INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

PHYLOViZ Web Platform

A Modular and Web-Based Tool for Phylogenetic Analysis

André Páscoa
André Jesus
Nyckollas Brandão

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

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

May 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

May 2023

Abstract

Modern biomedical research has seen a remarkable increase in the production and computational analysis of large datasets, leading to an urgent need of tools for data integration, data processing and visualization, as well as the need of sharing standardized analytical techniques. One important field of biomedical research is phylogenetic analysis, which allow to understand the evolution of bacterial and viral epidemics, such as SARS-CoV-2 or some E. coli strains, in order to determine their origin, evolution and resistance to the various treatments under study.

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. 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 (desktop and web) are not the same, and the web version is not modular, which makes it difficult to extend and maintain.

We developed PHYLOViZ Web Platform, web-based tool for phylogenetic analysis, with a modular architecture, that not only unifies both versions of PHYLOViZ, but is also able to integrate recently developed prototypes and future ones, being flexible and extensible. This is possible because the platform makes use of several modern open-source technologies, which allow it to be maintained and extended in the future. PHYLOViZ Web Platform 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.

Key Words: PHYLOViZ; phylogenetic analysis; data processing; visualization.

Resumo

A investigação biomédica moderna tem visto 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. Um campo importante da pesquisa biomédica é a análise filogenética, que permite entender a evolução de epidemias bacterianas e virais, como SARS-CoV-2 ou algumas estirpes de E. coli, a fim de determinar sua origem, evolução e resistência aos vários tratamentos em estudo.

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

Desenvolvemos o PHYLOViZ Web Platform, uma ferramenta web para análise filogenética, com uma arquitetura modular, que não só unifica as duas versões do PHYLOViZ, mas também é capaz de integrar protótipos desenvolvidos recentemente e futuros, sendo flexível e extensível. Isto é possível, porque a plataforma faz uso de diversas tecnologias modernas, que permitem a sua manutenção e extensão no futuro. O PHYLOViZ Web Platform permite que os utilizadores acedam e realizem análises e visualizações filogenéticas a parti de qualquer lugar com conexão à Internet, sem a necessidade da instalação de software ou acesso a recursos de computação com alto desempenho. Isso é possível executando a maioria das operações de computação e gestão de dados no lado do servidor.

Palavras Chave: PHYLOViZ; análise filogenética; processamento de dados; visualização.

Acknowledgements

...

Lisbon, May 2023 *André Jesus*

...

Lisbon, May 2023 *André Páscoa*

...

Lisbon, May 2023 *Nyckollas Brandão*

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
eBURST	electronic Based Upon Related Sequence Type
HTTP	Hypertext Transfer Protocol
IAM	Identity and Access Management
IMM	Instituto de Medicina Molecular
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
UML	Unified Modeling Language
UI	User Interface
WMS	Workflow Management System

Chapter 1

Introduction

Phylogenies, or evolutionary trees, are the basic structures necessary to think clearly about differences between species, and to analyze those differences statistically. They have been around for over 140 years, but statistical, computational, and algorithmic work on them is barely 40 years old. A urgent need of tools for data integration, processing and visualization has been felt in modern biomedical research, due to the remarkable increase in the production and computational analysis of large datasets [1]. This need is also felt in the field of phylogenetic analysis, which allows to understand the origin and evolution of bacterial and viral epidemics.

Phylogenetic analysis aims at uncovering the evolutionary relationships between different species or taxa, to obtain an understanding of the evolution of life on Earth. Phylogenetic trees are widely used to address this task and are usually computed from molecular sequences. They also have applications in many other areas. For example, they are used to determine the age and rate of diversification of taxa, to understand the evolutionary history of gene families, in sequence-analysis methods to allow phylogenetic *footprinting*, in epidemiology to trace the origin and transmission of infectious diseases, or to study the co-evolution of hosts and parasites [2].

