferramentas e/ou algoritmos para executar tarefas de análise comuns para este tipo de dados. PHYLOViZ e PHYLOViZ Online fornecem processamento de análise filogenética, nomeadamente métodos de inferência, visualização e integração de dados epidemiológicos e auxiliares. No entanto, as funcionalidades em ambas as versões do PHYLOViZ não são as mesmas, e a versão web não é modular, dificultando a sua extensão e manutenção.

Para resolver essas limitações, apresentamos o PHYLOViZ Web Platform, uma ferramenta web para análise filogenética, com uma arquitetura modular. Esta plataforma unifica as duas versões do PHYLOViZ e oferece suporte a fluxos de trabalho estruturados e personalizados, para processamento e análise de dados. Promove a reprodutibilidade de análises filogenéticas através da sua gestão de projetos e suporta a integração de ferramentas desenvolvidas recentemente e futuras, mantendo-se flexível, extensível e sustentável. A plataforma oferece suporte para análises em larga escala, contando com um sistema de fluxo de trabalho que permite a distribuição de computações paralelas em ambientes de nuvem e de HPC. A sua arquitetura modular permite a integração de novos métodos, ferramentas e fluxos de trabalho personalizados.

Suportado por tecnologias modernas open-source, o PHYLOViZ Web Platform foi projetado para ser mantido e estendido no futuro. Este permite que os utilizadores acedam e realizem análises filogenéticas em qualquer lugar com conexão à Internet, eliminando a necessidade de instalação de software ou acesso a recursos de HPC.

Palavras-chave: análise filogenética; processamento e visualização de dados; arquitetura modular; fluxos de trabalho centrados nos dados.

Contents

List of Figures

List of Tables

List of Acronyms

API	Application Programming Interface
AWS	Amazon Web Services
CSV	Comma-Separated Values
CI	Continuous Integration
CSRF	Cross-site request forgery
DB	Database
DNA	Deoxyribonucleic acid
DOM	Document Object Model
DSL	Domain Specific Language
eBURST	electronic Based Upon Related Sequence Type
HPC	High-Performance Computing
HTTP	Hypertext Transfer Protocol
IAM	Identity and Access Management
ID	IDentifier
iMM	Institute of Molecular Medicine
JS	JavaScript
JSON	JavaScript Object Notation
goeBURST	global optimal eBURST
MLST	Multi Locus Sequence Typing
MLVA	Multi Locus Variable Number Tandem Repeat Analysis
OIDC	OpenID Connect
REST	REpresentational State Transfer
SAML	Security Assertion Markup Language
ST	Sequence Type
*S*3	Simple Storage Service
SNP	Single Nucleotide Polymorphism
TS	TypeScript
UAT	User Acceptance Testing
UI	User Interface
UML	Unified Modeling Language

$UUID$	Universally Unique IDentifier
WMS	Workflow Management System

Chapter 1

Introduction

Phylogenies, or evolutionary trees, are fundamental structures for understanding species differences and analyzing them statistically. While phylogenies have been around for over 140 years, the statistical, computational, and algorithmic aspects of their analysis are relatively young, with only 40 years of active research. In modern biomedical research, the need for tools that integrate, process, and visualize data has become urgent due to the substantial increase in both data production and computational analysis of large datasets [1]. This need is particularly evident in the field of phylogenetic analysis, which plays a crucial role in understanding the origin and evolution of bacterial and viral epidemics.

Phylogenetic analysis aims to uncover the evolutionary relationships between species or taxa, providing insights into the evolution of life on Earth. These analyses heavily rely on phylogenetic trees, which are typically computed from molecular sequences. Additionally, phylogenetic trees have diverse applications in various domains. For example, they are utilized to determine the age and rate of diversification of taxa, understand the evolutionary history of gene families, facilitate phylogenetic "footprinting" in sequence analysis methods, trace the origin and transmission of infectious diseases in epidemiology, and explore the co-evolution of hosts and parasites [2].

Numerous approaches and tools have been proposed to support phylogenetic analysis, ranging from standalone software to integrative web applications that incorporate various tools and algorithms for common analysis tasks. The PHYLOViZ software, developed by the PHYLOViZ team for the iMM (Instituto de Medicina Molecular), and its web version, PHYLOViZ Online, offer phylogenetic analysis capabilities, inference methods, visualization, and integration of epidemiological and ancillary data [3, 4, 5, 6]. However, there are discrepancies between the features of both versions, and the web version lacks modularity, which hinders its extensibility and maintainability.

Moreover, performing inference and visualization optimization tasks on the client side often becomes infeasible for large datasets, requiring data transfer from existing databases. To address these limitations, the prototype phyloDB was developed to efficiently manage phylogenetic data, supporting algorithm deployment for pattern inference/detection and pre-computing and optimizing phylogenetic visualizations. Another prototype, FLOWViZ, was created to support the usage of a workflow system for data processing [7, 8].

This project aims to develop the PHYLOViZ Web Platform, a new web-based tool for phylogenetic analysis. The platform adopts a modular architecture [9], integrating both versions of PHYLOViZ and facilitating the use of additional tools such as phyloDB and FLOWViZ, not being compromised by these.

By operating on the web, the PHYLOViZ Web Platform allows users to access and perform phylogenetic analyses and visualizations from anywhere with an internet connection, eliminating the need for software installation or access to high-performance computing resources. This is made possible by performing most computing operations and data management on the server-side.

To support large-scale analyses, the platform relies on a workflow system that enables parallel computations distributed across cloud and high-performance computing environments. The platform offers extensive customization of these workflows for data processing and analysis tasks, allowing for the replacement and alteration of workflows and tasks even during runtime.

Furthermore, the platform promotes the reproducibility of previous phylogenetic analyses through its management of personal projects. It efficiently stores and organizes phylogenetic data, ensuring easy retrieval of analysis results, including visualizations and applied transformations. Additionally, it provides valuable information about the data and its relationships within the dataset. This comprehensive approach enhances the transparency and reliability of phylogenetic analyses, fostering an environment conducive to scientific rigor and further advancements in the field.

The PHYLOViZ Web Platform also supports the integration of recently developed and future prototypes. For example, FLOWViZ can enable the creation and execution of customized analysis workflows, while phyloDB provides efficient storage and retrieval of large phylogenetic datasets. The system is designed to remain flexible, extensible, and maintainable across its various components, ensuring compatibility and scalability throughout its architecture.

1.1 Outline

This document consists of eight chapters that collectively provide a comprehensive understanding of the PHYLOViZ Web Platform.

Chapter 2 introduces the reader to the domain of phylogenetic analysis, explaining key concepts and techniques. It also includes a running example that illustrates these concepts and techniques throughout the document.

Chapter 3 presents a detailed description of the problem domain, discussing the challenges faced and the requirements for an effective solution. It outlines our approach to solving the problem and provides an overview of existing solutions, highlighting the need for the PHYLOViZ Web Platform.

Chapter 4 provides an in-depth overview of the PHYLOViZ Web Platform's architecture, emphasizing its key components and their interactions. It delves into how the architecture effectively addresses the goals outlined in Chapter 3, including robust data management, efficient access control, and seamless extensibility.

Chapter 5 introduces the software stack employed in the PHYLOViZ Web Platform, explaining the selection of specific technologies and frameworks. It highlights the rationale behind these choices and discusses how they contribute to the platform's robustness and performance.

Chapter 6 delves into the implementation details of each component that constitutes the PHYLOViZ Web Platform. It discusses their functionalities, interactions, and deployment processes. Additionally, it explores the extensibility of the platform, demonstrating how researchers can customize and expand its capabilities to suit their specific needs.

Chapter 7 focuses on the testing and evaluation process conducted on the PHYLOViZ Web Platform. It examines the performance differences between various implementations, substantiating design decisions and highlighting key features. Furthermore, it provides a comparative analysis of the platform's performance and features in relation to its competitors.

Chapter 8 presents final remarks and outlines our vision for the future of the PHYLOViZ Web Platform. It reflects on the accomplishments of the project, identifies potential areas for further improvement, and discusses the platform's potential impact on phylogenetic analysis.

Chapter 2

Background

This chapter provides an overview of the phylogenetic analysis domain, namely the concepts and techniques used in this field. A running example will also be presented to illustrate the concepts and techniques presented.

Phylogenetics is the study of the evolutionary history and relationships of single or groups of organisms. These relationships are found using phylogenetic inference techniques, which analyze reported heritable features like deoxyribonucleic acid (DNA) sequences or morphology using an evolutionary model [10]. The parameters used to characterize the rates at which one nucleotide replaces another throughout evolution vary between these models, which attempt to represent the development of the species from which a sequence of symbols evolves into another set of characteristics. This enables us to infer evolutionary events that happened in the past, and also provides more information about the evolutionary processes operating on sequences.

2.1 Phylogenetic Analysis

Phylogenetic analysis [2] aims at uncovering the evolutionary relationships between different species, or even between individuals of the same species, to obtain an understanding of their evolution. The result of this analysis is a phylogeny, which can be a phylogenetic tree or network, that is a diagrammatic hypothesis about the history of the evolutionary relationships of a group of organisms. In this project, we will focus on microbial phylogenetic analysis.

Phylogenetic analyses includes in general the following steps: 1) the alignment of genetic sequences, 2) the application of a typing methodology, 3) the application of phylogenetic inference methods, and finally 4) the visualization and exploration of inference results, integrating epidemiological and ancillary data for isolates under study.

2.1.1 Phylogenetic Data

The first step of phylogenetic analysis is the alignment of genetic sequences [11]. This is done in order to assembly the genomes of the organisms under study. This process can be defined as the reconstruction of the organism genome system with random small parts of it to determine the nucleic acid sequence, that is, the order of nucleotides in the DNA. It allows to map genomes of new organisms, finish genomes of previously known organisms, or to compare genomes across multiple samples.

Locus and Alleles

The sequences assembled in the alignment process may occupy a given position of a locus and define distinct alleles of that locus. A locus is a specific location in the chromosome, and every unique sequence, either DNA or peptide depending on the locus, is defined as a new allele. An allele can also be defined as a viable DNA coding sequence for the transmission of traits, and it is represented with a number identifying the allele and string containing the sequence.

It is expected that the alleles are represented through files which usually follow the FASTA format. FASTA is a text-based format for representing either nucleotide sequences or amino acid sequences, in which nucleotides or amino acids are represented using single letter codes. An example of this format can be found in Figure 2.1.

```
> Sequence 1
GAAGCGAGTGACTTGGCAGAAACAGTGGCCAATATTCGTCGCTACCAGATGTTTGGCATC
GCGCGCTTGATTGGTGCGGTTAATACGGTTGTCAATGAGAATGGCAATTTAATTGGATAT
> Sequence 2
GAACCGAGTGACTTGGCAGAAACAGTGGCCAATATTCGTCGCTACCAGATGTTTGGCATC
GCGCGCTTGATTGGTGCGGTTAATACGGTTGTCAATGAGAATGGCAATTTAATTGGATAT
```

Figure 2.1: Example of two sequences in FASTA format. The figure demonstrates two sequences represented in the FASTA format, labeled as Sequence 1 and Sequence 2. Each sequence consists of a string of nucleotides and is preceded by a header line starting with a greater-than symbol (>). The sequences are stored in a text file and follow the conventions of the FASTA format, which uses single letter codes to represent nucleotides or amino acids.

2.1.2 Typing Methodology

After the alignment phase, a typing methodology [12] is applied to identify each organism based on the genes that are presented in almost all organisms. This process provides the means to execute phylogenetic inference methods, which then produces a hypothesis about the history of the evolutionary relationships about a group of organisms. There are several typing methodologies, such as the MLST, MLVA, and SNP. The result of this process is called typing data.

Typing data refers to the allelic profiles of different loci (such as genes or tandem repeats) that are used to assign a type to each bacterial isolate. The typing data is usually the result of a molecular typing technique, such as Multi Locus Sequence Typing (MLST) or Multi Locus Variable Number Tandem Repeat Analysis (MLVA). It is usually stored in a tabular format, where each row represents a profile verified in an isolate, and each column represents a locus. There may be more than one isolate with this profile. The values in the table are the allele number for each locus in each isolate. The Figure 2.2 shows an example of MLST profiles.

ST	*nusA*	*rpoB*	*eno*	*gltB*	*lepA*	*nuoL*	*aroe*
1	1	26	2	2	59	8	1
2	1	26	2	4	59	2	1
3	1	26	2	2	62	8	2
4	1	26	7	2	59	3	2
5	1	27	1	1	62	9	1

Figure 2.2: Example of five sequences in MLST format. Each row in the table corresponds to a profile verified in an isolate, and each column represents a locus. The profiles are identified by a Sequence Type (ST) followed by the allele numbers for each locus. In the example, the profiles with ST 1, 2, 3, 4, and 5 are associated with specific combinations of alleles for the loci `nusA`, `rpoB`, `eno`, `gltB`, `lepA`, `nuoL`, and `aroe`. The numbers below each locus indicate the allele identifiers, which map to different alleles.

Single nucleotide polymorphisms (SNPs) [13] are polymorphisms that are caused by point mutations that give rise to different alleles containing alternative bases at a given position of nucleotide within a locus. The SNP format represents each sequence by a line with a sequence of 1's and 0's preceded by a number that identifies the sequence. A value of 0 in any location represents the character state that was mostly found on that location, while a 1 represents any other possible character state. An example of this format can be found in Figure 2.3.

```
1 01000001111010100010001010101010010101000111010110001010 10
2 11110100101011001001010101010010000010100100010101010101 00
```

Figure 2.3: Example of two sequences in SNP format. The sequences identifiers are '1' and '2'. Each sequence exhibits variations at multiple nucleotide positions within the locus.

Isolate and Ancillary Data

The main goal of the typing methods is the characterization of organisms existing in a given sample. Some microorganisms from the sample collected need to be isolated to be characterized. Thus, each organism isolated from the microbial population becomes an isolate.

An isolate can be associated with typing information and ancillary details. Ancillary details include information about the place where the microorganism was isolated, the environment or the host, and other possible contextual details. These details are usually represented through comma-separated values (CSV) formatted files, where the tab character is used to separate values.

An example of ancillary data is shown in Figure 2.4.

id	*ST*	*isolate*	*species*	*country*	*continent*
1	1	*AU*2523	*A.denitrificans*	*USA*	*NorthAmerica*
2	2	*AU*8059	*A.denitrificans*	*Unknown*	
3	3	*AU*8060	*A.denitrificans*	*France*	*Europe*
4	1	*AU*8080	*A.insolitus*	*USA*	*NorthAmerica*
5	5	*ACH*26	*A.insolitus*	*USA*	*NorthAmerica*

Figure 2.4: Example of ancillary data. The figure presents a table containing ancillary details associated with microbial isolates. Each row represents a specific isolate and includes information such as the isolate's unique identifier (id), Sequence Type (ST) indicating the profile, isolate name, species, country, and continent. Multiple isolates can be associated with the same profile, as shown by the isolates with id 1 and 4, associated with the same profile with ST 1.

2.1.3 Phylogenetic Inference

Succeeding the typing process follows the execution of a phylogenetic inference method to the results. A phylogenetic inference method [10] is the application of computational algorithms to phylogenetic data that allows to produce a diagrammatic hypothesis about the history of the evolutionary relationships of a group of organisms. There are several types of phylogenetic inference method, and some of them depend on the calculation of a distance matrix [14].

Distances

Distance matrix methods rely on the genetic distance between the sequences being classified. The distances are often defined as the fraction of mismatches at aligned positions, with gaps either ignored or counted as mismatches. An example of a distance matrix, resultant from applying the hamming distance between each profile of a typing data file, is shown in Figure 2.5.

$$D = \begin{bmatrix} 0 & 2 & 2 & 3 & 5 \\ 2 & 0 & 4 & 4 & 5 \\ 2 & 4 & 0 & 3 & 5 \\ 3 & 4 & 3 & 0 & 6 \\ 5 & 5 & 5 & 6 & 0 \end{bmatrix}$$

Figure 2.5: Example of distance matrix, resultant from applying the Hamming distance. Each row and column correspond to a specific sequence profile. The values in the matrix indicate the genetic distances between the profiles. For example, the value at row 1 and column 3 represents the distance between profile 1 and profile 3. In this case, they differ by a distance of 2, indicating two mismatches between the sequences.

Inference Algorithms

An inference algorithm is then executed to compute the diagrammatic hypothesis based on the distance matrix previously calculated. The diagrammatic hypothesis calculated comes in the form of a graph or a tree, but in this work we will only focus on algorithms that generate trees.

Usually, the process of phylogenetic inference is begun with a multiple sequence alignment. From this, one can pursue either a distance-based analysis, or a sequenced-based one [2]. If the algorithm depends on the calculation of a distance matrix, and in their visualization, each node represents the allelic profile, and the relationships between them are quantified by the distances.

The Globally Optimized eBURST (goeBURST) [15] algorithm is an example of an implementation of these algorithms.

Visualization Algorithms

After executing an inference algorithm, a visualization algorithm is executed to compute the optimal coordinates for each node of the received graph or tree. Afterwards, the coordinates are provided to a render framework which then presents each profile and relationship to a user interface.

A phylogenetic tree can be rooted or unrooted [1]. A rooted tree is a dendrogram that indicates the common ancestor, or ancestral lineage, of the tree. An example of this type of tree is present in Figure 2.6. An unrooted tree however makes no assumption about the ancestral line, and does not show the origin or “root” of the gene or organism in question. An example of this type of tree is present in Figure 2.7.

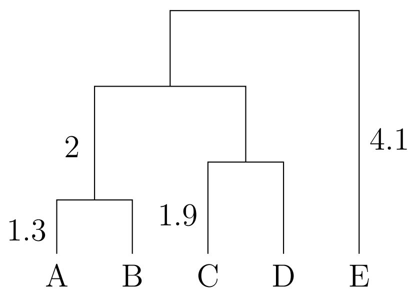


Figure 2.6: Example of a rooted phylogenetic tree. Each node in the tree represents a specific organism or gene, denoted by letters A, B, C, D, and E. The numbers attached to the branches represent the genetic distances between the connected nodes.

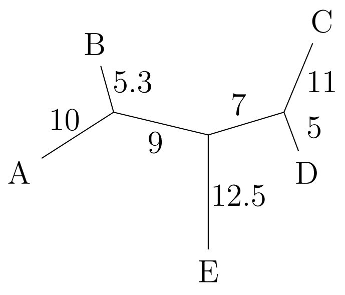


Figure 2.7: Example of an unrooted phylogenetic tree. Unlike a rooted tree, an unrooted tree does not indicate a common ancestor or ancestral lineage. The nodes in the tree correspond to organisms or genes labeled as A, B, C, D, and E. The numbers attached to the branches represent the genetic distances between the connected nodes.

2.2 Running Example

In this section we will present typical use cases of the expected phylogenetic analysis, that can be generally described as a sequence of processing steps (i.e, pipelines).

A specific use of phylogenetic analysis is the identification of the evolutionary relationships between species. This is done by comparing the DNA sequences of the species, and by using phylogenetic analysis tools to infer the evolutionary relationships between them.

After applying a typing methodology, it follows the tree inference step. If it is used an inference method based on distances, it starts by computing the distance matrix and then infer the phylogenetic tree from those calculated distances (distances can be also dynamically calculated). After this, a layout algorithm is used to generate a visual representation of the phylogenetic tree, which we will refer to as tree view.

A tree view can be obtained by using three different pipelines, represented in Figure 2.8.