Numerous 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. PHYLOViZ [3, 4] and PHYLOViZ Online [5], developed by the PHYLOViZ [6] team for the IMM (Instituto de Medicina Molecular), provide phylogenetic analysis processing, namely inference methods, visualization and integration of epidemiological and ancillary data. Nevertheless, features in both PHYLOViZ versions (desktop and web) are not the same, and the web version is not modular, which makes it difficult to extend and maintain. Also, inference and/or visualization optimization tasks are run on the client side, which becomes often unfeasible for large amounts of data as well as data must also be transferred from existing databases in order to be analysed. A prototype, phyloDB [7], was developed for efficiently managing phylogenetic data, supporting, for instance the deployment of algorithms for inferring/detecting patterns and for pre-computing and optimizing phylogenetic visualizations. Other prototype, FLOWViZ [8] was developed to support the usage of a workflow system for processing data.

This project aims to develop a new tool for phylogenetic analysis - PHYLOViZ Web Platform - with a modular architecture [9], that unifies both versions of PHYLOViZ and allows for the use of other tools such as phyloDB and FLOWViZ, not being compromised by these. With its modular architecture, other modules can be used and easily integrated in the future, so that the platform can be extended to support new analysis methods and tools.

The PHYLOViZ Web Platform 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.

The modular architecture allows for easy integration of new analysis methods and tools, making it a flexible and extensible platform for biomedical researchers. For instance, FLOWViZ could be used to enable the creation and execution of customized analysis workflows, while the phyloDB prototype for efficient storage and retrieval of large phylogenetic datasets.

Overall, the development of the PHYLOViZ Web Platform with a modular architecture and advanced data management and optimization capabilities represents a significant step forward in the field of phylogenetic analysis. By providing a user-friendly, web-based platform that can handle large and complex datasets with ease, this new tool will help researchers to better understand the evolution of bacterial and viral epidemics and ultimately contribute to the development of new treatments and preventative measures.

1.1 Outline

This document is composed by eight chapters.

In chapter 2 we will introduce the reader to the domain of phylogenetic analysis, namely the concepts and techniques used in this field. We also present a running example that will be used throughout this work to illustrate the concepts and techniques presented.

In chapter 3 we will describe the domain of our problem in more depth, our approach to solve it, and what other solutions exist.

In chapter 4 we will overview the architecture of PHYLOViZ Web Platform, namely its components and how the goals pointed out in chapter 3 will be reached with this framework.

In chapter 5 we will introduce the technologies that support PHYLOViZ Web Platform.

In chapter 6 we will discuss implementation details about each of the components that compose PHYLOViZ Web Platform, their functionalities and their interactions.

In chapter 7 we show and analyse differences in performance between various implementations, to reinforce why certain decisions were made and features implemented. We will also outline the differences in performance and features between PHYLOViZ Web Platform and its competitors.

In chapter 8 we will present final remarks and do an overview about what we think should be the future of PHYLOViZ Web Platform.

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.