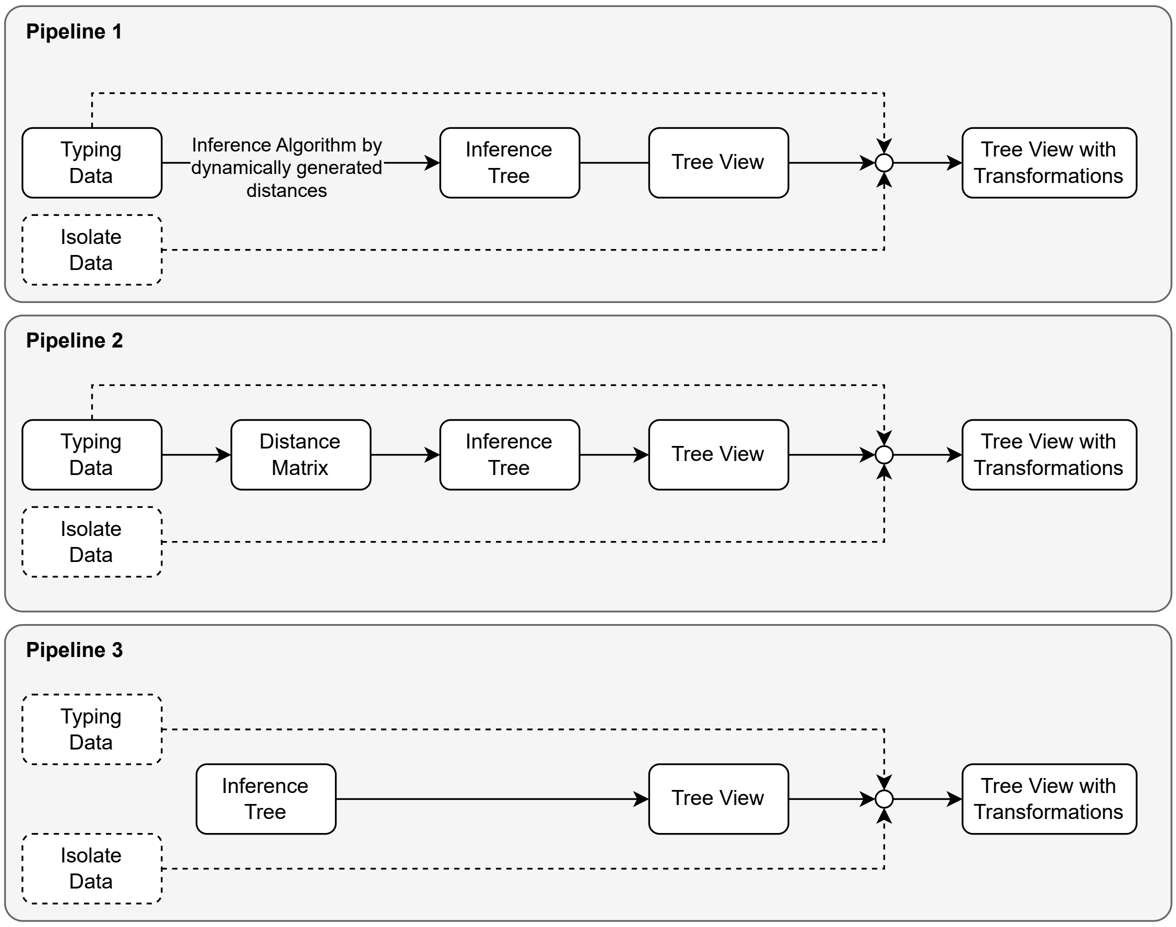


Figure 2.8: Examples of phylogenetic analysis pipelines.

As you can see, all pipelines generate a tree view using an inference tree. After obtaining the tree view with the specified layout, transformations can be applied to it which include the optional use of typing data and isolate data.

In the first pipeline, the inference tree is generated directly from the typing data, using distances dynamically generated in the inference algorithm.

The second pipeline uses a static distance matrix, which is generated from the typing data, and then the inference tree is generated from the distance matrix.

The third pipeline simply uses an existing inference tree.

In this running example we will use the second pipeline. The example of it is shown in Figure 2.9.

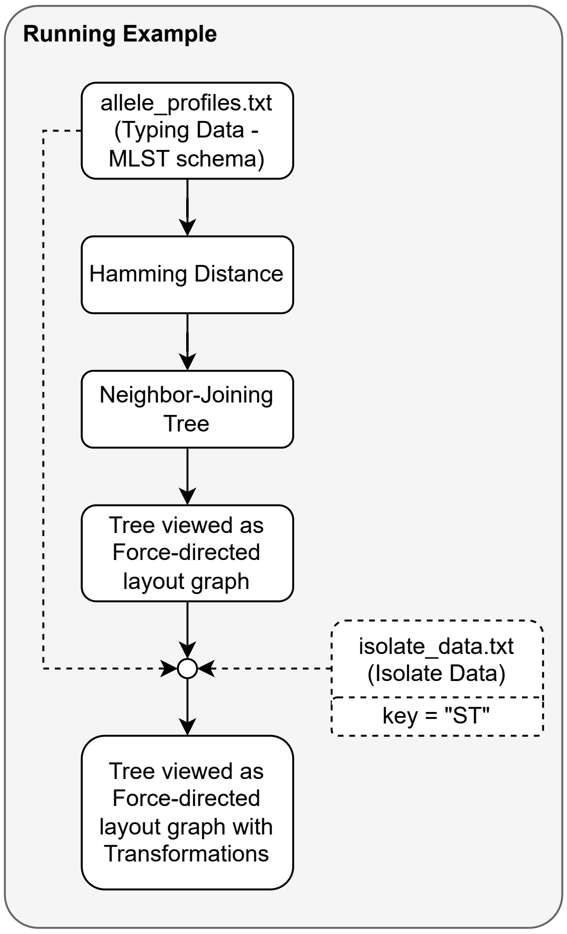

Running Example
allele_profiles.txt (Typing Data - MLST schema)
Hamming Distance
Neighbor-Joining Tree
Tree viewed as Force-directed layout graph
isolate_data.txt (Isolate Data)
key = "ST"
Tree viewed as Force-directed layout graph with Transformations

Figure 2.9: Pipeline representation of the running example.

The pipeline begins with the typing data generated by the MLST (Multi-Locus Sequence Typing) methodology. This data undergoes processing to compute a Hamming distance matrix. These distances may be subject to distance correction based on an appropriate model of evolution, also known as a substitution model.

The resulting distance matrix is then used in conjunction with the Neighbor-Joining algorithm to reconstruct a phylogenetic tree.

Subsequently, the tree is visualized using the force-directed layout based on the tree generated in the previous step.

Optional transformations can be applied to the visualization, including the integration of typing data, and isolate data if provided. If isolate data is provided, an isolate data key must also be provided. In this case, the column header ”ST” has been selected as the key within the isolate data, as it represents the associated profile identifier in the typing data according to the MLST schema.

Chapter 3

Problem Description

Phylogenetic analysis techniques raise some challenges since, depending on the tools, type and amount of data, some steps can take a long time to be executed. For instance, the alignment of sequences or the application of inference algorithms over large datasets can be time-consuming, consequence of the high time complexity of the operations over the large amounts of data. Other problem is the efficiency of the visualization process, since it involves processing and rendering large amounts of data, which can be unfeasible to be executed in its entirety on the client side, making it not scalable and not suitable for large datasets.

One important aspect of phylogenetic analysis is customization. This customization includes:

- What algorithms and parameters are used for the computations - various algorithms may yield different results, and thus reveal different aspects and relations of the profile.

- What layout is used for the tree visualization - the layout dictates how the tree is presented.

- Transformations to the visualization itself, which alter and enhance it to the user's needs. E.g., change color of profiles, show labels, organize and group profiles by common information.

Other challenge is the modularity [9] of the system. Modularity is an important feature for a software system, since it allows for the easy integration of new analysis methods and tools, making it flexible and extensible. This is a challenge because we need to design an entire new architecture, with replaceable components that can be easily integrated in the future. The modularity is a concept difficult to implement, since it requires the process of decomposition of a system into modules, and define the interfaces between them.

With this in mind, we think a good platform should address the above mentioned challenges by attending to the following objectives:

- Have a user friendly platform that allows users to access, manage and perform phylogenetic analyses and visualizations easily and intuitively, preserving data for future use;
- Provide a variety of customization options to the user, including the choice of what algorithms and parameters are used for the computations, what layouts for the tree visualization and different transformations to the visualization;
- Use a workflow management system, looking to efficiently control execution of workflows, that execute phylogenetic analysis tasks;
- Have an efficient and scalable visualization of trees, by using state of the art techniques for performance and/or controlling what parts of the tree are visualized at a time;
- Store and manage phylogenetic data efficiently, allowing for it to be scalable;
- Provide a modular architecture that allows for easy integration of new analysis methods and tools, making it flexible and extensible.

3.1 Related Tools

Several approaches to support phylogenetic analysis have been proposed, varying from standalone tools to integrative web applications that include tools and/or algorithms for executing the common analysis tasks for these kind of data. In this section we will present some of the most relevant tools and applications, developed by the PHYLOViZ [6] team, that are related to PHYLOViZ Web Platform, and their downsides and aspects to be improved.

PHYLOViZ

The PHYLOViZ [3, 4] desktop application is a good non-web option. Multiple customization options are available, including choice of inference algorithms, layouts and transformations on tree visualization.

However, since it is a desktop application, it performs all computations and data management client-side and all the data is stored in the local machine of the user. Furthermore, the visualization does not scale effectively for large datasets.

PHYLOViZ Online

PHYLOViZ Online [5, 16] was created as the web version of PHYLOViZ. It operates in the web, so there is no need to download software. It also provides a lot of transformations for the visualization. On the other hand, there are several points where it isn't optimal:

- It doesn't have inference algorithm customization: only has a "Launch Tree" option that doesn't allow the user to choose algorithms, nor exposes what algorithms were used;
- User isn't given the choice of layout, the only layout available is the force-directed layout;
- Just like the desktop version, the visualization is not efficient or scalable enough for large datasets;
- Lacks modularity and only uses a few specific algorithms.

FLOWViZ

FLOWViZ [8] is a middleware that allows you to seamlessly integrate phylogenetic platforms with workflow scheduling and execution, through the Apache Airflow workflow system. This tool is useful for PHYLOViZ Web Platform, since it allows the creation and execution of customized analysis workflows, which can be used to perform phylogenetic analysis tasks using different tools and algorithms.

PhyloDB

PhyloDB [7] is a prototype for efficient storage and retrieval of large phylogenetic datasets. It is a database built on top of the graph database Neo4j, with a data model specific for phylogenetic data, which allows for fast querying. This tool is useful for PHYLOViZ Web Platform, since it allows the storage and management of large datasets in a data model built specifically for phylogenetic data.

PhyloLib

PhyloLib [17] is a library of efficient algorithms for phylogenetic analysis in the form of a command line application. It can be useful, since it performs some inference tasks that are not supported in phyloDB.

3.2 Alternative Tools

The integration of the results obtained from inference algorithms with epidemiological data (also called isolate or ancillary data) and simultaneous analysis is still limited by visualization and processing techniques. Although, besides PHYLOViZ and PHYLOViZ Online, there are other known tools for visualizing and analyzing such data, allowing the integration of epidemiological data. We briefly describe some of the most known:

SplitsTree

SplitsTree [18] is a widely used application for computing unrooted phylogenetic networks from molecular sequence data. Given an alignment of sequences, a distance matrix or a set of trees, the program will compute a phylogenetic tree or network using methods such as split decomposition, neighbor-net, consensus network, super networks methods or methods for computing hybridization or simple recombination networks. The current release is SplitsTree4.

Phylogeny.fr

Phylogeny.fr [19] is a free, simple to use web service dedicated to reconstructing and analysing phylogenetic relationships between molecular sequences. Phylogeny.fr runs and connects various bioinformatics programs to reconstruct a robust phylogenetic tree from a set of sequences.

There is also an upgraded version of Phylogeny.fr, called NGPhylogeny.fr, that supports execution of tasks in a workflow system. NGPhylogeny.fr [20] is a webservice dedicated to phylogenetic analysis. It provides a complete set of phylogenetic tools and workflows adapted to various contexts and various levels of user expertise. It is built around the main steps of most phylogenetic analyses

However, both Phylogeny.fr and NGPhylogeny.fr have some limitations. Both of them do not allow the choice of the tree visualization layout. Other limitation is that these services do not provide a differentiated set of inference algorithms, which limits the analysis of phylogenetic data.

GrapeTree

GrapeTree [21] facilitates the analyses of large numbers of allelic profiles by a static "GrapeTree Layout" algorithm that supports interactive visualizations of large trees within a web browser window. GrapeTree also implements a novel minimum spanning tree algorithm (MSTree V2) to reconstruct genetic relationships despite high levels of missing data. GrapeTree is a stand-alone package for investigating phylogenetic trees plus associated metadata and is also integrated into EnteroBase to facilitate cutting edge navigation of genomic relationships among bacterial pathogens.

3.3 PHYLOViZ Web Platform Solution

In order to fulfil all the objectives listed above, we have developed *PHYLOViZ Web Platform*.

PHYLOViZ Web Platform is a web platform that allows users to access and perform phylogenetic analyses and visualizations from anywhere with an internet connection, without requiring installation of software or access to high-performance computing resources. This is achievable by performing most computing operations and data management server-side.

It is intended as an alternative for PHYLOViZ, and substitute for PHYLOViZ Online, with many of the features present in PHYLOViZ, but improving on it by being a web version and scaling for larger datasets. At the moment, laboratories such as IMM are using PHYLOViZ, and this platform aims to provide an upgraded and more versatile alternative for their phylogenetic analysis requirements. With the PHYLOViZ Web Platform, laboratories will benefit from a range of enhancements and advancements in comparison to the current PHYLOViZ software.

Nowadays, datasets are becoming larger and larger, and their dimensions tend to increase even more in the future. This means that the tools used to analyze and visualize these datasets must be able to scale to these dimensions. Currently, publicly available datasets can reach dimensions of up to 500 000 profiles [22], with an anticipated growth in the number of profiles in the future.

The platform should additionally offer enhanced configuration and customization options for the phylogenetic analyses, and organization through the project and dataset management functionality. PHYLOViZ Online, the original web version, should therefore also become deprecated as it has less features and provides less modularity.

3.3.1 Functional Requirements

Functional requirements state what the system must do, and how it must behave or react to. The functional requirements that were identified for the PHYLOViZ Web Platform are as follows:

- Have a REST API that allows the execution of phylogenetic analysis tasks, such as creation and management of projects and datasets, phylogenetic inference and visualization algorithms.

- Have a user interface accessible through a web browser, that uses the REST API to allow users to perform phylogenetic analysis tasks.
- Allow users to create and manage multiple projects and phylogenetic datasets.
- Enable the user to upload files, namely typing data and isolate data to initialize the phylogenetic analyses.
- Allow the user to start computations, generating distance matrices and inference trees, and visualize the trees.
- Store all data and computation results, including visualizations and transformations for future use.
- Provide a variety of customization options to the user, including the choice of what algorithms and parameters are used for the computations, what layouts for the tree visualization and different transformations to the visualization;
- Allow multiple different distance matrices, trees and tree views to coexist within the same dataset, making different combinations possible, provided that customization of algorithms exists.
- Use a workflow management system to execute the phylogenetic analysis tasks and computations, looking to efficiently control execution of workflows.
- Support of authentication and authorization, to limit the project visibility to only its owner; in the future, it should be possible to share projects with other users, creating a collaborative environment.

3.3.2 Non-Functional Requirements

Non-functional requirements or quality attributes requirements are qualifications of the functional requirements or of the overall product. The non-functional requirements that were identified for the PHYLOViZ Web Platform are as follows:

- The platform code should be modular, in a way that it allows to easily reuse and extend it - add and remove workflows, tools - as well as integrate different databases and repositories for efficient storage and management of phylogenetic data.
- The platform should be stable and reliable, with minimal downtime and errors.

- The visualization of trees should be efficient and scalable, by using state of the art techniques for performance and/or controlling what parts of the tree are visualized at a time.
- The platform should be able to handle large datasets efficiently and scale to accommodate increasing data dimensions.
- The user interface should be simple and intuitive.
- The user interface should work in different browsers and should be responsive, being visually appealing in different screen sizes.
- The platform should be complemented with documentation that explains how to use it, the development process, and its deployment.

3.3.3 Use Cases

With the requirements listed above, we have identified the use cases that the platform shall support. A use case is a written description of how users will perform tasks on a system. It outlines, from the user perspective, the behaviour of the system as it responds to a request. The use cases identified shall allow to reason about who are the users of the platform, their objectives, the actions they are able to perform, and how the platform shall respond to each action.

The use cases are divided into three categories: main use cases, project use cases and dataset use cases. The main use cases are presented in Figure 3.1.

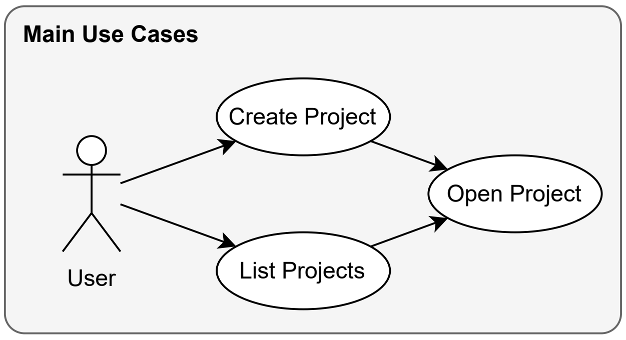


Figure 3.1: Main use cases.

The main use cases are the ones that are available when the user accesses the platform. The user can create a new project or list the projects he has access to. After creating a project or selecting a project, the user can open the project. The use cases available when a project is open are presented in Figure 3.2.

The project use cases are the ones that are available when a project is open. The user has access to the project information, including files and datasets.

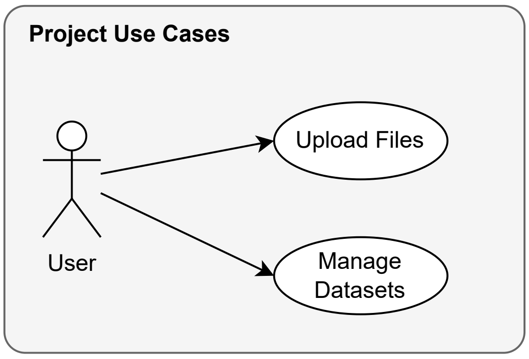


Figure 3.2: Project use cases.

A new dataset can be created or an existing dataset can be selected. Opening a dataset provides access to the previously saved state and associated information. This functionality allows users to revisit and analyze datasets that were previously used. By loading a dataset, the user can seamlessly continue their analysis from where they left off, ensuring continuity and preserving the progress made in their research.

The user can also upload files to the project, that can be used to create datasets.

After selecting a dataset, the user can perform the dataset use cases, presented in Figure 3.3.

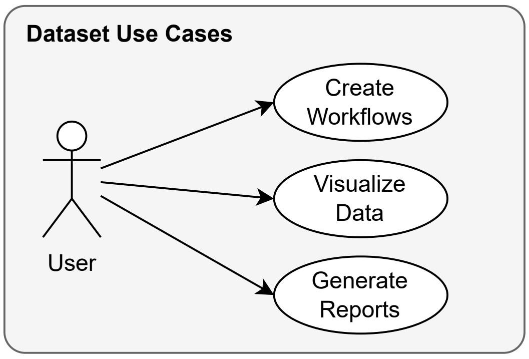


Figure 3.3: Dataset use cases.

The dataset use cases are the ones that are available when a dataset is selected. The user can perform phylogenetic analysis tasks, such as inferring a phylogenetic tree, using workflows. These workflows will be executed in the background and the user notified upon its termination.

The user can also visualize the results of these tasks and generate reports, which are documents that contain the analysis outputs, including the phylogenetic tree, statistical metrics, and other relevant visualizations. These reports serve as a concise summary of the analysis performed and provide a convenient way for the user to share and communicate their findings with collaborators or stakeholders.

Chapter 4

Architecture

This chapter provides an overview of the system's components and their interactions. It outlines the capabilities of the project and presents the architecture, entities, and implementation blueprint that have been designed and developed.

4.1 Overview

Figure 4.1 presents a diagram illustrating the main components of the system and their interactions. The system consists of a backend application (server-side) and a frontend application (client-side).

The backend comprises a collection of microservices [23], responsible for data processing and storage. To execute these tasks, it makes use of workflows, whose tasks run in containerized tools managed by a Workflow Manager.

The frontend is composed of a web application, that facilitates data visualization and user interaction. Communication between the frontend and the backend is achieved through a REST API, utilizing the HTTP protocol [24], which is provided by the Gateway.

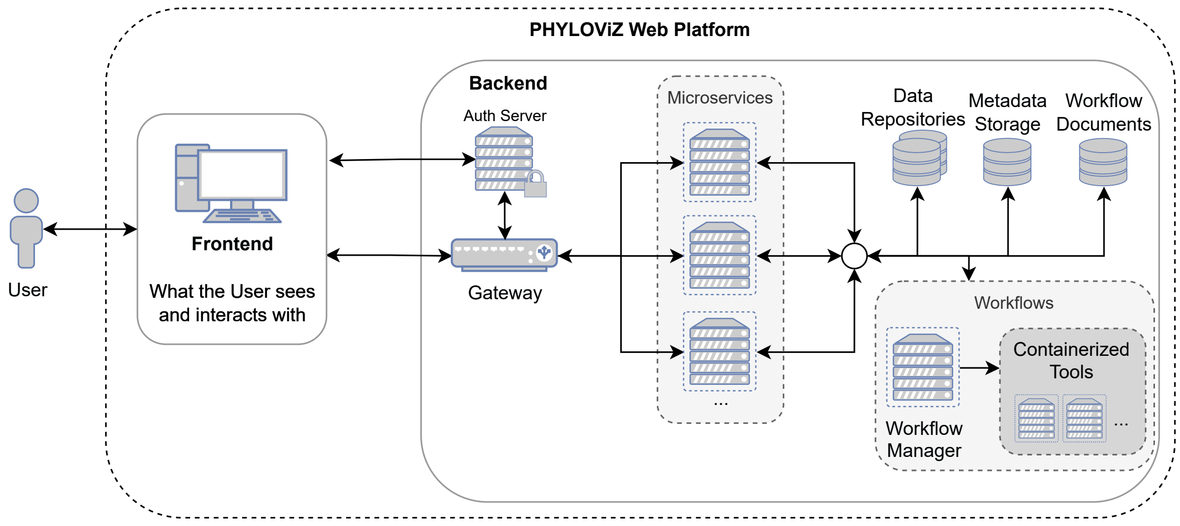


Figure 4.1: System architecture.

4.2 Data Model

The data in our system is categorized into two categories: metadata and phylogenetic data. This division can be observed in Figure 4.1, which shows *Metadata Storage* and *Data Repositories*.

In addition to the two categories mentioned, our system also has documents related to workflows, that are either used for their configuration - workflow templates and tool templates - or represent a specific instance of a workflow - workflow instances. This can also be observed in Figure 4.1, which shows *Workflow Documents*.

Metadata describes the application's resources, including their properties and relationships. Additionally, metadata serves as a reference to the data representations of these resources stored within the data repositories, which contain the actual content associated with each resource.

As an illustration, let's consider a tree resource that represents a calculated phylogenetic tree. This tree is stored exclusively as a file in the newick format within an S3 bucket, which serves as its data repository - a file repository or object storage. The tree not only contains the necessary data but also possesses associated metadata, enabling accurate tracking of its location. Its metadata comprises the following details:

- The name of the tree;
- The project and dataset to which it belongs;
- Its source, which may indicate the algorithm used for its generation;
- Information regarding the data repositories where the tree's data is stored. In this case, it specifies that the tree is stored within an S3 bucket, along with its

location (URL).

In the initial deployment, for the phylogenetic data, the data model from PhyloDB [25] was used, as it was the data repository of choice for the phylogenetic data. To store the files, an object storage like Amazon S3 can be used as the file repository. To better understand the job of the main data repositories, why they aren't necessarily restricted to be two, and why the data model of these repositories can be variable, refer to the section *Data Repositories* [4.2.2].

In this section, we will explain all the entities that make up our data model and how they are stored and connected to each other by the metadata system, as well as the workflow documents and how they are used to configure or keep track of workflows.

4.2.1 Metadata

The metadata database plays a crucial role in storing and managing the metadata of all resources within the system. For this purpose, a non-relational database, such as MongoDB, is commonly utilized.

The metadata associated with a resource encompasses a wide range of information. This can include essential details like the resource's name, description, creation date, and any other pertinent attributes that define and characterize it.

Moreover, the metadata also maintains information regarding the relationships between different entities within the system. These relationships can include associations with other resources, such as parent-child relationships or hierarchical connections, as well as any other relevant interconnections that exist between various entities [4.2.1].

For instance, consider the hierarchical connection between a project and a dataset as an illustrative example. Within the metadata, the dataset contains the identifier of the project, establishing a clear parent-child relationship. This hierarchical linkage ensures the systematic organization of resources, allowing for seamless navigation and traceability between related entities.

The metadata also serve as an abstraction on top of the data repositories to which the data is stored in: the user does not need to know what repositories/databases are internally at use and only need to refer to metadata to access the resource. This way, metadata internally provides to the application information about what data representations of the resource are available, and how to access said data. Refer to the example at the beginning of this section, 4.2.

By storing and managing such metadata, the system ensures a robust framework for organizing, querying, and retrieving information about the resources. This facilitates efficient search functionalities, enables data-driven analyses, and allows for comprehensive understanding and exploration of the relationships between different entities within the system.

In summary, the metadata database serves as a repository for both descriptive and relational information about system resources. By capturing essential resource attributes and their interconnections, it enhances the overall functionality and usability of the system, providing researchers and users with a rich source of information for their analyses and investigations.

Entities

The root entity of PHYLOViZ Web Platform is the project. A project is a collection of files and datasets.

The dataset refers to a phylogenetic study and as such contains the data relevant to that study. Each dataset within a project is separate from each other, and operates on its own set of resources and data - its important to mention that the typing data and isolate data files of each dataset are shared among all the datasets in the project. This allows for flexibility and customization of each dataset, but also gives the user full control over how they want to organize themselves. The responsibility of assuring that, in a project, the datasets come together as part of a bigger picture, lies on the user. However, this is not a strict requirement, and the user can choose to have datasets that are not coherent or related to each other. If the user has a clear vision and purpose for the project, they can align the datasets accordingly to achieve a coherent and consistent outcome.

The single shared base resources among all the datasets in the project are the uploaded files: typing data and isolate data. These are uploaded directly into the project itself, and can be reused by the different datasets. Each dataset will hold a reference to one single typing data file, and optionally, to a single isolate data file.

Figure 4.2 is a diagram that shows the relation between project's datasets and files.

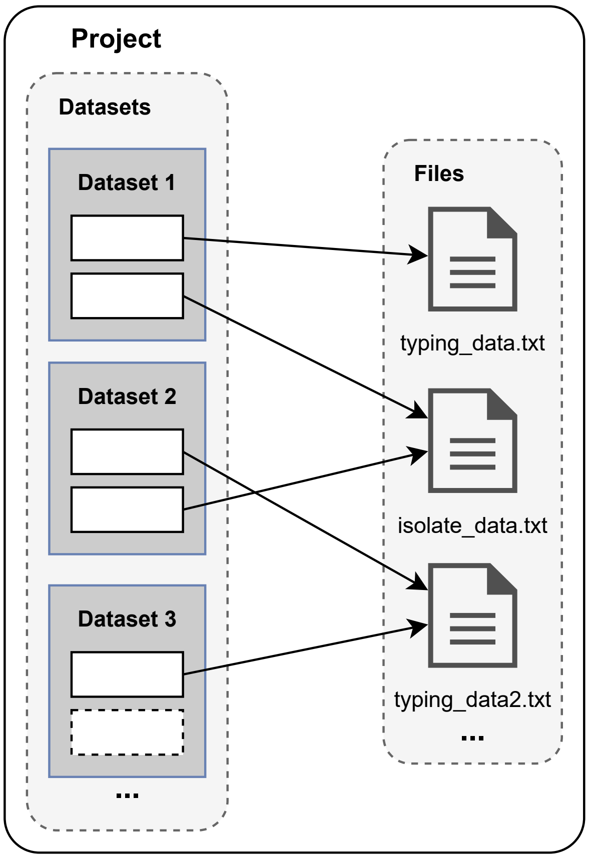


Figure 4.2: Relation between the project, datasets and files. The project has 3 datasets that share files. The dataset 1 and the dataset 2 share the same isolate data file, named "isolate_data.txt", while dataset 3 has no isolate data file. Dataset 1 is the only one with the file "typing_data.txt", while datasets 2 and 3 share the same typing data file "typing_data2.txt".

The other resources in a dataset, which are not shared between datasets of a project, are the result of algorithms/computations applied to the typing data. In our architecture, we discussed to what level the abstraction of the phylogenetic data should be. We decided on having the following entities (also called dataset resources throughout the report):

- **Distance Matrix** - result of applying a distance function to the typing data, generating the distances between each allelic profile - the basis for asserting which are closer in nature to each other. The generation of this resource might be skipped if the tree is generated with an inference algorithm that dynamically creates the distance matrix.

- **Tree** - result of applying an inference algorithm, based on distances, generating a phylogenetic tree with proper hierarchical relationships between the various profiles.

- **Tree Visualization/Tree View** - result of applying a layout to the phylogenetic tree in order to allow for its visualization. After the layout is imposed, transformations can be applied to alter the visualization to the user's needs.

Some resources are derived from other resources that are already available (e.g. a tree can be derived from a distance matrix). Therefore, these resources may have some relationships with each other, such as hierarchy or dependency [4.2.1].

A dataset can contain multiple of each of these resources. For example, a dataset can have two distance matrices, one calculated using the Jukes-Cantor model and another using the Kimura model, and then have four phylogenetic trees, two constructed using the neighbor-joining algorithm and two using the goeBURST algorithm, each with a different distance matrix as input. The dataset can also have different views of each tree, with a different layout and different transformations applied, such as showing only certain branches or highlighting certain taxa.

Figure 4.3 is a diagram that shows the relation between the project resources and all entities of our data model.

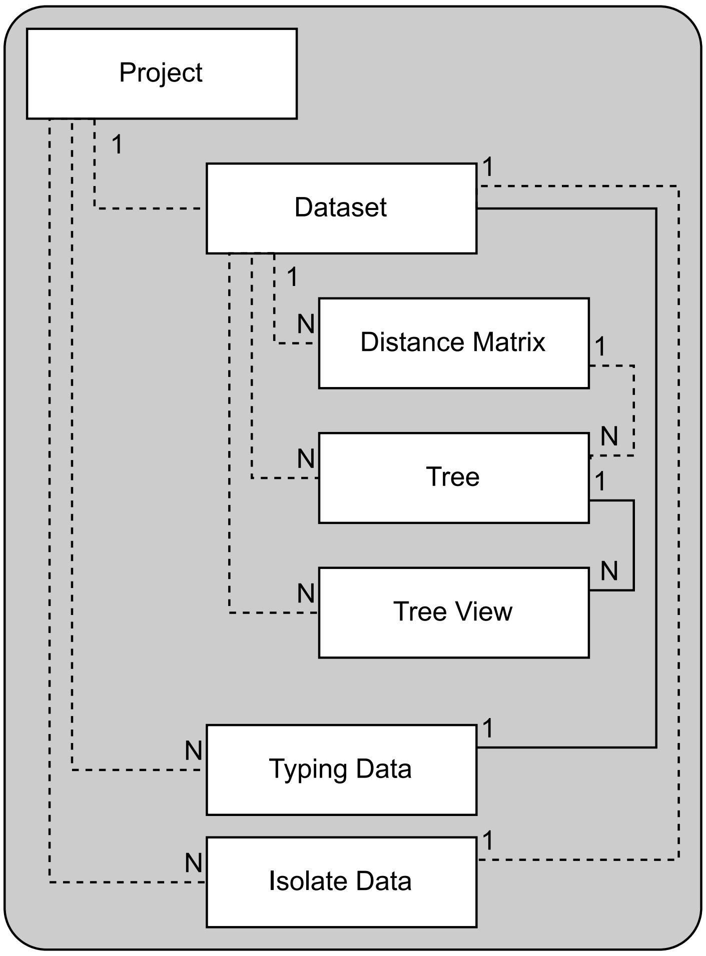


Figure 4.3: Entity overview diagram, showing the relationships between the entities of the platform. A project can have multiple datasets and multiple files. A dataset can have multiple distance matrices, trees and tree views; and is required to have a single associated typing data file and optionally one single associated isolate data file. Each of the dataset resources can be related to each other, such as a distance matrix being used to generate multiple trees, or multiple tree views being constructed from a tree.

Multiple data representations

In the case of files, distance matrices, trees and tree views, the metadata of each one not only contains general information about it, but also points to data representations of it in the data repositories [4.2.2]. Therefore, the metadata for these also contains:

- What repositories the data is on.
- Data specific to each repository, used to access the data stored on it.

Figure 4.4 shows how the metadata includes the information about the multiple data representations of a resource.

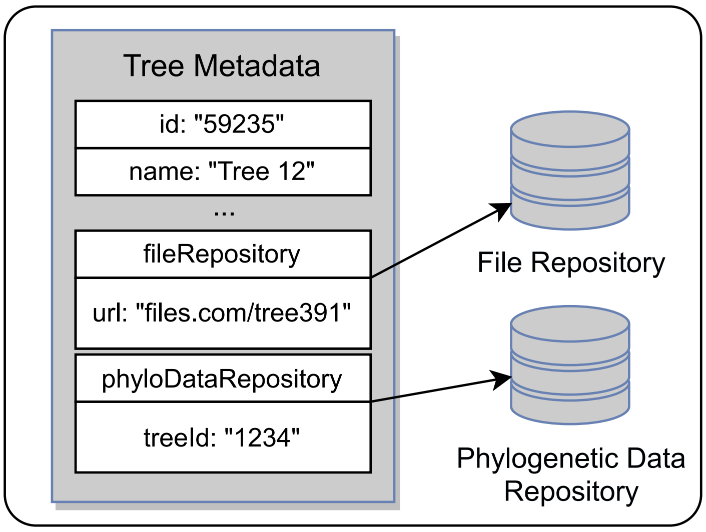


Figure 4.4: Metadata of a tree with the identifier ”59235”, containing two data representations: one in a file repository, with the URL where the data is; and one in a phylogenetic data repository, with a ”treeId” that identifies this data. Notice that the data specific to the repository doesn't need to be directly correspondent to the general information in the metadata. There may exist a tree with the identifier ”59235”, or name ”Tree 12”, for whom a data representation actually identifies it with the id ”1234” inside the repository.

Resource Origin and Source in Metadata

Within the metadata context, resources such as distance matrices, trees, and tree views possess an inherent origin or source that provides insights into their creation process. This origin, referred to as the ”source” within the metadata, consists of two components: the source type and the specific source details associated with that type.

To elaborate further, let's explore the different resource types and their corresponding sources:

- **Distance Matrix**:
  - Generated by a function: The source specifies the function utilized for generating the distance matrix, such as the Hamming distance function.
- **Trees**:
  - Generated by an algorithm using an existing distance matrix: The source includes information about the algorithm employed including the chosen parameters and the specific distance matrix utilized in the generation process;
  - Generated by an algorithm using typing data directly (distance matrix dynamically generated): The source indicates the algorithm utilized for tree generation directly from typing data and the chosen parameters;

  - From a file: The source encompasses information about the file, including its type/format (e.g., newick, nexus), file name, and additional relevant details based on the specific file type.

- **Tree View**:
  - Contains the tree used: The source indicates the tree incorporated within the tree view.

In the metadata, the source details of each resource are exposed, providing valuable insights into how the resource was obtained or derived. This information facilitates traceability, allowing researchers to understand and contextualize the origins of the resources within their analyses.

Figure 4.5 illustrates how the resources can serve as sources of other resources.

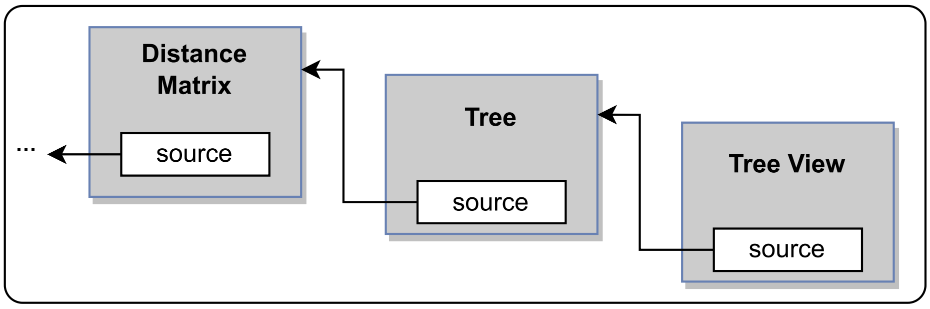
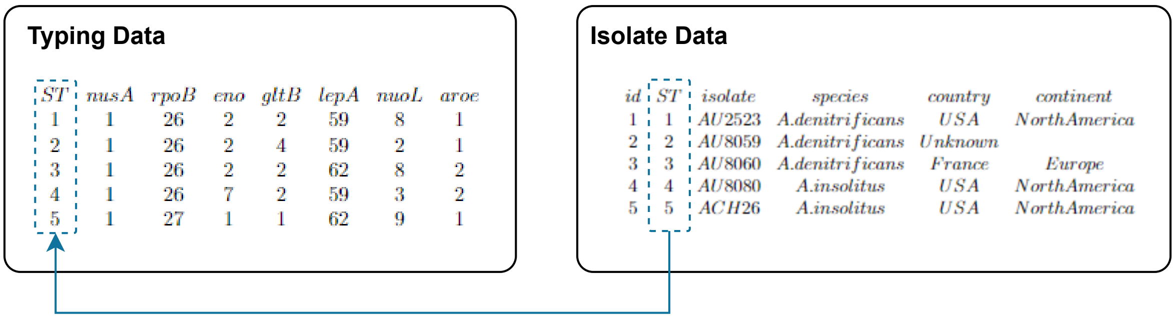


Figure 4.5: Example of source of resources. For example, a distance matrix can be the source of a tree, and subsequently, that tree can become the source of a tree view.

Isolate data key

The metadata associated with isolate data includes a list of keys, from which one can be chosen during dataset creation. This chosen key serves as the primary means of linking isolate data with typing data. Subsequently, the dataset retains essential information about this key, ensuring the connection between the isolate data and its corresponding typing data.

Figure 4.6 provides an illustrative example of the relationship between isolate data and typing data, highlighting the utilization of the ”Sequence Type” (ST) column as the key for establishing this connection.


Typing Data

ST	nusA	rpoB	eno	gltB	lepA	nuoL	aroe
1	1	26	2	2	59	8	1
2	1	26	2	4	59	2	1
3	1	26	2	2	62	8	2
4	1	26	7	2	59	3	2
5	1	27	1	1	62	9	1

Isolate Data

id	ST	isolate	species	country	continent
1	1	AU2523	A.denitrificans	USA	NorthAmerica
2	2	AU8059	A.denitrificans	Unknown	
3	3	AU8060	A.denitrificans	France	Europe
4	4	AU8080	A.insolitus	USA	NorthAmerica
5	5	ACH26	A.insolitus	USA	NorthAmerica

Figure 4.6: Relationship between isolate data and typing data, using ST column as the key for establishing the connection. Its important to highlight that a profile can have zero or multiple isolates associated with it.

Tree View Transformations

The transformations for the tree view should be stored in the Tree View metadata, so the user can recover the visualization if they close it, in order to retrieve the same view with the same transformations the next time the user accesses it.

4.2.2 Data Repositories

To store the phylogenetic data itself, the system resorts to a variable set of repositories/databases, instead of just one, and as such isn't restricted to a single data model for these data. This allows us to change what repositories are used, and with the help of the metadata, even allow for multiple different repositories to be used all at once [4.2.1]. The repositories for phylogenetic data are responsible for storing the resources generated by the computations and the files uploaded by the user.

Generally, these repositories should compose of at least two:

- **File Repository** - this is responsible for storing files, such as typing data, and file representations of resources, such as a distance matrix, or a tree in newick format. These files are usually large, and therefore this needs to be scalable solution, such as Amazon S3 (Simple Storage Service) [26];

- **Phylogenetic Data Repository** - this is also responsible for storing the phylogenetic data, just like the file repository does. However, this should focus more on the relations between the resources, and allow for efficient querying. Therefore, should use an actual database that follows a data model for phylogenetic data, such as PhyloDB [7].

By using the appropriate repositories the system can ensure that the phylogenetic data is stored and retrieved efficiently.

To enable multiple representations of a resource across different repositories, data replication becomes necessary. This process ensures that the data is duplicated, although stored in different formats, and accessible in each designated repository.

In order to make it possible for these repositories whose libraries or API follow different interfaces to work seamlessly in our platform, we resort to adapters, which transform, or adapt, the code to be compatible with a common interface. This is known as the *Adapter design pattern* [27, 28]. Initially, we even called the data repositories "adapters", because of this emphasis on having to adapt already existing repositories, but ended up labeling them as "data repositories", as the business logic code does not need to know if the data access code is adapted, and only cares that it follows a common interface.

To better represent the advantage of allowing more data repositories than the aforementioned two, consider a scenario where efficient storage and retrieval of distance matrices is desired. The chosen general phylogenetic data repository may not include an optimal solution or sufficient efficiency for this specific purpose. In such cases, an alternative approach can be adopted by incorporating a repository specifically optimized for distance matrices alongside the existing repositories, without causing any interference. This allows the option to selectively utilize the specialized repository for the desired operations, providing a more effective solution.

4.2.3 Workflows

Our platform incorporates a database for storing documents relevant to workflow configuration and tracking. To facilitate this, we utilize a non-relational database like MongoDB, which offers the advantage of configuring workflows using a language such as JSON.

For workflow configuration, we have workflow templates and tool templates. Additionally, for information about a specific workflow instance, we have workflow instances, which can also be referred to as workflow instance metadata or simply workflow metadata.

Workflow Templates

Workflow templates serve as blueprints for the actual workflows to be executed, representing the existence of a workflow type itself: adding a workflow template means adding a new workflow type to be used by the application. Each template includes the following details:

- Workflow type: This serves as the identifier for the template, categorizing the workflows built from it.

- Name and description: These provide a clear understanding of the workflow's purpose.

- Schema of input arguments: Workflows have specific input arguments that define the data they will work with. The template specifies the schema for input argument validation, as these arguments are passed on to the tasks.

- Tasks: Each workflow consists of tasks, each associated with a corresponding tool where a specific command will be executed. The template includes the commands to be executed in each task, with placeholders that are later substituted with the appropriate input arguments during workflow creation. See 4.5.3 for more information on placeholder substitution during workflow creation.

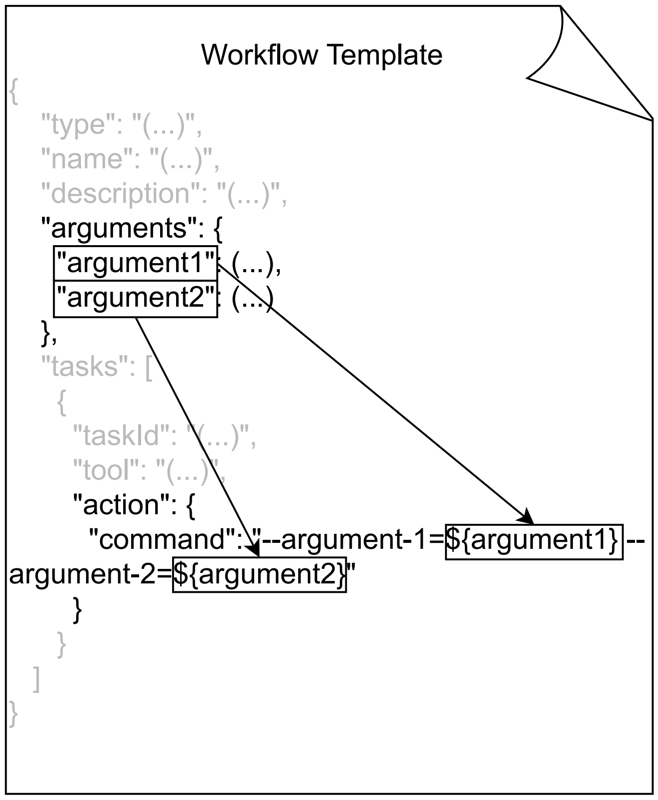
Each argument in the schema has a type, which can be one of the following:

- `string`: The argument is a string and may have an "allowed-values" field that limits the acceptable values. This prevents code injections since these arguments are directly substituted into the placeholders of the tool's command arguments;
- `uuid`: The argument is a universally unique identifier (UUID);
- `objectid`: The argument is an object identifier (ObjectId);
- `number`: The argument is a numerical value;
- `datasetId:` The argument represents the ID of a dataset;
- `distanceMatrixId`: The argument represents the ID of a distance matrix;
- `treeId`: The argument represents the ID of a tree;
- `treeViewId`: The argument represents the ID of a tree view.

Each argument may also specify if it is required, or an accompanying prefix to be applied during placeholder substitution. By default, every argument is required and has no prefix.

The `projectId` and `workflowId` arguments are reserved and serve as identifiers for workflows within the system. Every workflow is associated with a unique project ID and workflow ID. If these arguments are present, they will automatically replace the corresponding placeholders in the task during workflow execution.

The relation between placeholders of tasks and the input arguments on the schema is shown in Figure 4.7.



```
{
  "type": "(...)",
  "name": "(...)",
  "description": "(...)",
  "arguments": {
    "argument1": (...),
    "argument2": (...)
  },
  "tasks": [
    {
      "taskId": "(...)",
      "tool": "(...)",
      "action": {
        "command": "--argument-1=${argument1} --argument-2=${argument2}"
      }
    }
  ]
}
```

Figure 4.7: Relation between placeholders and arguments of a workflow template. Arguments "argument1" and "argument2" specified in the schema of the workflow template, correspond respectively to the placeholders ${argument1} and ${argument2} in the command of the task.

In summary, a workflow template document encompasses all the necessary details for a specific workflow type.

Tool Templates

Similar to how workflow templates specify a workflow, tool templates define a tool. They provide the following information:

- Name and description: The name serves as the identifier for the tool and is referenced by the tasks. The description provides additional details about the tool's functionality.
- Access instructions: This specifies how to access the tool, whether through an API or a library. It includes instructions for the tool's Docker image, the Docker volume used for storing output data and other tools configurations such as network. The Docker volume path contains placeholders that are substituted with the project ID and workflow ID, ensuring that each workflow has a dedicated directory for task outputs. See 4.5.3 for more information on placeholder substitution during workflow creation.

Workflows Instances

Workflow instances are generated during workflow creation based on workflow templates. They represent the actual execution or non-execution state of the workflows. Although not directly passed to the workflow manager, this document provides valuable information about the workflow, allowing for better integration with the application rather than being solely controlled by the workflow manager. Each workflow instance document includes the following details:

- Workflow ID: This unique identifier distinguishes each specific workflow.
- Workflow type: Corresponding to the type of the workflow template it was built upon.
- Project ID: Every workflow belongs to a project.
- Workflow object: This object is passed to the workflow manager and contains the tasks to be executed, each with its associated tool and action (command) to be executed within the tool.
- Workflow status: It can be one of the following: RUNNING, SUCCESS, or FAILED.
- Failure reason: If the workflow failed, the reason why it failed.
- Task Logs: list of logs of the tasks, in string format.
- Workflow progress: Represented as a percentage value from 0 to 100. The progress is managed directly by the tasks and serves to inform the user about the workflow's progress.
- Additional data from tasks: This includes any additional information or results generated by the tasks during the computation.

Tools

The tools used in the tasks of the workflows can be:

- Totally external to our application, like tools whose job is only to execute algorithms with the given command arguments. They have no knowledge of the application's resources other than those which are present in the directory of the docker container. An example of a tool like this is phylolib [17], which is simply a library for phylogenetic analysis algorithms, built well before PHYLOViZ Web Platform.

- Internal to our application. These can have knowledge of all the resources of the application, and are built specifically to work in our infrastructure. They can access the data repositories, the metadata and even the workflow instance, and can also serve as "auxiliary" tools for external tools.

It is very noteworthy that these internal auxiliary tools are needed. The job of these is, not to run algorithms, but handle data and metadata before and after the tasks that execute algorithms. There should exist auxiliary tools for:

- Retrieving the data from data repositories. A tool like this should therefore access the resource's metadata to know the location of that resource in the data repository. For example, retrieving/downloading a typing data file from the S3 file repository into the docker container's volume, for another tool to run an algorithm using it.

- Uploading the data into the data repositories and write the associated metadata for the newly created resource. This metadata might need to include the origin of the resource, such as what algorithm was used and the parameters [4.2.1]. For example, a previous tool ran an algorithm on a typing data file and generated an inference tree as a file in the newick format; this tree file needs to be uploaded to S3, and a new metadata for it needs to be created, pointing to this S3 data representation.

The internal tools are not only auxiliary, as they can also run algorithms and execute any other operations, the main point being they have total access to the application's resources, like metadata, the data repositories and the workflow instances. For instance, imagine there isn't an external tool that calculates the layout coordinates of a tree view, and that calculation is done directly in the phylogenetic data repository: an internal tool could communicate with the phylogenetic data repository and compute the tree view with a specific layout, also creating the metadata for that tree view resource.

4.3 Frontend Application

The frontend application is composed of a web application, which is responsible for the visualization of the data and the interaction with the user. This application provides a simple and intuitive interface for the user to interact with the system, allowing it to manage projects and datasets, upload files, execute computation tasks, and visualize the results, among other functionalities.

This application is divided into multiple pages and components, and has a service layer that is responsible for communicating with the backend application through the REST API provided by the API Gateway.

4.4 Gateway

The API Gateway is responsible for routing requests to the microservices and providing cross cutting concerns to them such as: security, monitoring/metrics, and resiliency.

In the context of authentication, the Gateway is used as an OpenID client, that will authenticate users using access management server capabilities. Once the user is authenticated, the Gateway will use the Access Token received from access management server to forward requests to the appropriate microservice.

4.5 Microservices

The backend application is composed of a set of microservices, which are responsible for the execution of the algorithms and the storage of the data. Each microservice is responsible for a specific task, and is independent of the others, which means that they can be replaced by other microservices that provide the same functionalities.

The microservices communicate with data storages described in the data model: a set of data repositories for phylogenetic data storage and a metadata database; and in the case of the Compute microservice, a database for workflow documents.

The system is composed of the following microservices:

- **Administration microservice**: responsible for the projects and datasets management, including the creation, retrieval, deletion, and edition of projects and datasets;
- **FileTransfer microservice**: responsible for the upload and download of project files;
- **Compute microservice**: responsible for the execution of the phylogenetic analysis workflows, including the execution of the phylogenetic analysis tools, the distance matrices calculation, and the phylogenetic tree construction;
- **Visualization microservice**: responsible for the visualization of all the results of the phylogenetic analysis workflows, including the visualization of typing data and isolate data files, distance matrices and phylogenetic trees. In addition, when the user requests the visualization of the tree, the visualization microservice ensures that any previously applied transformations to the tree visualization are included in the generated output. This allows for a seamless and consistent viewing experience, as the user can readily access the tree visualization with all the desired transformations intact.

In the following subsections we take a deeper look at each microservice architecture.

4.5.1 Administration Microservice

The Administration microservice is responsible for managing the projects, datasets, files and dataset resources.

Figure 4.8 is a diagram that shows the architecture of the Administration microservice.

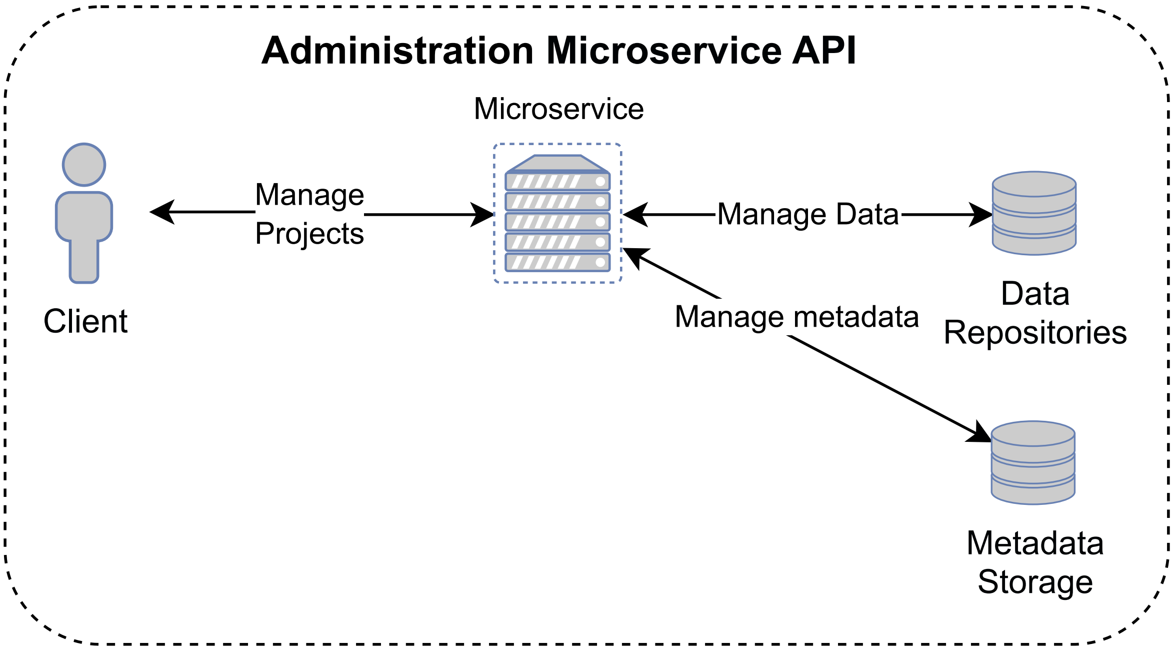


Figure 4.8: Administration microservice architecture.

As seen in the diagram, the Administration microservice communicates with data repositories and the metadata storage.

The following operations are supported:

- Create, retrieve, edit and delete a project;
- Create, retrieve, edit and delete a dataset;
- Edit and delete resources: distance matrix, tree, tree view, typing data, isolate data.

It is noteworthy that the creation operations for data within projects and datasets are not available in the Administration Service. Instead, files such as typing and isolate data are uploaded to FileTransfer, and dataset resources such as distance matrix, tree, and tree view are added by the workflows started by Compute.

When retrieving a project, not only is the project's information transmitted, but also the complete project structure, including all files, datasets, and resources within it. This also applies to retrieving a dataset, where all resources contained within the dataset are sent. The transmitted information encompasses various metadata, such as ID, name, description, relationships between resources, origin, and parameters used in calculations - all stored in the metadata documents [4.2.1]. Overall, this comprehensive information effectively prepares the user or front-end to proceed with further tasks, interactions, and the invocation of additional operations within the project.

Deleting a project results in the cascading deletion of everything inside of it: files, datasets, and resources inside datasets. The same can be said for deleting a dataset, all the resources inside of it are deleted. In the case of deleting a file of a project, if any dataset is using it, the deletion should be prevented. Deletion of a dataset resource is prevented if it is a dependency of another resource, meaning it is marked as a source of another resource [4.2.1].

Deleting a resource results not only in the deletion of its metadata, but also its data from the data repositories the metadata is pointing to.

When it comes to editing the information of a resource, such as the name and description of a project, the modifications exclusively impact the metadata stored within the system.

4.5.2 FileTransfer Microservice

The FileTransfer microservice is responsible for the upload and download of files of the projects.

Initially, this functionality was implemented in the Administration microservice, but it was decided to separate it into a different microservice, because the Administration microservice was very extensive and upload and download of files are not directly related to the management of projects and datasets.

In the future, if needed, this microservice can be divided into two other microservices, one responsible for the upload of files and another responsible for the download of files. This was not done because it would not be worth it, since this microservice already has a simple and intuitive architecture.

Figure 4.9 is a diagram that shows the architecture of the FileTransfer microservice.

As seen in the diagram below, the FileTransfer microservice communicates with data repositories and the metadata storage. It communicates with a data repository to upload and download files. On the other hand, it communicates with the metadata storage to create and retrieve the metadata of the files. If a file is uploaded, the metadata of the file is stored in the metadata storage, and if a file is downloaded, the metadata of the file needs to be retrieved from the metadata storage before the file is downloaded, in order to the microservice know the location of the file in the repository.

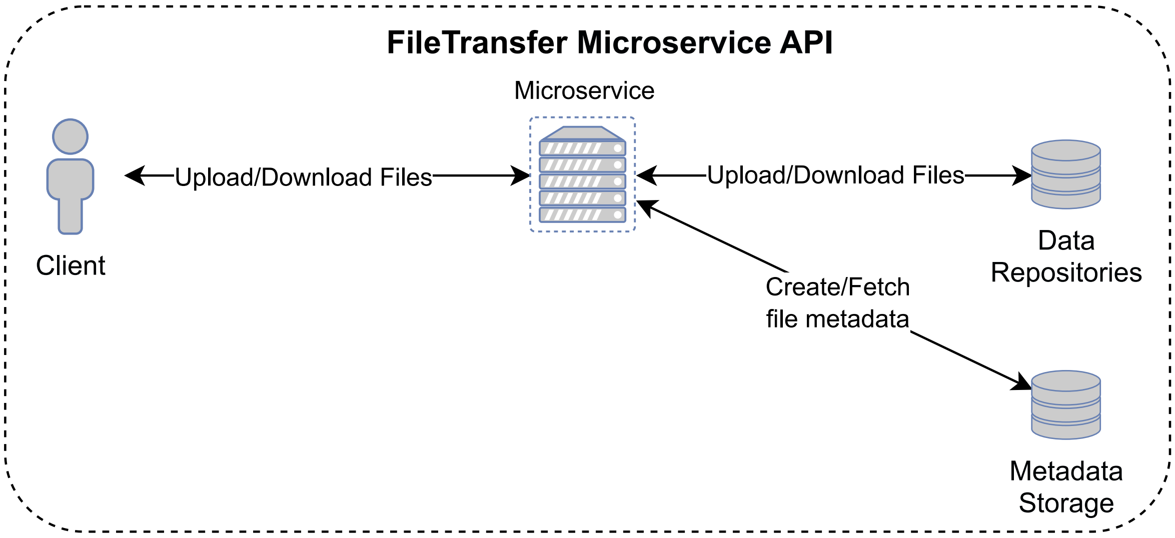


Figure 4.9: FileTransfer microservice architecture.

4.5.3 Compute Microservice

The Compute microservice is responsible for the execution of workflows for the phylogenetic analysis, which may include execution of the alignment of sequences, the distance matrices calculation, the phylogenetic tree inference methods and computing of the tree layout.

Figure 4.10 is a diagram that shows the architecture of the Compute microservice.

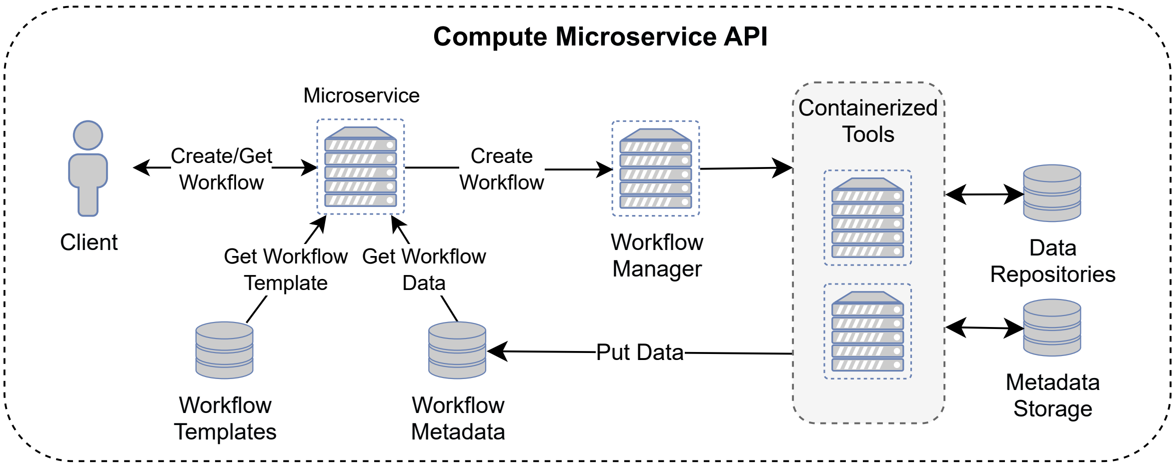


Figure 4.10: Compute microservice architecture.

This is the most complex microservice of the system, because it needs to communicate with multiple tools, which are responsible for the execution of the phylogenetic analysis workflows.

Workflows

A workflow is a sequential procedure consisting of multiple tasks or steps that are often interdependent, where the output of one task serves as the input for subsequent tasks. This pattern continues until the final task generates an output, marking the completion of the procedure. The structure and flow of a workflow resemble a pipeline or a flowchart, which is why these procedures are commonly referred to as pipelines or *workflows* in this document.

For details about the specific documents and data model employed by our application for workflows, please refer to the *Workflows* section in the Data Model [4.2.3].

Workflow creation

The creation of a workflow involves specifying the workflow type and providing the necessary input arguments. The compute microservice follows a series of steps to ensure the validity of the workflow creation process.

Firstly, the microservice verifies the existence of the specified workflow type. Once confirmed, it retrieves the corresponding workflow template. The template plays a crucial role in the validation process by defining the expected schema for the input arguments.

During validation, the input arguments are checked against the schema defined in the template. For instance, if a parameter has a type of 'string', it may be required to have a predefined set of acceptable values. This validation step serves as a crucial security measure to prevent code injections, particularly if the arguments are directly incorporated into the command arguments of the executed tools.

Figure 4.11 illustrates an example of how this argument matching process occurs for the purpose of validation.

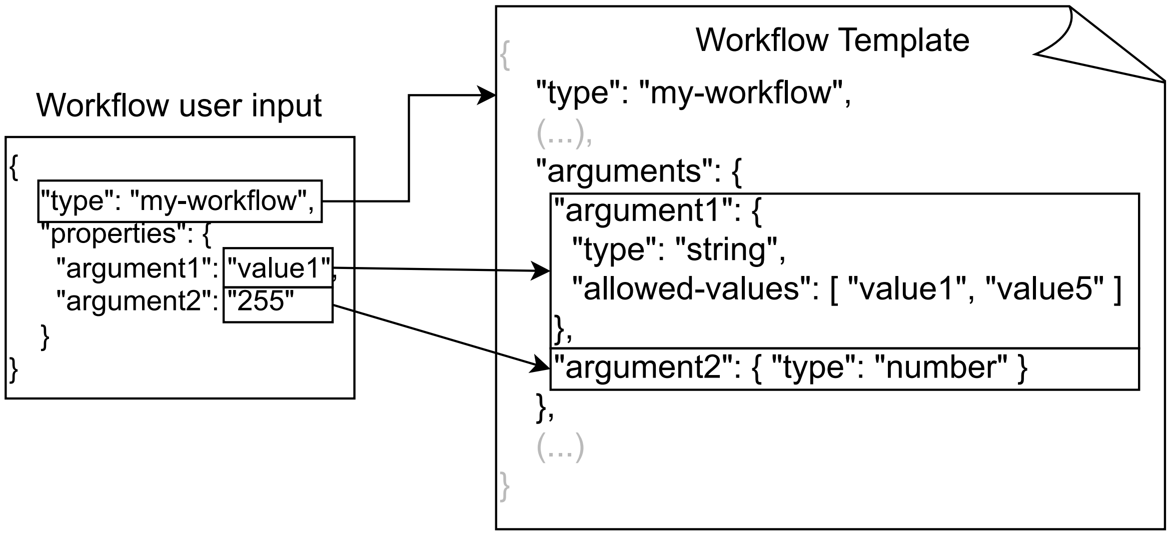


Figure 4.11: Argument validation during workflow creation. The workflow creation request by the user contains the type of the workflow, "my-workflow", and the properties/arguments for it, "argument1" with the value "value1", and "argument2" with the value "255". The workflow template of type "my-workflow" is received and the arguments are matched with the schema, in which "argument1" is of type string and has "value1" and "value5" as the allowed values; and "argument2" is of type number.

Once the input arguments have been successfully validated, the compute microservice proceeds to populate the placeholders within the workflow template using the provided arguments. This placeholder substitution process is illustrated in Figure 4.12, where each argument is matched with one or more placeholders within the tasks of the workflow template. As a result, a new workflow is generated, ready for execution.

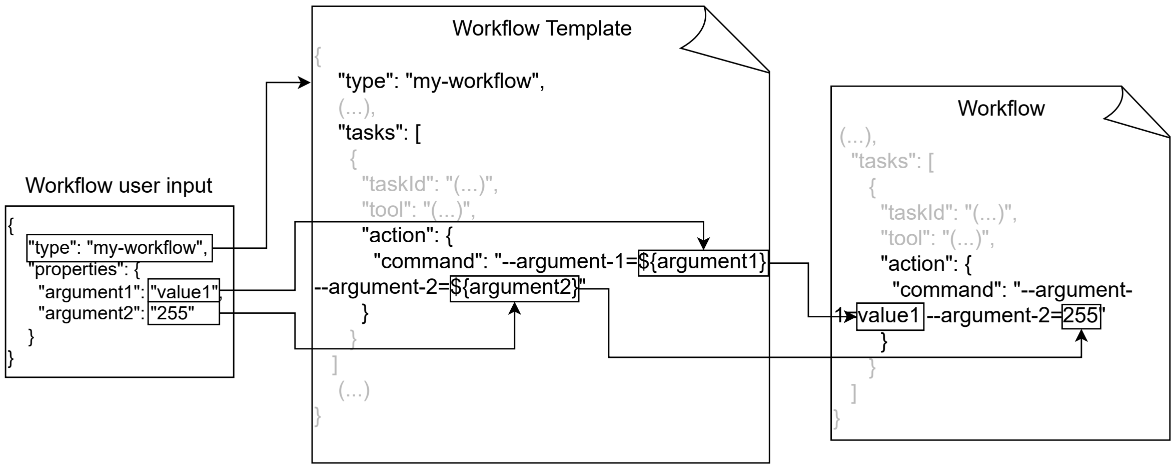


Figure 4.12: Workflow template placeholder substitution. The workflow creation request by the user contains the type of the workflow, "my-workflow", and the properties/arguments for it, "argument1" with the value "value1", and "argument2" with the value "255". The workflow template of type "my-workflow" is received and the placeholders in the tasks are substituted by the arguments, creating a workflow object with the values "value1" and "255" appropriately set in the command lines of the tasks.

Afterwards, the tool templates of the tools associated with the tasks of the workflow are also retrieved, and their placeholders are filled with the ID of the project and the ID of the workflow. After all placeholders in the workflow and tools are substituted, these newly created objects are passed onto the manager.

The manager then executes this workflow, initializing the tasks in containerized environments. The tools of the tasks will communicate with both the metadata storage and the data repositories with full freedom, to create new resources, storing them in the data repositories, and create the associated metadata for these. The tasks may also incorporate additional data into the workflow instance metadata. For instance, in a workflow focused on computing a distance matrix, the workflow can include the identifier of the generated distance matrix within the workflow instance metadata. For more information on workflow tools and what they can do, check section 4.2.3.

Getting workflow status and data

To retrieve the status and associated data of a workflow, the microservice examines the metadata of the workflow instance. This metadata contains additional information and serves as a reference point for accessing the workflow's current status. However, it's important to note that the workflow instance is simply a document within our application [4.2.3].

To update the workflow instance, communication with the workflow manager is necessary. This interaction ensures that the document remains up-to-date and reflects the current state of the workflow. In the event of a failed workflow, the reason for the failure is specified. The task logs are always present, and as such may provide information regarding the cause of the failure.

Workflow Management System and Tools

In order to manage the workflows, the microservice uses a Workflow Management System (WMS), which is responsible for the execution of the workflows and the communication with the tools. This WMS is composed of a workflow engine, which is responsible for the execution of the workflows, and a workflow scheduler, which is responsible for the scheduling of the workflows.

The tools are containerized, which means that they are isolated from the host operating system, and they are executed in a container, which is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.

Configuration

With the existence of documents such as workflow templates [4.2.3], workflows should be easily configurable by the administrator, enabling the definition of new tools and workflows. Consequently, the API for the Compute Microservice should focus on validating initial parameters and launching workflows, while parameter validation and other intrinsic behaviors should be handled within the workflow itself.

For information on how to configure the existing workflows and create new ones, refer to *Customize Workflows* section [6.6.1].

4.5.4 Visualization Microservice

The Visualization microservice is responsible for the visualization of all the results of the phylogenetic analysis workflows, including the visualization of the distance matrices, and of the phylogenetic tree. Figure 4.13 is a diagram that shows the architecture of the Visualization microservice.

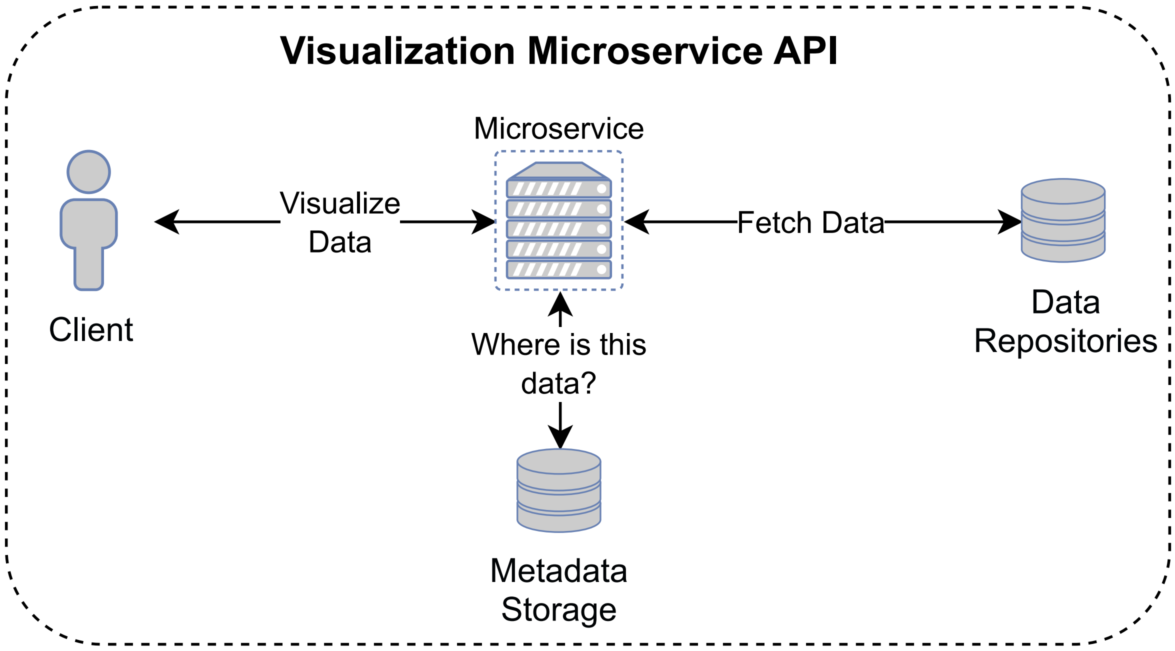


Figure 4.13: Visualization microservice architecture.

As seen in the diagram above, this microservice communicates with the metadata storage to retrieve the location of the data, and communicates with the data repositories to retrieve it.

In the case of tree views, the transformations associated with the view are also retrieved from the Tree View metadata, in order to recreate the view just like the last time it was accessed. Therefore, this microservice should also have endpoints to save a tree view's transformations and coordinates generated in the frontend.

4.6 Access Management

In order to access the system, the user must be authenticated. The authentication is done using the OAuth 2.0 protocol [29], which is an authorization framework that enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf. The authentication is done using the authorization code grant type, which is the most secure and recommended flow for web applications.

As seen in Figure 4.1, an auth server - also mentioned as access management server - is responsible for authenticating the user and issuing access tokens to the client application. The client application is responsible for sending the access token to the API Gateway, which will use it to forward requests to the appropriate microservice.

4.7 Conclusion

We believe we have built an architecture that allows for phylogenetic analyses, seeking to solve many of the problems associated with this kind of system. It provides the user flexibility, customization, and control over phylogenetic data, through project and dataset creation and management. Namely, how the resources are organized within the datasets, like allowing multiple resources of the same type to coexist in the same dataset, resulting in the possibility of having multiple combinations of resources generated through different methods and algorithms, and in the visualization, choice of layouts and transformations to suit the user's needs.

The use of a workflow manager for executing resource-intensive computations in an external controlled environment alleviates the burden of computation from the user. This ensures that users are not overwhelmed by the computational load and enables them to concurrently run multiple workflows. Additionally, a workflow manager offers several other advantageous features, such as streamlined task management, efficient resource allocation, and comprehensive monitoring capabilities.

We also believe we have achieved a highly modular architecture in terms of what tools, algorithms, and databases for phylogenetic data are used. The metadata management system makes it easy to define new repositories for the phylogenetic data, which may prove useful along the years as new, more robust, performance optimized data storage and data handling solutions may appear. The workflow configuration system allows for new workflows to be created with ease, integrating both these new data repositories and new algorithmic tools. Our system is therefore extensible and adaptable.

Chapter 5

Software Stack

In this chapter we introduce the most relevant technologies that support the different components of the PHYLOViZ Web Platform, explaining what they are and why they are being used in the context of our project.

We will start by introducing the technologies that are used in the backend, then we will introduce the technologies that are used in the frontend, and finally we will present the tools that enable version control, continuous integration, and other relevant practices in the software development process.

We obtained experience in some of these throughout the several courses of our bachelor's degree in computer science, which highly influenced us in choosing them in our project:

- Web Application Development (Desenvolvimento de Aplicações Web) [30] - Spring, Docker, React, Material-UI, Webpack, GitHub Actions;
- Systems Virtualization Techniques (Técnicas de Virtualização de Sistemas) [31] - Docker;
- Software Laboratory (Laboratório de Software) [32] - Spring, Docker, GitHub Actions;
- Software Development Techniques (Técnicas de Desenvolvimento de Software) [33] - MongoDB;

Git and GitHub were used during all of them.

5.1 Backend Technologies

The backend technologies used in the PHYLOViZ Web Platform are essential for powering the underlying logic and functionality of the system. These technologies encompass the server-side components that handle data processing, database management, and

API integrations. In this section, we will explore into the key backend technologies employed in our project, providing a comprehensive overview of their functionalities, benefits, and relevance to our system. We will explore how these technologies work together to support the seamless operation and performance of the PHYLOViZ Web Platform, enabling efficient data processing, storage, and retrieval to deliver a robust and scalable web application.

The main programming language used in the backend is Java [34], which is a general-purpose, object-oriented programming language. We choose Java because it is a widely documented and supported language in the area of backend development. An alternative language that we considered was Kotlin, which is a cross-platform, statically typed, general-purpose programming language with type inference. Kotlin [35] is similar to Java and built as a better alternative for it, compiling to Java and therefore having full support for all its libraries. We opted for Java instead of Kotlin for this project, because Java is more prevalent in the job market and we wanted to gain experience in programming with it for a large-scale project, which would prepare us for various development environments.

Python was also used in the backend to develop some basic starter compute tools. We choose Python because it is simple and makes it easy to develop these tools.

Spring

Spring [36] is a framework for developing Java applications. It provides a set of tools to facilitate the development of applications, such as dependency injection, transaction management, and web services. Spring is a modular framework, which means that it can be used in different ways, depending on the needs of the application.

For example, Spring Boot [37] is a module that provides a set of tools to facilitate the creation of stand-alone, production-grade Spring-based applications. Spring Boot is used in this project to create the web application of each microservice.

Spring Cloud Gateway [38] is a module that provides a simple, yet effective way to route to APIs and provide cross cutting concerns to them such as: security, monitoring/metrics, and resiliency. Spring Gateway is used in this project to create the API Gateway, which is responsible for routing requests to the microservices.

We choose Spring as the framework for the web application because it is the most popular framework for developing Java applications. It is also a well documented and supported framework in the area, and we already have experience with it.

Keycloak

Keycloak [39] is an open-source identity and access management solution. It provides a set of tools to manage users, authentication, authorization, and other security-related features.

In the context of this application, Keycloak acts as an OpenID [40] identity provider (also mentioned in the report as access management server and auth server) that authenticates users and issues access tokens that can be used to access protected resources. It supports various protocols such as OAuth2, OpenID Connect, and SAML for authentication and authorization.

We wanted an existing identity and access management solution because this means we would not have to implement these features from the ground-up, reducing the development time from our application and minimizing the potential sources of errors. We use Keycloak as the auth server because it is an open-source solution, which means that we can freely use it in our project. Nonetheless, it implements various security, authentication and authorization techniques, only requiring configurations in order to be used. Paid alternatives include Auth0, Okta and Microsoft Azure Active Directory, and the free versions of these, if available, may turn out to be too limited.

Docker

Docker [41] is an open-source project which wraps and extends Linux containers technology to create a complete solution for the creation and distribution of containers. The Docker platform provides a vast number of commands to conveniently manipulate containers.

A container is an isolated, yet interactive, environment configured with all the dependencies necessary to execute an application. The use of containers brings advantages such as:

- Having little to no overhead compared to running an application natively, as it interacts directly with the host OS kernel and no layer exists between the application running and the OS;
- Providing high portability since the application runs in the environment provided by the container; bugs related to runtime environment configurations will almost certainly not occur;
- Running dozens of containers at the same time, thanks to their lightweight nature;
- Executing an application by downloading the container and running it, avoiding going through possible complex installations and setup.

To easily configure the virtual environment that the container hosts, Docker provides Docker images. Images are snapshots of all the necessary tools and files to execute an application. Containers can be started from images, the same way virtual machines run snapshots. To effortlessly distribute images, Docker provides registries. These are public or private stores where users may upload or download images. Docker provides a cloud-based registry service called DockerHub [41].

In addition to the Docker platform, we use Docker Compose to orchestrate the containers. Docker Compose is a tool for defining and running multi-container Docker applications. With Compose, we can define a multi-container application in a single file, then spin it up in a single command which does everything that needs to be done to get it running. Compose is especially useful in development environments, testing environments, and CI workflows.

We use Docker as an alternative for software containerization because it is the most well known and actively developed and supported in the area. Many frameworks already support it or are starting to support it.

In our project, we also use Docker to run docker containers for the workflows, one container per task (handled by Airflow). We created a custom local image registry to store the workflow tools as an alternative to using DockerHub, in order to privatize some of the tools used in our workflows [42].

OpenStack

OpenStack [43, 44] is an open-source cloud computing platform. It provides a set of tools to manage the infrastructure of a cloud computing environment. It is mostly deployed as infrastructure-as-a-service in both public and private clouds where virtual servers and other resources are made available to users.

The initial deployment of the application will happen in the infrastructure of Biodata [45], namely in private clouds from Instituto Superior Técnico based on OpenStack. It will provide virtual machines that host the application components and storage of large data files by using the S3 protocol [26], being a scalable and reliable storage solution. In the initial implementation present in this report, the data repository for storage of files follows the S3 protocol for this reason.

MongoDB

MongoDB [46, 47] is a document-oriented database. It is a NoSQL database, which means that it does not use the relational model. Instead, it stores data in JSON-like documents, which makes it easier to store and retrieve data.

We use MongoDB to store the metadata of the application, such as the information of the projects, datasets and other resources. It is also used to store the workflow templates, tool templates, and workflow instances; documents whose purpose is to allow for the creation of custom workflows and tools.

Mongo's schema-less nature allows for documents in the same collection to have different and varying fields. In our application, this blends nicely into our need to have multiple data representations of the same resource, being able to store the access information for these in the same document - for more in-depth information about the data model check the [4.2] section. Since the metadata documents store data specific to each data repository (representation) in object fields, these fields can't have a general schema.

Relational databases could be used to store the metadata but this would mean storing each different data repository information in a different table, altering the data model every time one is introduced or removed, simply for the sake of data which does not have relations with the rest of the database and is only intended to be read.

On another note, the workflow and tool documents do not require enforcing strong relations or constraints, and they have mostly read-only data. In addition, in the case of templates, they are not edited by the application itself, being configured only by the system administrator. Storing the templates as documents, following the JSON format, allow for flexible and easy configuration and modification of workflows by the administrator.

We choose MongoDB over other non-relational databases because we already have experience with it, but never before had we used it to the extent that we did on PHYLOViZ Web Platform, delving into its more complex features like index creation and schema validation.

Redis

Redis [48, 49] is an open-source in-memory data structure store, used as a database, cache and message broker. It is a NoSQL database, which means that it does not use the relational model. Instead, it stores data in key-value pairs, which makes it easier to store and retrieve data.

In the context of this project, Redis is utilized as a caching mechanism for storing session information of users. This allows for quick retrieval of session data, such as user authentication details. Redis's in-memory data storage capability enables faster access to session data, enhancing the overall performance and responsiveness of the web platform.

Overall, Redis was a suitable choice for our project's session management needs, offering efficient in-memory data storage, high performance, and advanced data structure support, making it a reliable and scalable solution for the PHYLOViZ Web Platform.

5.2 Frontend Technologies

The frontend technologies utilized in the PHYLOViZ Web Platform are crucial for delivering a visually appealing and user-friendly interface that enables users to interact with the system. These technologies encompass the client-side components responsible for rendering the user interface, handling user interactions, and providing a seamless user experience. In this section, we will provide an in-depth overview of the frontend technologies employed in our project, including the frameworks, libraries, and tools used for building the user interface. We will explore how these technologies enable the creation of an intuitive and responsive user interface that enhances the usability and accessibility of the PHYLOViZ Web Platform, providing a compelling user experience for our target audience.

The main programming language used in the frontend is TypeScript [50]. TypeScript is a typed superset of JavaScript that compiles to plain JavaScript. It is a strongly typed language that provides support for the latest JavaScript features, including classes, interfaces, and modules. TypeScript is a language that is easy to learn and use, and it is widely used in the development of web applications. We choose TypeScript over JavaScript because it is a more robust and scalable language, thanks to its static typing and object-oriented features, which makes it easier to develop and maintain large applications.

React

React [51] is a JavaScript library for building user interfaces. It is maintained by Facebook and a community of individual developers and companies. React can be used as a base in the development of single-page or mobile applications.

React is a component-based library, which means that the application is built by assembling components. Each component is a small piece of code that can be reused in different parts of the application. React is also declarative, which means that it is possible to describe the user interface without specifying how the user interface should be updated.

In PHYLOViZ Web Platform, we use React to create the user interface. We also use React Router to manage the routing of the application. React Router is a collection of navigational components that compose declaratively with your application.

Material-UI

In addition to React, we also use Material-UI [52] to create the user interface. Material-UI is a React component library that implements Google's Material Design, which is a design language that combines the classic principles of successful design along with innovation and technology. Material-UI provides a set of components that can be used to create a user interface that follows the Material Design guidelines, such as buttons, cards, and tables. This makes it easier to create a consistent user interface.

Webpack

Webpack [53] is a module bundler. It takes modules with dependencies and generates static assets representing those modules. Webpack is used to bundle JavaScript files for usage in a browser. It also provides a set of plugins that can be used to optimize the application, such as minification and code splitting.

In PHYLOViZ Web Platform, we use Webpack to bundle the JavaScript files of the application, optimizing them for production. The technology also provides a development server, which is used to serve the application during development. We also use ts-loader to compile TypeScript files into JavaScript.

Cosmos

Cosmos [54] is a WebGL Force Graph layout algorithm and rendering engine. All the computations and drawing happen on the GPU in fragment and vertex shaders avoiding expensive memory operations, making use of the parallelism advantage of the GPU.

It enables real-time simulation of network graphs consisting of hundreds of thousands of nodes and edges on modern hardware, being a powerful alternative and strong competitor to several other network graph visualization libraries like Vivagraph, Sigma and React-force-graph [55].

In PHYLOViZ Web Platform, we use Cosmos to render the tree visualizations. We choose Cosmos because it is a fast and open-source library that provides a set of base features that are useful for our application.

Several modifications to the base implementation needed to be made in order to allow for certain required features regarding transformation of the visualization. Some of these new features include: dragging of nodes, displaying text on top of each node and edges, and more complex color/pattern application on each node (specifically a multi-color pie chart on each node). Some of these modifications are to be added as official contributions to the cosmos repository.

5.3 DevOps Technologies

In this section we describe the technologies used in the DevOps pipeline of the application. These tools play a crucial role in enhancing collaboration, automating tasks, and ensuring code quality throughout the development lifecycle. We will provide an overview of their purpose and significance, as well as how they are used in the context of our project. By incorporating these tools into our development workflow, we aim to streamline our processes and improve the efficiency and effectiveness of our software development efforts.

Git and GitHub

Git [56, 57] and GitHub [58, 59] are widely used version control tools that play a critical role in modern DevOps practices. Git is a distributed version control system that allows teams to efficiently manage changes to source code, collaborate, and track code changes over time. GitHub, on the other hand, is a web-based hosting service for Git repositories that provides additional collaboration and project management features.

In our project, for the first time, we made use of the multi-branch feature of git, creating new branches when adding separate sets of improvements, maintaining the master branch clean and untouched until clear intention of including the newly added code to the main code. Pull requests were used to notify each other of changes that are intended to be added to the main branch, allowing for the discussion and reviewing of changes.

GitHub Projects [60] was also used to organize and track completion of tasks, creating new task items that fall into the categories “Todo”, “In Progress” and “Done”. Each task can also be associated with an issue, triggering its automatic completion upon closing of the issue.

GitHub Actions

GitHub Actions [61, 62] is a powerful and versatile continuous integration and deployment tool that is tightly integrated with GitHub, providing automated workflows for building, testing, and deploying code changes. With GitHub Actions, teams can define custom workflows that are triggered by events, such as pushes, pull requests, or other actions, to automatically perform tasks and streamline their software development processes.

This tool has been a valuable addition to our DevOps toolkit, providing automated workflows that enable efficient and automated software development processes. Its integration with GitHub allows for seamless collaboration and code review, and its flexibility and scalability make it a powerful tool for our continuous integration and deployment needs.

Postman

Postman [63] is a widely-used software tool that played a vital role in our software stack. It allowed us to define and document APIs effectively, making it easier for developers to understand and interact with our application programming interfaces. Additionally, Postman provided an intuitive user interface for manual testing, enabling us to thoroughly test our API endpoints and ensure their correctness and reliability.

We chose to use Postman in this project due to our previous positive experience with the software and its suitability for our needs. It allowed us to design, describe, and share API specifications seamlessly. Moreover, Postman's comprehensive features, such as request collections and test scripts, streamlined our testing efforts and facilitated collaboration among team members. Overall, Postman proved to be an indispensable tool in our software development lifecycle, empowering us to build robust and well-tested APIs.

diagrams.net

diagrams.net/draw.io [64] is a popular online diagramming tool that allows users to create a wide variety of diagrams, including flowcharts, process diagrams, UML diagrams, and more. It offers a user-friendly interface with a wide range of shapes, icons, and connectors that make it easy to create visually appealing and professional-looking diagrams. Draw.io also provides powerful collaboration features, allowing multiple team members to work on the same diagram in real-time, making it an ideal tool for team-based diagramming tasks.

In our project, we chose Draw.io as our preferred diagramming tool due to its intuitive and user-friendly interface, extensive shape library, and powerful collaboration features. Draw.io allowed us to create high-quality diagrams quickly, even for team members without extensive diagramming experience. Its collaborative features facilitated real-time collaboration, making it easy to share, discuss, and revise diagrams together. Overall, Draw.io proved to be a valuable tool in PHYLOViZ Web Platform, enabling us to create professional-looking diagrams efficiently and collaboratively.

Chapter 6

Implementation

In this chapter we describe in more detail each component of PHYLOViZ Web Platform, how they interact, their functionalities, the technologies that support them and implementation details.

6.1 Frontend Application

The frontend application, as described in Section 4.3, is composed of a web application, which is responsible for the visualization of the data and the interaction with the user. This application provides a simple and intuitive interface for the user to interact with the system, allowing it to manage projects and datasets, upload files, execute computation tasks, and visualize the results, among other functionalities.

The frontend client is implemented using React with TypeScript. To read more about the technologies used in the frontend, see section 5.2.

Application Structure

The structure of the frontend application is the following:

- `src`: contains the source code of the application;
- `public`: contains the static files of the application;
- `package.json`: contains the dependencies of the application;
- `tsconfig.json`: contains the TypeScript configuration;
- `webpack.config.js`: contains the Webpack configuration.

The source code of the application is divided into multiple folders, which are responsible for different functionalities of the application:

- `Assets`: contains the assets of the application, such as images and icons;
- `Components`: contains the components of the application;

- `Layouts`: contains the layouts of the application, which are used to define the structure of the pages;
- `Pages`: contains the pages of the application; a page contains multiple components;
- `Services`: contains the services of the application that are responsible for communicating with the backend application;
- `Session`: contains components and structures used to define and maintain the user session;
- `Utils`: contains the utility functions of the application.

The `index.tsx` file is the entry point of the frontend application, while the `App.tsx` contains the main component of the React application.

6.1.1 Services

The services in the frontend application are responsible for communicating with the backend application. Each backend microservice has a corresponding service in the frontend application, which acts as a namespace containing functions responsible for handling each request to the backend. These functions are asynchronous and return a promise that resolves to the response of the request.

To facilitate communication with the backend, the Fetch API [65] is used, which is a modern replacement for the older `XMLHttpRequest`. The Fetch API provides client functionality for transferring data between a client and a server, and is known for its simplicity and cleanliness, using promises instead of callbacks.

To simplify the process of making fetch calls, a function named `apiFetch` was developed. This function accepts the request parameters, returning a promise that resolves to the response from the request. It handles both sending the request and processing the response, including validation and error handling if necessary.

Moreover, to enhance the abstraction of the `apiFetch` function, additional functions were implemented to encapsulate the specific request methods utilized throughout the application, such as GET, POST, PUT, and DELETE. These functions further streamline the usage of the `apiFetch` function for each respective request method.

To facilitate testing of the services without requiring the backend application, mock services were implemented. These mock services return mock data instead of sending requests to the backend application. This is beneficial for testing the frontend application without needing the backend application to be running, making the development process faster and allowing for decoupling of the frontend from the backend.

6.1.2 Components

In this section we describe some relevant components of the React application.

App

The `App` component serves as the root component of the application. It plays a crucial role in defining and managing the application's routes using the React Router library.

This component is utilized in the `index.tsx` file, which acts as the entry point for the application. Within this file, the `ReactDOM` object's render method is invoked to facilitate the rendering of the application within the DOM.

Dashboard

The `Dashboard` component serves as the central layout for the application, encompassing the header and sidebar functionalities. It also acts as a container for the child components specific to the current page being displayed.

The header section prominently features the application's logo and name. Additionally, it incorporates a user menu that provides convenient access to various user-related operations, including profile management, settings configuration, and logout functionality.

The sidebar component houses the navigation menu, which offers links to the different pages within the application. This menu enables seamless navigation between the various sections and functionalities of the platform.

Auth Provider

The `AuthProvider` component plays a crucial role in user authentication and the storage of authentication information within the application state. To facilitate this functionality, it leverages the React Context API, which eliminates the need to manually send data from one component to another at each level of the application, enabling seamless data sharing between components.

The `Session` interface represents the user session, encapsulating relevant authentication details. On the other hand, the `SessionManager` interface acts as a wrapper around the `Session` interface, providing essential methods to effectively manage the user session. These methods include `setSession`, which sets the current session, and `clearSession`, responsible for clearing the session.

The `AuthProvider` component wraps its child components within the `SessionManagerContext.Provider`, providing the session manager instance and its associated methods as the context value. This facilitates the availability of authentication data throughout the component tree, allowing child components to access and utilize the user session information as needed.

Project

The `Project` component serves as a central component within the application, functioning as the project page. It leverages the `useProject` hook, responsible for retrieving project data from the backend application and managing it within the application state. The component comprises two essential sections: the project structure and the project content.

The project structure utilizes the `TreeView` [66] component provided by the Material-UI library, enabling the hierarchical display of project files and folders. It also facilitates various actions that can be performed on these elements, such as creating, renaming, and deleting.

On the other hand, the project content dynamically renders based on the current route using the React Router library. This is achieved through the utilization of an *outlet*, which serves as a placeholder for the content specific to the current route.

Outlets play a crucial role in enabling the rendering of different components within the project content section. They provide a flexible mechanism for dynamically displaying appropriate content based on the route. The `Project` component passes down a context to the outlet, containing essential information such as the current project, updates to file structure, and updates to workflows. This context empowers the outlet to interact with and manipulate the project data effectively, ensuring a seamless user experience.

Typing Data and Isolate Data

The `TypingData` and `IsolateData` components serve as outlets within the project page. These components leverage the powerful features provided by the `DataGrid` [67] component from Material-UI to display and manipulate data in a tabular format.

The `DataGrid` component is configured within these components to display the retrieved data in an organized manner. It supports various features, such as pagination, sorting, and filtering, out of the box. The columns and rows of the `DataGrid` are populated based on the data retrieved from the backend, allowing users to interact with and explore the typing or isolate data effectively.

To retrieve the necessary data for these components and manage the associated loading and error states, the `useTypingData` and `useIsolateData` hooks are utilized, respectively. These hooks handle the communication with the backend application, fetching the required data and handling any potential errors. They also provide convenient functions to clear errors and manage the loading state.

Compute

The `Compute` components are responsible for configuring computations within the application. These components allow users to specify various parameters and settings required for performing specific algorithms or calculations. They play a crucial role in enabling users to customize and control the computational processes according to their specific needs. One such example is the `GoeBURSTConfig` page, which provides a user interface for configuring the goeBURST algorithm that computes a tree.

To handle the computation itself, the `useCompute` hook is employed. This hook manages the communication with the backend application, specifically for creating workflows associated with the computation. In addition, it interacts with the `useProjectContext` hook to access the current project and updates to workflows.

The combination of these components and hooks enables users to configure and initiate computations within the application. By providing intuitive interfaces, error handling, and navigation capabilities, these components contribute to a seamless user experience when working with computational algorithms and calculations.

6.1.3 Tree Visualization

The `TreeView` component is one of the most complex components in the application. It is responsible for the tree visualization, including rendering and application of transformations to the visualization.

The `TreeView` component utilizes various visual components and libraries to achieve its functionality. One notable library is `cosmos.js`, used for the force-directed layout. It enables the manipulation and transformation of the tree visualization, providing features such as zooming, panning, and applying various graph algorithms for layout and organization.

To create the graph visualization within the `TreeView` component, the `canvasRef` is utilized. This reference to the canvas element is used to render the graph and apply transformations, allowing for interactive exploration of the tree structure.

In addition to the `cosmos.js` library, the `TreeView` component also leverages the functionality provided by the `chart.js` library. `chart.js` provides charting capabilities for rendering the tree visualization. The `ChartOptions` object is utilized to configure the chart's options, such as the position of the legend. The `Doughnut` component from the `react-chartjs-2` library is also used to display a doughnut chart alongside the tree visualization.

Within the `TreeView` component, there are multiple sub-components responsible for different aspects of the tree visualization. These sub-components include:

- `TreeViewInfoCard`: This component displays information about the tree view, utilizing the `treeView` object obtained from the `useTreeView` hook.

- `TreeViewSearchCard`: This component allows users to search for specific elements within the tree visualization.
- `TreeViewSettingsCard`: This component provides various settings and controls for customizing the tree visualization. It includes options to pause or restart the animation, reset simulation configurations and filters, and adjust parameters related to the visualization, such as link spring, gravity, and node size. It also includes all the options for the transformations to the tree visualization which are relevant to the phylogenetic analysis, like showing node labels, edge labels, and displaying colors and pie-charts on top of each node based on the profile's isolates.

These sub-components are rendered within the `TreeView` component based on the current state and user interactions. The `useTreeView` hook is utilized to manage the state and provide functions for updating the tree visualization and handling user interactions.

Additionally, the `TreeView` component utilizes the `toPrintRef` reference to facilitate printing the tree visualization.

Improvements to Cosmos.js

Several enhancements were implemented in the `cosmos.js` library to enhance the functionality of the `TreeView` component. These improvements include the implementation of node and link labels, node dragging, and the integration of pie charts on the nodes.

One of the main challenges faced during these implementations was the need for efficient rendering, especially when visualizing thousands of nodes. To address this, a batching technique was used to render all elements in a single draw call. This optimization allowed for smooth performance and improved the user experience when exploring large tree structures.

The addition of node and link labels provides contextual information to the tree visualization, enhancing the understanding of the relationships between nodes. Users can easily identify and interpret specific nodes and links based on the provided labels.

The implementation of node dragging enables users to interactively manipulate the tree structure. This feature enhances the flexibility and interactivity of the visualization, allowing users to rearrange nodes and explore different tree configurations.

One of the notable features added to the tree visualization is the integration of pie charts on the nodes. This enhancement provides a visual representation of additional data associated with each node, such as profile isolates. The use of pie charts allows for a compact and informative display, conveying multiple data points in a single node visualization.

Through these enhancements and optimizations, the `TreeView` component, powered by `cosmos.js`, offers an interactive and customizable tree visualization experience, enabling users to explore and analyze complex tree structures effectively.

6.1.4 User Interface and Navigation

The user interface of the PHYLOViZ Web Platform is thoughtfully designed to offer an intuitive and user-friendly experience for navigating and interacting with its features. To provide a clear understanding of the system's structure, this section presents a comprehensive explanation of the navigation flow. By following this flow, users can seamlessly explore and utilize the platform's functionalities.

The navigation diagram (refer to Figure B.1 in the appendices) showcases the different views and pages of the platform, highlighting their connections and the flow of navigation between them. Each section represents a specific area of functionality, while pages represent individual screens or interfaces within those sections.

In addition to the navigation diagram, a comprehensive Usage Guide [68] is available in the wiki section of our platform. This guide provides detailed instructions on how to effectively utilize the user interface and navigate through various pages. It includes step-by-step explanations accompanied by screenshots of different screens, ensuring a seamless user experience while exploring the platform's functionalities.

Homepage and Authentication

Upon accessing the home page, not authenticated users have access to information about the platform, including the about page, getting started guide, and other project documentation. They also have the option to sign up or sign in to their account.

After signing in, the user is redirected back to the home page, now equipped with enhanced privileges, having access to additional features, such as viewing their profile, creating new projects, and accessing existing ones. Clicking on a project redirects the user to the dedicated project page, where they can leverage a range of project-related functionalities.

Project Page

In the project page, users can see the project structure displayed on the left side of the interface. This structure includes project files, datasets, and workflows. Users have the ability to upload files to the project, create datasets using the uploaded files, and perform various operations on the datasets.

Within a dataset, users can view detailed information about the data, such as its type and structure. They can also visualize the dataset resources, including typing data, isolate data, distance matrices, trees and tree views.

Furthermore, users can perform computations using workflows within the dataset. This includes computing distance matrices, generating trees and tree views. When visualizing data, the user can perform relevant transformations, using features like searches, filters, sortings, and other operations to manipulate and refine the visualizations.

Additionally, users have the ability to generate reports for the dataset, allowing them to compile and present the analysis results in a comprehensive manner.

By following this navigation flow, users can seamlessly explore and interact with the platform's features, from creating projects and datasets to performing computations, visualizations, and generating reports for their data. The user interface is designed to provide a cohesive and efficient experience throughout the entire workflow.

6.2 Gateway

The Gateway is implemented using Spring Cloud Gateway, a reactive web server that handles routing requests and provides cross-cutting concerns like security, monitoring/-metrics, and resiliency. It is easy to configure and extend, making it a suitable choice for the system.

The Gateway's configuration, specified in the `GatewayConfig` class, includes properties for customization and injection. Security is configured using Spring Security and OAuth2, with rules for authentication and authorization. The `springSecurityFilterChain` bean defines security filters and handles various security aspects. The Gateway also supports logout handling, CSRF protection (which is disabled for now), and OAuth2 login.

The `getSession` endpoint, implemented in the `AuthController` class, allows clients to retrieve session information. It verifies the authentication token and extracts the user's name, email, and picture. The response contains this session information.

The Gateway provides additional features like security, monitoring/metrics, and resiliency. It integrates with Spring Security for authentication and authorization. It can be integrated with monitoring tools like Prometheus and Grafana for collecting and analyzing metrics. It supports circuit breakers and load balancing for resiliency.

6.3 Microservices

In this section, we offer a thorough description of the implementation of each microservice in the system. For a more comprehensive understanding of the microservices architecture, please consult the section on microservices architecture [4.5].

All microservices are implemented using Spring Boot, a framework that simplifies the development of stand-alone, production-grade Spring-based applications. Spring Boot offers a wide range of features and integrations that facilitate the creation of robust and scalable microservices.

6.3.1 Common Implementation

The microservices in our system share a common implementation pattern, ensuring consistency and ease of maintenance across the different components. This common implementation pattern is based on a layered architecture [69], comprising the HTTP layer, Services layer, and Repository layer.

Components within this architecture pattern are organized into horizontal layers, each layer performing a specific role within the application (e.g., presentation logic or business logic). Although the layered architecture pattern does not specify the number and types of layers that must exist in the pattern, most layered architectures consist of four standard layers: presentation, business, persistence, and database.

The HTTP layer serves as the entry point for communication with the microservice. It handles incoming requests and routes them to the appropriate service methods for further processing. The HTTP layer also handles response formatting and status code management, providing a seamless interaction between the microservice and external clients.

The Services layer encapsulates the business logic and application-specific operations of each microservice. It provides a set of services that define the core functionalities and operations supported by the microservice. This layer acts as an intermediary between the HTTP layer and the underlying data storage or external services, ensuring the appropriate handling of requests and orchestrating the necessary operations.

The Repository layer is responsible for interacting with the data storage or external services required by the microservice. It provides an abstraction over the underlying data access mechanisms, such as databases or third-party APIs. The Repository layer handles data persistence, retrieval, and any necessary transformations, allowing the Services layer to focus on business logic without being tightly coupled to the specific data storage implementation.

By adhering to this layered architecture pattern, the microservices in our system exhibit modularity, maintainability, and scalability. The separation of concerns provided by this pattern enables easier testing, debugging, and extension of the system's functionalities. It also allows for independent development and deployment of each microservice, fostering a more flexible and resilient architecture.

Metadata Repositories

section under construction...

Data Repositories

section under construction...

In the following subsections, we delve into the specific implementation details of each microservice, highlighting their unique functionalities and interactions within the system.

6.3.2 Administration Microservice

The Administration microservice plays a crucial role in managing the projects within the system. It provides functionalities for creating, updating, and deleting projects, as well as handling project-related operations.

The implementation of the Administration microservice follows the layered architecture pattern described earlier, with the HTTP, Services, and Repository layers working in concert to achieve its functionalities.

At the HTTP layer, the Administration microservice exposes a set of endpoints that allow clients to interact with the system's project management capabilities. These endpoints serve as the entry points for various operations, enabling users to manage projects, datasets, and more.

Table 6.1 provides a comprehensive overview of the available endpoints and their corresponding operations. It summarizes the API endpoints offered by the Administration microservice in relation to projects.

Endpoint	**HTTP Method**	**Description**
/projects	GET	Retrieve a list of all projects
/projects	POST	Create a new project
/projects/{projectId}	GET	Retrieve details of a specific project
/projects/{projectId}	PATCH	Update a specific project
/projects/{projectId}	DELETE	Delete a project

Table 6.1: API endpoints provided by the Administration microservice related to projects.

Table 6.2 presents a summary of the API endpoints provided by the Administration microservice for managing datasets, which are crucial components of projects. It outlines the available operations for datasets within the microservice.

Endpoint	HTTP Method	Description
/projects/{projectId}/datasets	POST	Create a new dataset
/projects/{projectId}/datasets/{datasetId}	GET	Retrieve details of a specific dataset
/projects/{projectId}/datasets/{datasetId}	PATCH	Update a specific dataset
/projects/{projectId}/datasets/{datasetId}	DELETE	Delete a dataset

Table 6.2: API endpoints provided by the Administration microservice related to datasets.

As mentioned earlier, a dataset consists of various resources, such as distance matrices, trees, and tree views. The Administration microservice also facilitates operations to update (change the name) and delete these dataset resources.

6.3.3 FileTransfer Microservice

The FileTransfer microservice is responsible for handling file transfers within the system. It provides endpoints for uploading and downloading files, specifically focusing on two types of files: typing data files and isolate data files.

Table 6.3 provides a summary of the API endpoints offered by the FileTransfer microservice. It outlines the available endpoints for interacting with the microservice.

Endpoint	HTTP Method	Description
/projects/{projectId}/files/typing-data	POST	Upload typing data file
/projects/{projectId}/files/typing-data/{typingDataId}/file	GET	Download typing data file
/projects/{projectId}/files/isolate-data	POST	Upload isolate data file
/projects/{projectId}/files/isolate-data/{isolateDataId}/file	GET	Download isolate data

Table 6.3: API endpoints provided by the FileTransfer microservice.

6.3.4 Compute Microservice

The Compute microservice is a crucial component responsible for executing the computation tasks within the system. It handles tasks such as calculating distance matrices, phylogenetic trees, and visualizations. The microservice leverages the FLOWViZ Workflows Management System (WMS) [8] for executing workflows.

Table 6.4 summarizes the three API endpoints that have been defined for the Compute microservice. These endpoints enable clients to interact with the computation functionalities provided by the microservice.

Endpoint	HTTP Method	Description
/projects/{projectId}/workflows	GET	Retrieve workflows for a specific project
/projects/{projectId}/workflows	POST	Create a new workflow for a specific project
/projects/{projectId}/workflows/ {workflowId}/status	GET	Retrieve the status of a specific workflow

Table 6.4: API endpoints provided by the Compute microservice for computation tasks.

Workflows

section under construction...

6.3.5 Visualization Microservice

The Visualization microservice is responsible for retrieving the files of the projects, including the distance matrices, phylogenetic trees and the respective visualizations.

Table 6.5 provides an overview of the API endpoints implemented for the Visualization microservice. These endpoints enable clients to retrieve various types of files and visualizations associated with a project.

Endpoint	HTTP Method	Description
/projects/{projectId}/datasets/{datasetId}/ trees/{treeId}	GET	Retrieve a tree
/projects/{projectId}/datasets/{datasetId}/ tree-views/{treeViewId}	GET	Retrieve a tree view
/projects/{projectId}/datasets/{datasetId}/ distance-matrices/{distanceMatrixId}	GET	Retrieve a distance matrix
/projects/{projectId}/files/isolate-data/ {isolateDataId}	GET	Retrieve isolate data keys
/projects/{projectId}/isolate-data/ {isolateDataId}/rows	GET	Retrieve isolate data rows
/projects/{projectId}/typing-data/ {typingDataId}	GET	Retrieve the typing data schema
/projects/{projectId}/typing-data/ {typingDataId}/profiles	GET	Retrieve the typing data profiles

Table 6.5: API endpoints provided by the Visualization microservice for retrieving project files and visualizations.

6.4 Access Management

The access management server is responsible for authenticating the user and issuing access tokens to the client application. It is implemented using Keycloak [39].

Keycloak is an open-source Identity and Access Management (IAM) solution that provides features such as authentication, authorization, and user management. It supports various protocols such as OAuth2, OpenID Connect, and SAML for authentication and authorization.

In the context of PHYLOViZ Web Platform, Keycloak acts as an OpenID identity provider that authenticates users and issues access tokens that can be used to access protected resources.

The Spring Cloud Gateway is used as an OpenID client, that will authenticate users using Keycloak's OIDC capabilities. Once the user is authenticated, Spring Cloud Gateway will use the Access Token received from Keycloak to forward requests to the appropriate microservice.

The microservices, as OpenID resource servers, will validate the Access Token received from Spring Cloud Gateway to ensure that the user is authorized to access the requested resource.

6.4.1 Authentication Process

In the Figure A.1, in the Appendix A, its shown the authentication sequence diagram.

The following is a high-level overview of the authentication process:

1. The user sends a request to access a protected resource;
2. Spring Cloud Gateway intercepts the request and redirects the user to Keycloak's login page;
3. The user enters their credentials on Keycloak's login page and submits the form;
4. Keycloak authenticates the user and generates an Access Token, which includes the user's identity and authorization information;
5. Keycloak sends the Access Token back to Spring Cloud Gateway through User-Agent redirection;
6. Spring Cloud Gateway receives the Access Token and stores it in a session cookie;
7. Spring Cloud Gateway forward the request to the appropriate microservice and relays the Access Token;

8. The microservice receives the request and validates the Access Token via introspection to ensure that the user is authorized to access the requested resource;
9. If the Access Token is valid, the microservice processes the request and sends a response back to Spring Cloud Gateway;
10. Spring Cloud Gateway receives the response and forwards it back to the user.

6.4.2 Configuration

The configuration of Keycloak is done using a configuration file, which is a JSON file that contains the configuration of the realm, clients, users, and roles.

The repository of PHYLOViZ Web Platform contains this configuration file, containing the client and providers required, but, since secrets are not included, you must configure them yourselves. To do so, its necessary to configure the Keycloak server, and then configure the Spring Gateway and the microservices.

Keycloak Server Configuration

To configure the Keycloak server, you must follow the following steps:

1. Login to the Keycloak server administration console;
2. Navigate to the realm where you want to create the client;
3. Click on `Clients` in the left-hand menu;
4. Select the `phyloviz-web-platform-client` client;
5. In the "Credentials" tab, press save, regenerate and then copy the `Secret` value to your clipboard.

Gateway and Microservices Configuration

To configure the Gateway and the microservices, you must follow the following steps:

1. Configure the password of Redis in `redis/redis.conf` and the Redis credentials in the `application.yml` of the gateway.
2. Open the `application.yml` file in `shared` folder and the gateway, and replace the client-secret property with the "Secret" from the Keycloak client.

After this configurations, you can start the Keycloak server, the Gateway and the microservices, and the authentication process should work.

6.5 Deployment

In this section, we will describe the deployment process for our application, outlining the steps required to set up and run the system in a production environment. Deploying the application involves configuring and launching the various components that make up the system, ensuring their proper communication and functionality. We will provide instructions on how to deploy each component, including any necessary dependencies and configurations.

Furthermore, we will share insights into our own deployment of the application, highlighting the specific technologies and infrastructure we utilized. By sharing our deployment experience, we aim to provide a practical example and showcase the choices we made to ensure a robust and scalable system. Our deployment approach takes into consideration factors such as performance, security, and maintainability to ensure the smooth operation of the application in a real-world setting. Let's dive into the details of deploying our application and the lessons we learned during the process.

6.5.1 Deployment Process

In this subsection, we outline the step-by-step process of deploying our application in a production environment. The deployment process involves several key steps that ensure the successful setup and operation of the system. We provide detailed instructions for each step, including any prerequisites and configurations required. The deployment process includes the following stages:

Prerequisites

Before proceeding with the deployment, ensure that you have the following prerequisites in place:

- Have Docker installed and running on your system.
- Clone the PHYLOViZ Web Platform repository [70].
- Clone the FLOWViZ repository, configure and deploy it following their provided tutorial.
- Clone the PhyloDB repository, configure and deploy it following their provided tutorial.
- Set up S3 buckets and ensure the file repository is functional.

Deployment Steps

Follow the steps below to deploy the application in a production environment:

1. Start with the configuration of the access management system (Keycloak) as described in the section 6.4.2.

2. Configure the application.yml files with the appropriate settings, including the FLOWViZ, S3, and PhyloDB credentials.

3. Ensure that the workflow templates of the application are also using the correct credentials for the respective data repositories. Tool templates should also be configured with the correct registry IP and network specific configurations.

4. Launch the Docker containers for the PHYLOViZ Web Platform using the docker-compose.yml file.

By following this deployment process, you will be able to set up and run our application in a production environment successfully. Make sure to carefully execute each step and address any configuration requirements specific to your deployment environment.

6.5.2 Deployed Solution

In this subsection, we will provide an overview of the deployed solution for our application. We will discuss the specific technologies, infrastructure, and architectural choices that were made to ensure a robust and scalable system. By sharing details about our deployed solution, we aim to provide a practical example that can serve as a reference for others.

The application is deployed on the web at web.phyloviz.net, which falls under the domain of the PHYLOViZ project (phyloviz.net). Additionally, we have a domain specifically dedicated to the authorization server, named auth.phyloviz.net. This domain can be utilized for authentication in other applications within the PHYLOViZ project, which may be developed in the future.

The initial deployment of the application took place in the infrastructure of Biodata [45], specifically within private clouds hosted by Instituto Superior Técnico, utilizing OpenStack. This choice was made to leverage the scalability and reliability offered by OpenStack. The infrastructure provides virtual machines that host the various components of the application, along with storage for large data files using the S3 protocol [26]. This storage solution ensures efficient handling of data and supports the application's scalability requirements. The data repository for storing files follows the S3 protocol, which aligns with the initial implementation described in this report.

By deploying our application in this manner, we have established a robust and scalable system that can handle the demands of our users. The specific technologies, infrastructure, and architectural choices made during deployment contribute to the overall performance and reliability of the system.

6.6 Extensibility

The extensibility of the PHYLOViZ Web Platform is a fundamental aspect of its design, allowing researchers to customize and expand the platform according to their specific requirements. This section explores two key elements of extensibility: creating and customizing workflows, and establishing data repositories.

By offering the flexibility to define workflows and effectively manage data, the platform becomes a versatile tool that can adapt to various research needs. Through these extensibility features, researchers can seamlessly integrate their analysis pipelines, incorporate new analysis methods, and organize and access their phylogenetic data with ease.

This section delves into the possibilities offered by the PHYLOViZ Web Platform, highlighting how researchers can leverage its extensibility to advance their research and gain deeper insights into evolutionary relationships.

6.6.1 Customize Workflows

The PHYLOViZ Web Platform offers users the flexibility to customize workflows according to their specific requirements. This customization capability allows users to define and configure procedures tailored to their needs.

To learn more about the documents used for workflows, please refer to the *Workflows* section 4.2.3. Familiarity with these documents is essential to understand how to configure workflows effectively.

By providing configuration flexibility, the PHYLOViZ Web Platform enables users to efficiently customize and adapt workflows to meet their specific requirements.

Customizing workflows, whether it involves adding, removing, or editing elements, requires configuring the following:

- Workflow templates: These templates define the tasks executed, the corresponding commands to be run, and the arguments received by each task.

- Tool templates: These templates specify the Docker image associated with each tool and other tool-related configurations, e.g. network mode.

- Tools: Certain tools themselves may require configuration to seamlessly integrate workflows into the application. These tools are auxiliary tools that handle data input for algorithms or manage application metadata [4.2.3]. For example, in our implementation, the "downloader" tool is responsible for downloading files such as typing data from an S3 bucket into the workflow's Docker volume, enabling subsequent tasks to use them as local files.

In the following sections, we will outline the steps for customizing workflows in different scenarios and provide illustrative examples.

Adding a New Workflow

To add a new workflow, several steps are required:

Workflow Templates:

1. **Create a new workflow template:** Assign a unique type to the template and provide a name and description to distinguish it from other workflows.

2. **Add the input argument schema of the workflow to the template:** Define the arguments and their types. This step ensures that badly formatted arguments are filtered out during workflow creation. These arguments are directly inputted into the command arguments of the tools, replacing placeholders.

3. **Add all the necessary tasks to the template:** Include each task with the appropriate tool (referencing its tool template) and the corresponding command to execute. Each command may contain placeholders that are replaced with input arguments during workflow creation. Specify the order of execution by including the name of the task to run afterward in the "children" field of the preceding task.

Tool Templates:

1. **Create tool templates for any new tools:** Specify the tool name, which is used in the tasks of the workflow template, and the Docker image associated with the tool. This step may not be necessary if all the required tools already exist.

Tools:

1. **Ensure the tools meet the desired requirements:** Verify that the tools can understand the specified command arguments in the tasks and perform their intended functions. Pay special attention to auxiliary tools, as they are responsible for reading and writing resource metadata. If desired, these tools should not only create metadata but also specify the source of the resource in the metadata, such as the algorithm used [4.2.1].

For example, suppose we want to add a new workflow that calculates two trees, each using a different algorithm. The following steps are necessary:

Create the workflow template and add type, name, description, and argument schema:

```
"type": "compute-two-trees",
"name": "Compute Two Trees",
"description": "Computes two trees from a single distance matrix,
each with a different algorithm.",
"arguments": {
  "distanceMatrixId": { "type": "distanceMatrixId" },
  "datasetId": { "type": "datasetId" },
  "firstTreeAlgorithm": {
    "type": "string",
    "allowedValues": [(...)]
  },
  "secondTreeAlgorithm": {
    "type": "string",
    "allowedValues": [(...)]
   }
}
```

Add all the necessary tasks to the template. The JSON representation of the tasks for this example can be found in Appendix C and includes:

- One task for downloading the distance matrix.
- Two tasks, each for calculating a tree using a different algorithm based on the arguments (`firstTreeAlgorithm` and `secondTreeAlgorithm`).
- Two tasks, one for each tree, to upload the trees to S3 and create their metadata, which includes the algorithm used.

If the required tools do not exist, they need to be created along with their tool templates. If the existing tools do not meet the desired requirements, they should be edited accordingly.

In this example, let's assume that the required tools already exist. However, for the purpose of explanation, let's say we used a single task of the "uploader" tool instead of two tasks. In that case, the "uploader" tool would need to be modified to handle the arguments for each of the different trees. If a custom tool with the same functionality were used, a tool template for it would need to be created.

Editing Existing Workflows

There are multiple possibilities for making changes to existing workflows. For example, let's consider the scenario where you want to support a new algorithm for tree computation.

If the library you are using already includes the desired algorithm and your existing workflow is designed to work generically for any algorithm, you may only need to modify the input argument schema to include the new algorithm as an allowed value.

```
{
"type": "compute-tree",
"name": "Compute Tree",
"description": "Computes a tree, given an existing distance matrix
of the dataset and the tree calculation algorithm.",
"arguments": {
  (...),
  "algorithm": {
    "type": "string",
    "allowed-values": [
      "goeburst",
      "edmonds",
      // Add new algorithm
    ]
  }
(...)
```

You may also add new arguments related to the parametrization of the algorithm, and in that case, making use of the `required` field of the argument, to specify that it's not required for all instances of the workflow (as other algorithms may not use this new parameter); and also make use of the `prefix` field, and add it to the command line of the task.

However, if the library you are using does not have the desired algorithm in its current version, simply modifying the argument schema to accept that algorithm will not suffice. In such cases, you may need to consider the following options:

1. If a new version of the library is available and it includes the desired algorithm, you can update the tool template for that library by changing the Docker image (if necessary).

2. If the algorithm is part of another library or you want to use a different library altogether, you must create a new tool template for the new library. Additionally, you would need to modify the existing workflow template to use the new library instead. It's important to note that if the new library does not include all the algorithms from the previous library or if it performs worse in terms of performance, you can create a separate workflow specifically for the new library while keeping the existing one intact. This approach ensures continued support for all algorithms at all times.

Removing Workflows

Removing a workflow is as simple as deleting its corresponding workflow template.

Frontend Changes

Configuring workflows may result in the deprecation of certain views in the frontend application. Therefore, it is necessary to make changes to the frontend to accommodate the modifications to the workflows. This involves creating or editing views that effectively call these workflows, with a particular focus on parametrization, which is closely tied to the chosen UI library, MUI, and its input components.

The specific changes required in the frontend depend on the nature of the workflow configuration:

- If a new workflow is added (i.e., a new workflow template), the corresponding operation and/or view should be implemented to invoke it.
- If an existing workflow is renamed, the frontend should reflect the updated name.
- If the input parameters of an existing workflow are modified, the views should be updated accordingly to accommodate the changes.
- If tasks are modified within a workflow, no further action is required in the frontend.

6.6.2 Customize Data Repositories

section under construction...

Chapter 7

Testing and Evaluation

In this chapter, we present the testing and evaluation processes employed for the application code. We discuss both manual and automatic testing approaches and analyze the performance differences between various implementations. Additionally, we compare the performance and features of the PHYLOViZ Web Platform with its competitors.

7.1 Code Testing

Testing the application code is crucial to ensure its reliability and functionality. In this section, we describe the approaches used for code testing.

7.1.1 Manual Testing

Manual testing involves hands-on evaluation of the application by testers. It allows for a comprehensive assessment of the user interface and functionalities. We employed the following techniques for manual testing:

- **Service mocks in the frontend:** We utilized service mocks in the frontend to substitute the backend services accessible by API. This approach enabled us to test the visual aspects of the application without requiring the server to be running. By simulating different scenarios, we could evaluate the user interface and its responsiveness effectively.

- **Postman [5.3] for API testing:** Prior to the development of the front-end application, we employed Postman to test the API through HTTP requests. This allowed us to verify the correctness of the backend functionalities and ensure proper communication between the front-end and back-end components.

By conducting manual testing, we gained valuable insights into the usability and functionality of the application from a user's perspective.

7.1.2 Automatic Testing

Automatic testing involves the use of software tools to verify the correctness and robustness of the application. This approach enables repetitive and thorough testing of the code base. We employed the following techniques for automatic testing:

- **Unitary tests:** We utilized JUnit and Mockito frameworks to perform unit testing in the backend. Unit tests assess the individual units or components of the code to ensure their correctness in isolation. By verifying the functionality of each unit, we could identify and address any bugs or issues early in the development process.

- **Integration tests:** In addition to unit tests, we conducted integration tests to evaluate the interactions between different components of the application. This type of testing ensured that the various modules and services integrated seamlessly and functioned correctly as a whole.

By employing automatic testing, we enhanced the reliability and maintainability of the application, as well as accelerated the development process.

7.2 User Acceptance Testing

User acceptance testing (UAT) is a crucial step in evaluating the application's suitability for end-users. UAT involves real users testing the application and providing feedback on its usability and functionality.

In the case of the PHYLOViZ Web Platform, UAT was conducted by a dedicated team from the Institute of Molecular Medicine (iMM). This team consisted of domain experts and potential end-users who thoroughly examined the application from their perspectives. Their expertise and insights proved invaluable in identifying usability issues, suggesting improvements, and validating the functionality of the platform. Through user acceptance testing, we obtained valuable feedback that guided us in refining the application to better align with the needs and expectations of the intended user base.

The collaborative effort between the development team and the iMM testers significantly contributed to the overall quality and user satisfaction of the PHYLOViZ Web Platform.

7.3 Performance Testing

Performance testing assesses the system's responsiveness, scalability, and stability under various workload conditions. It helps identify potential bottlenecks and optimize the application's performance.

section under construction...

7.4 Discussion

In this section, we provided a comprehensive discussion based on the findings from testing and evaluation processes. We analyzed the results of manual and automatic testing, user acceptance testing, and performance testing to draw meaningful conclusions about the application's strengths, weaknesses, and areas for improvement.

Overall, the testing and evaluation conducted in this chapter provide valuable insights into the reliability, usability, and performance of the application. These findings contribute to reinforcing the decision-making process during development and guide future enhancements of the PHYLOViZ Web Platform.

Chapter 8

Final Remarks

In this beta version of the project, we have achieved significant milestones in the development of the PHYLOViZ Web Platform. Our focus has been on designing a robust system architecture and implementing essential components to support phylogenetic analysis. The backend microservices, data repositories, and access management using the auth server have been successfully implemented. We have also defined a comprehensive API that enables seamless communication between different modules of the platform. Furthermore, the frontend development has allowed users to create and manage projects, as well as visualize phylogenetic trees using the force directed layout. These accomplishments form a strong foundation for the future advancement of the platform.

While we have made substantial progress, there are a few key tasks that remain to be completed before the final release of the PHYLOViZ Web Platform. One of our priorities is to implement additional tree layouts, specifically the radial layout and phylogram layout. These layouts will enhance the visual representation of phylogenetic trees and provide users with more flexibility in analyzing and interpreting the data. Additionally, we recognize the importance of cluster visualization in the frontend. Implementing cluster visualization will enable users to identify and explore patterns and groupings within the phylogenetic data, enhancing their understanding of the evolutionary relationships between species.

Furthermore, we are committed to improving the testability of the system. Thorough testing and debugging processes will be undertaken to identify and address any existing issues or potential vulnerabilities, ensuring the reliability and robustness of the platform.

Looking ahead, our next steps involve finalizing these remaining tasks and delivering a fully functional PHYLOViZ Web Platform. We will prioritize the implementation of the radial layout, phylogram layout, and cluster visualization in the frontend, expanding the visual capabilities of the platform and providing users with a comprehensive set of tools for analysis. Concurrently, we will focus on improving the testability of the system through comprehensive testing and evaluation, ensuring the platform's reliability in handling various scenarios and datasets. Finally, we will conclude the project by finalizing the project report, documenting our progress and findings, and creating a comprehensive wiki that provides detailed information and guidance for users of the PHYLOViZ Web Platform. These next steps will bring us closer to delivering a reliable, user-friendly tool for phylogenetic analysis.

References

[1] Joseph Felsenstein and Joseph Felenstein. *Inferring phylogenies*. Vol. 2. Sinauer associates Sunderland, MA, 2004.

[2] Daniel H Huson, Regula Rupp, and Celine Scornavacca. *Phylogenetic networks: concepts, algorithms and applications*. Cambridge University Press, 2010.

[3] Alexandre P Francisco, Cátia Vaz, Pedro T Monteiro, José Melo-Cristino, Mário Ramirez, and Joao A Carriço. "PHYLOViZ: phylogenetic inference and data visualization for sequence based typing methods". In: *BMC bioinformatics* 13 (2012), pp. 1–10.

[4] Marta Nascimento, Adriano Sousa, Mário Ramirez, Alexandre P Francisco, João A Carriço, and Cátia Vaz. "PHYLOViZ 2.0: providing scalable data integration and visualization for multiple phylogenetic inference methods". In: *Bioinformatics* 33.1 (2017), pp. 128–129.

[5] Bruno Ribeiro-Gonçalves, Alexandre P Francisco, Cátia Vaz, Mário Ramirez, and João André Carriço. "PHYLOViZ Online: web-based tool for visualization, phylogenetic inference, analysis and sharing of minimum spanning trees". In: *Nucleic acids research* 44.W1 (2016), W246–W251.

[6] A. Francisco, C. Vaz, P. Monteiro, J. Melo-Cristino, M. Ramirez, and J. A. Carriço. *PHYLOViZ*. www.phyloviz.net, 2014. URL: `https://www.phyloviz.net/` (Retrieved 03/02/2023).

[7] Bruno Lourenço. "A framework for large scale phylogenetic analysis". In: *arXiv:2012.13363 [q-bio]* (Jan. 2021). URL: `https://arxiv.org/abs/2012.13363` (Retrieved 03/02/2023).

[8] Miguel Luis and Catia Vaz. "FLOWViZ: Framework for Phylogenetic Processing". In: *arXiv:2211.15282 [cs]* (Nov. 2022). URL: `https://arxiv.org/abs/2211.15282` (Retrieved 03/02/2023).

[9] Mark Richards and Neal Ford. *Fundamentals of software architecture: an engineering approach*. O'Reilly Media, 2020.

[10] Naruya Saitou. *Introduction to evolutionary genomics*. Springer, 2013.

[11] D Ashley Robinson, Edward J Feil, and Daniel Falush. *Bacterial population genetics in infectious disease*. John Wiley & Sons, 2010.

[12] John M Butler. *Advanced topics in forensic DNA typing: methodology*. Academic press, 2011.

[13] National Institutes of Health et al. “What are single nucleotide polymorphisms (SNPs)”. In: *Genetics Home Reference-NIH. US National Library ofMedicine. URL: https://ghr. nlm. nih. gov/primer/genomicresearch/snp* (2019).

[14] Cátia Vaz, Marta Nascimento, João A Carriço, Tatiana Rocher, and Alexandre P Francisco. “Distance-based phylogenetic inference from typing data: a unifying view”. In: *Briefings in Bioinformatics* 22.3 (2021), bbaa147.

[15] Alexandre P Francisco, Miguel Bugalho, Mário Ramirez, and João A Carriço. “Global optimal eBURST analysis of multilocus typing data using a graphic matroid approach”. In: *BMC bioinformatics* 10 (2009), pp. 1–15.

[16] Bruno Ribeiro-Gonçalves, Alexandre P Francisco, Cátia Vaz, Mário Ramirez, and João André Carriço. *PHYLOViZ Online*. online.phyloviz.net. URL: `https://online.phyloviz.net/` (Retrieved 05/02/2023).

[17] Luana Silva. “Library of efficient algorithms for phylogenetic analysis”. In: *arXiv:2012.12697 [cs]* (Dec. 2020). URL: `https://arxiv.org/abs/2012.12697` (Retrieved 03/02/2023).

[18] Daniel H. Huson and David Bryant. “Application of Phylogenetic Networks in Evolutionary Studies”. In: *Molecular Biology and Evolution* 23.2 (Oct. 2005), pp. 254–267. ISSN: 0737-4038. DOI: `10.1093/molbev/msj030`. eprint: `https://academic.oup.com/mbe/article-pdf/23/2/254/3894375/msj030.pdf`. URL: `https://doi.org/10.1093/molbev/msj030`.

[19] A Dereeper et al. “Phylogeny.fr: robust phylogenetic analysis for the non-specialist”. en. In: *Nucleic Acids Res* 36.Web Server issue (Apr. 2008), W465–9.

[20] Frédéric Lemoine et al. “NGPhylogeny.fr: new generation phylogenetic services for non-specialists”. In: *Nucleic Acids Research* 47.W1 (Apr. 2019), W260–W265. ISSN: 0305-1048. DOI: `10.1093/nar/gkz303`. eprint: `https://academic.oup.com/nar/article-pdf/47/W1/W260/28879782/gkz303.pdf`. URL: `https://doi.org/10.1093/nar/gkz303`.

[21] Zhemin Zhou et al. “GrapeTree: visualization of core genomic relationships among 100,000 bacterial pathogens”. en. In: *Genome Res* 28.9 (July 2018), pp. 1395–1404.

[22] Zhemin Zhou et al. “The EnteroBase user’s guide, with case studies on Salmonella transmissions, Yersinia pestis phylogeny, and Escherichia core genomic diversity”. In: *Genome research* 30.1 (2020), pp. 138–152.

[23] Shanshan Li et al. “Understanding and addressing quality attributes of microservices architecture: A Systematic literature review”. In: *Information and software technology* 131 (2021), p. 106449.

[24] Roy Fielding and Julian Reschke. *Hypertext transfer protocol (HTTP/1.1): Semantics and content.* Tech. rep. Internet Engineering Task Force (IETF), 2014.

[25] *Labeled Property Graph Model.* GitHub. URL: `https://github.com/phyloviz/PhyloDB/wiki/Labeled-Property-Graph-Model` (Retrieved 05/26/2023).

[26] AWS. *Cloud Object Storage — Store & Retrieve Data Anywhere — Amazon Simple Storage Service.* Amazon Web Services, Inc., 2018. URL: `https://aws.amazon.com/s3/` (Retrieved 03/15/2023).

[27] James E McDonough and James E McDonough. “Adapter Design Pattern”. In: *Object-Oriented Design with ABAP: A Practical Approach* (2017), pp. 191–205.

[28] Steven John Metsker and William C Wake. *Design patterns in Java.* Addison-Wesley Professional, 2006, pp. 17–31.

[29] Dick Hardt. *The OAuth 2.0 authorization framework.* Tech. rep. Internet Engineering Task Force (IETF), 2012.

[30] *Web Application Development — Instituto Superior de Engenharia de Lisboa.* www.isel.pt. URL: `https://www.isel.pt/en/leic/web-application-development` (Retrieved 05/17/2023).

[31] *Systems Virtualization Techniques — Instituto Superior de Engenharia de Lisboa.* www.isel.pt. URL: `https://www.isel.pt/en/leic/systems-virtualization-techniques` (Retrieved 05/17/2023).

[32] *Software Laboratory — Instituto Superior de Engenharia de Lisboa.* www.isel.pt. URL: `https://www.isel.pt/en/leic/software-laboratory` (Retrieved 05/17/2023).

[33] *Software Development Techniques — Instituto Superior de Engenharia de Lisboa.* www.isel.pt. URL: `https://www.isel.pt/en/leic/software-development-techniques` (Retrieved 05/17/2023).

[34] Ken Arnold, James Gosling, and David Holmes. *The Java programming language.* Addison Wesley Professional, 2005.

[35] Stephen Samuel and Stefan Bocutiu. *Programming kotlin.* Packt Publishing Ltd, 2017.

[36] Craig Walls. *Spring in Action, Sixth Edition.* Simon and Schuster, Apr. 2022.

[37] Craig Walls. *Spring Boot in action.* Simon and Schuster, 2015.

[38] John Carnell and Illary Huaylupo Sánchez. *Spring microservices in action.* Simon and Schuster, 2021.

[39] Stian Thorgersen and Pedro Igor Silva. *Keycloak - Identity and Access Management for Modern Applications: Harness the power of Keycloak, OpenID Connect, and OAuth 2.0 protocols to secure applications.* Packt Publishing, 2023.

[40] David Recordon and Drummond Reed. “OpenID 2.0: a platform for user-centric identity management”. In: *Proceedings of the second ACM workshop on Digital identity management.* 2006, pp. 11–16.

[41] Jeffrey Nickoloff and Stephen Kuenzli. *Docker in Action.* Simon and Schuster, Oct. 2019.

[42] *Deploy a registry server — Docker.* Docker Documentation. URL: `https://docs.docker.com/registry/deploying/` (Retrieved 03/15/2023).

[43] Omar Sefraoui, Mohammed Aissaoui, Mohsine Eleuldj, et al. “OpenStack: toward an open-source solution for cloud computing”. In: *International Journal of Computer Applications* 55.3 (2012), pp. 38–42.

[44] openstack. *Build the future of Open Infrastructure.* OpenStack, 2010. URL: `https://www.openstack.org/`.

[45] *Home — BioData.pt.* biodata.pt. URL: https://biodata.pt/ (Retrieved 05/10/2023).

[46] Kyle Banker, Douglas Garrett, Peter Bakkum, and Shaun Verch. *MongoDB in action: covers MongoDB version 3.0.* Simon and Schuster, 2016.

[47] MongoDB. *The most popular database for modern apps.* MongoDB. URL: https://www.mongodb.com/ (Retrieved 03/22/2023).

[48] Josiah Carlson. *Redis in action.* Simon and Schuster, 2013.

[49] Redis. *Redis.* redis.io. URL: https://redis.io/ (Retrieved 03/22/2023).

[50] Gavin Bierman, Martıén Abadi, and Mads Torgersen. "Understanding typescript". In: *ECOOP 2014–Object-Oriented Programming: 28th European Conference, Uppsala, Sweden, July 28–August 1, 2014. Proceedings 28.* Springer. 2014, pp. 257–281.

[51] Mark Thomas. *React in Action.* Simon and Schuster, May 2018.

[52] MUI. *MUI: The React component library you always wanted.* mui.com. URL: https://mui.com/ (Retrieved 03/23/2023).

[53] *webpack.* webpack. URL: https://webpack.js.org/ (Retrieved 03/23/2023).

[54] Nikita Rokotyan, Olga Stukova, and Denis Ovsyannikov. *Cosmograph: GPU-accelerated Force Graph Layout and Rendering.* Version 1.3.0. June 2022. URL: https://github.com/cosmograph-org/cosmos (Retrieved 04/05/2023).

[55] *Cosmos — Comparison with other network graph visualization libraries.* www.youtube.com. URL: https://www.youtube.com/watch?v=HWk78hP8aEE (Retrieved 04/28/2023).

[56] Scott Chacon and Ben Straub. *Pro git.* Springer Nature, 2014.

[57] Git. *Git.* Git-scm.com. URL: https://git-scm.com/ (Retrieved 03/23/2023).

[58] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. "Social coding in GitHub: transparency and collaboration in an open software repository". In: *Proceedings of the ACM 2012 conference on computer supported cooperative work.* 2012, pp. 1277–1286.

[59] GitHub. *GitHub.* GitHub, Inc. URL: https://github.com/.

[60] *Planning and tracking with Projects.* GitHub Docs. URL: https://docs.github.com/en/issues/planning-and-tracking-with-projects (Retrieved 03/23/2023).

[61] Chaminda Chandrasekara, Pushpa Herath, Chaminda Chandrasekara, and Pushpa Herath. "Introduction to github actions". In: *Hands-on GitHub Actions: Implement CI/CD with GitHub Action Workflows for Your Applications* (2021), pp. 1–8.

[62] *Features - GitHub Actions.* GitHub, Inc., 2018. URL: https://github.com/features/actions (Retrieved 03/23/2023).

[63] Postman. *Postman — The Collaboration Platform for API Development.* Postman, 2021. URL: https://www.postman.com/ (Retrieved 05/28/2023).

[64] *Diagram Software and Flowchart Maker.* www.diagrams.net. URL: https://www.diagrams.net/ (Retrieved 03/23/2023).

[65] *Fetch API - Web APIs — MDN.* developer.mozilla.org. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (Retrieved 05/24/2023).

[66] *Tree View React component - Material UI*. mui.com. URL: `https://mui.com/material-ui/react-tree-view/` (Retrieved 05/24/2023).

[67] *React Data Grid component - MUI X*. mui.com. URL: `https://mui.com/x/react-data-grid/` (Retrieved 05/24/2023).

[68] *Usage Guide*. GitHub. URL: `https://github.com/bodybuilders-team/phyloviz-web-platform/wiki/usage-guide` (Retrieved 05/31/2023).

[69] Mark Richards. *Software architecture patterns*. Vol. 4. O'Reilly Media, Incorporated 1005 Gravenstein Highway North, Sebastopol, CA . . ., 2015.

[70] *PHYLOViZ Web Platform*. GitHub, May 2023. URL: `https://github.com/bodybuilders-team/phyloviz-web-platform` (Retrieved 06/03/2023).

Appendices

Appendix A

Authentication Sequence

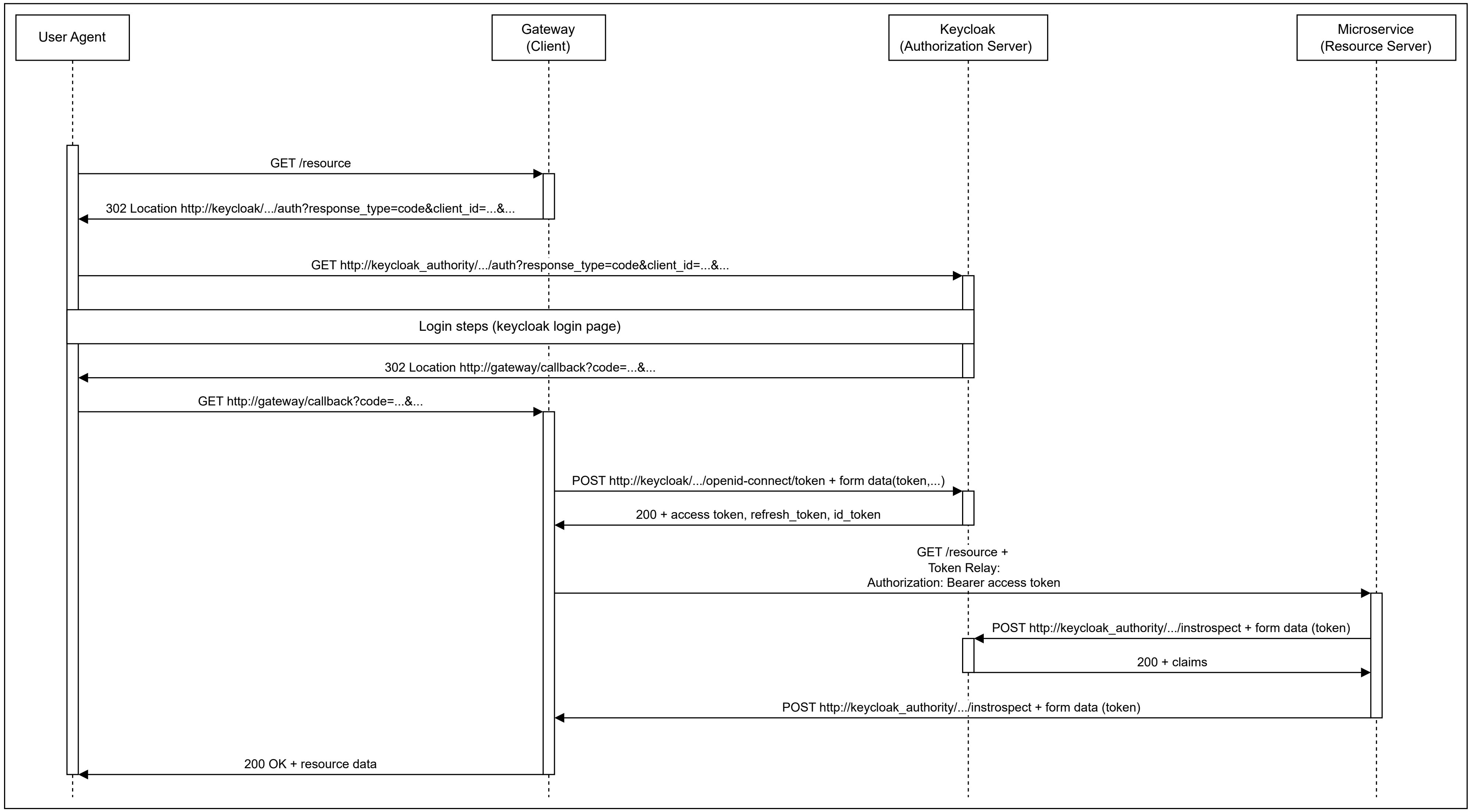

User Agent
Gateway (Client)
Keycloak (Authorization Server)
Microservice (Resource Server)
GET /resource
302 Location http://keycloak/.../auth?response_type=code&client_id=...&...
GET http://keycloak_authority/.../auth?response_type=code&client_id=...&...
Login steps (keycloak login page)
302 Location http://gateway/callback?code=...&...
GET http://gateway/callback?code=...&...
POST http://keycloak/.../openid-connect/token + form data(token,...)
200 + access token, refresh_token, id_token
GET /resource + Token Relay: Authorization: Bearer access token
POST http://keycloak_authority/.../instrospect + form data (token)
200 + claims
POST http://keycloak_authority/.../instrospect + form data (token)
200 OK + resource data

Figure A.1: Authentication sequence.

Appendix B

Navigation Diagram

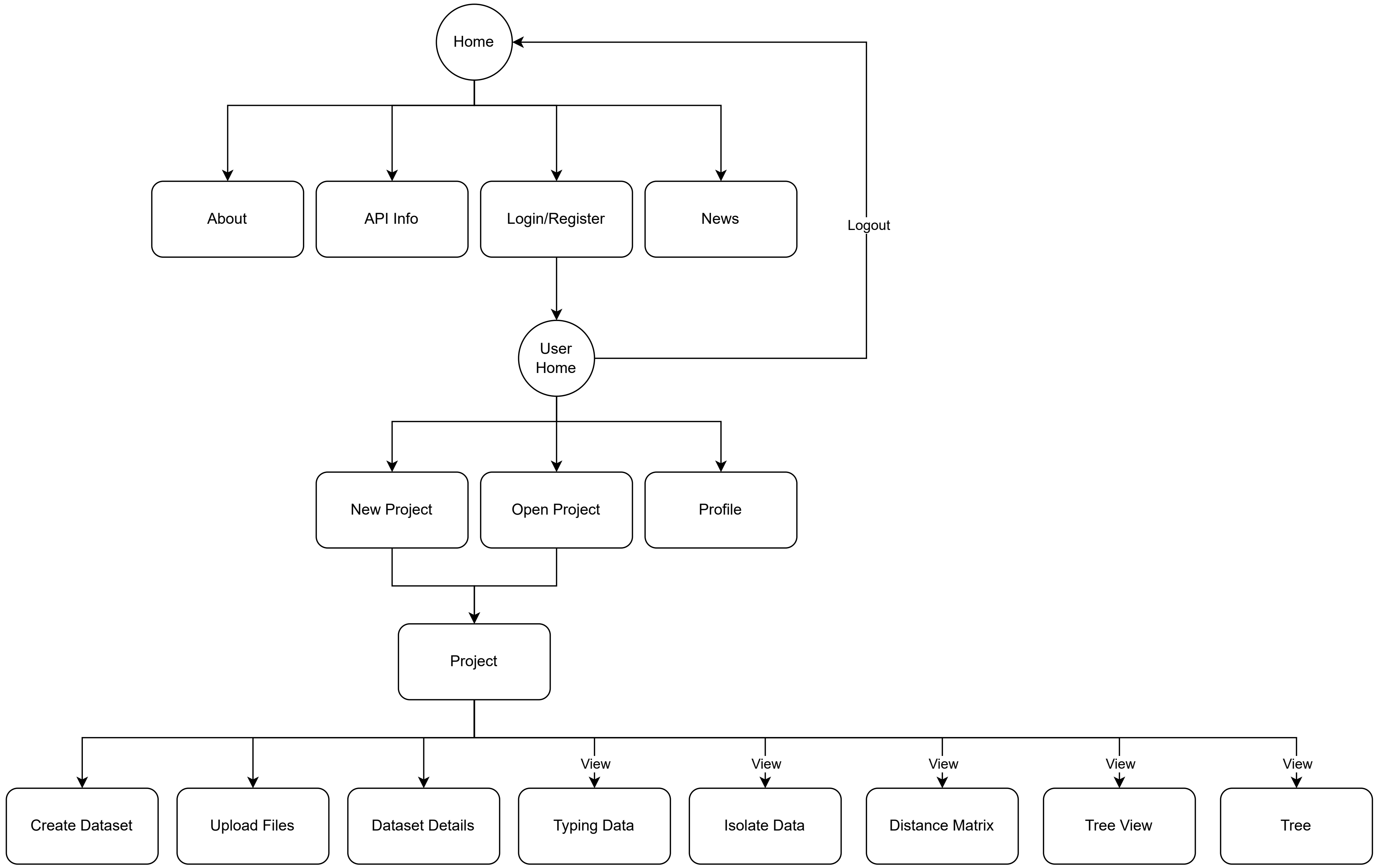


Figure B.1: Navigation diagram.

Appendix C

Example of Tasks of New custom workflow

```
"tasks": [
  {
    "taskId": "downloadDistanceMatrix",
    "tool": "downloader",
    "action": {
      "command": "--project-id=${projectId} --dataset-id=${datasetId} --
      resource-id=${distanceMatrixId} --resource-type=distance-matrix --
      workflow-id=${workflowId} --out=/phyloviz-web-
      platform/distance_matrix.txt"
    },
    "children": [
      "firstTreeCalculation"
    ]
  },
  {
    "taskId": "firstTreeCalculation",
    "tool": "phylolib",
    "action": {
      "command": "algorithm ${algorithm} --matrix=symmetric:/phyloviz-
      web-platform/distance_matrix.txt --out=newick:/phyloviz-web-
      platform/tree1.txt"
    },
    "children": [
      "secondTreeCalculation"
    ]
  },
  {
    "taskId": "secondTreeCalculation",
    "tool": "phylolib",
    "action": {
      "command": "algorithm ${algorithm} --matrix=symmetric:/phyloviz-
      web-platform/distance_matrix.txt --out=newick:/phyloviz-web-
      platform/tree2.txt"
    },
    "children": [
      "firstTreeUpload"
    ]
  },
  {
    "taskId": "firstTreeUpload",
    "tool": "uploader",
    "action": {
      "command": " --file-path=/phyloviz-web-platform/tree1.txt --
      project-id=${projectId} --dataset-id=${datasetId} --workflow-
      id=${workflowId} --resource-type=tree --source-type=algorithm-
      distance-matrix --algorithm=${firstTreeAlgorithm} --distance-
      matrix-id=${distanceMatrixId} --parameters={}"
    },
    "children": [
      "secondTreeUpload"
    ]
  },
  {
    "taskId": "secondTreeUpload",
    "tool": "uploader",
    "action": {
      "command": " --file-path=/phyloviz-web-platform/tree2.txt --
      project-id=${projectId} --dataset-id=${datasetId} --workflow-
      id=${workflowId} --resource-type=tree --source-type=algorithm-
      distance-matrix --algorithm=${secondTreeAlgorithm} --distance-
      matrix-id=${distanceMatrixId} --parameters={}"
    }
  }
]
```