Phylogenetic analyses comprise 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 0100000111101010001000101010101001010100011101011000101010
2 1111010010101100100101010101001000001010010001010101010100
```

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 the 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.

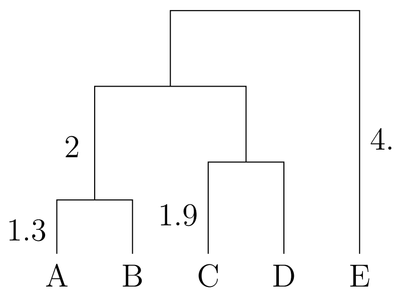


Figure 2.6: Example of a rooted phylogenetic tree. Each node in the tree represents a specific organism or gene, denoted by letters A, B, C, D, and E. The numbers attached to the branches represent the genetic distances between the connected nodes.

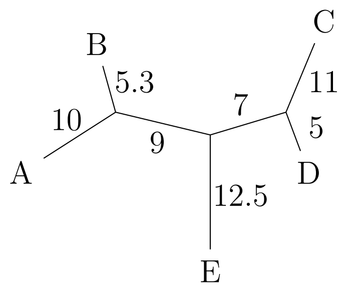


Figure 2.7: Example of an unrooted phylogenetic tree. Unlike a rooted tree, an unrooted tree does not indicate a common ancestor or ancestral lineage. The nodes in the tree correspond to organisms or genes labeled as A, B, C, D, and E. The numbers attached to the branches represent the genetic distances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

Usually phylogenetic analysis includes, the alignment of the DNA sequences, then a distance matrix must be calculated from that alignment, and finally a phylogenetic tree must be constructed from the distance matrix. The tree is generated by using an inference algorithm, which is a method that uses the distance matrix to infer the evolutionary relationships between the species. 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.

As you can see, all pipelines generate a tree view using an inference tree and, optionally, isolate data. In the first pipeline, the inference tree is generated directly from the typing data, using a dynamic distance in the inference algorithm. The second pipeline uses a static distance matrix, which is generated from the typing data, and

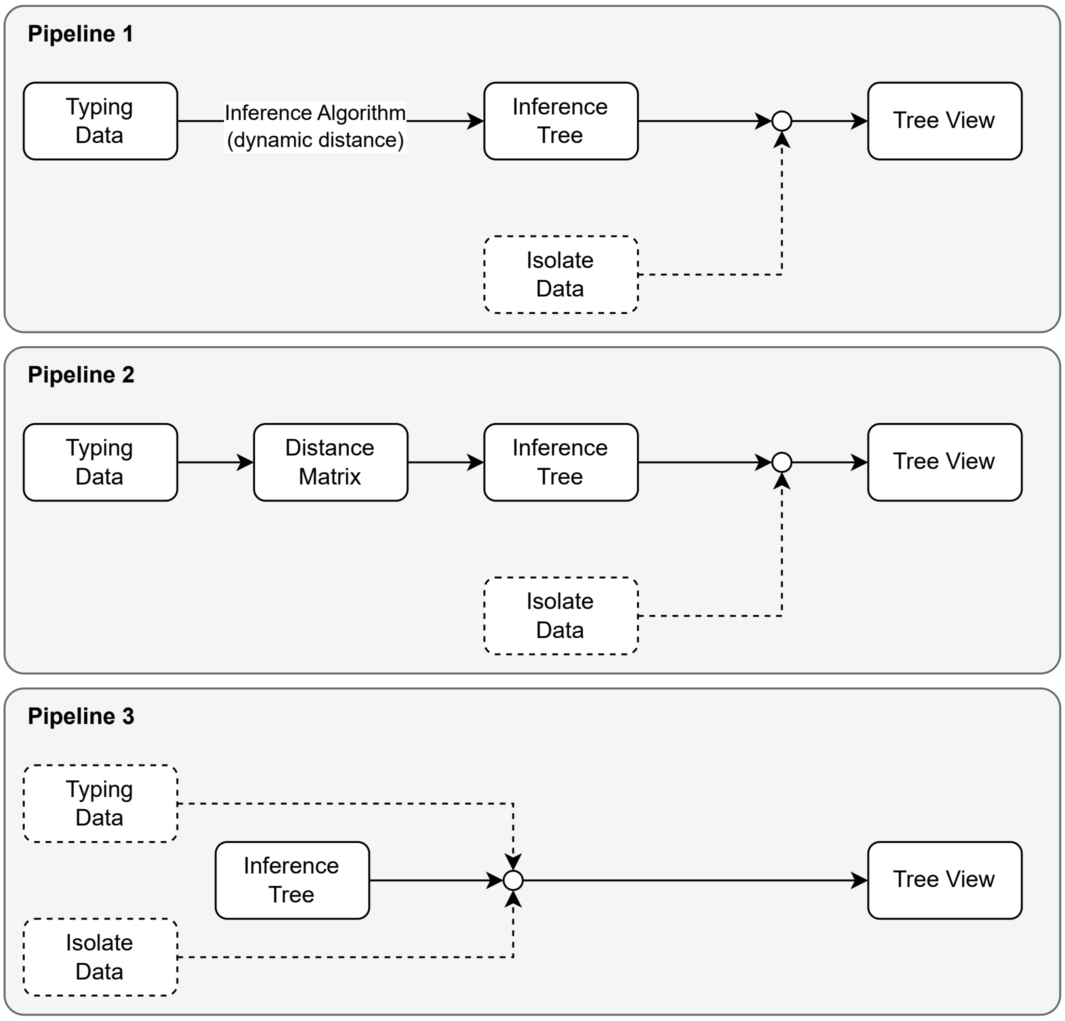


Figure 2.8: Examples of phylogenetic analysis pipelines.

then the inference tree is generated from the distance matrix. The third pipeline uses the inference tree and, optionally, typing and isolate data, to apply transformations to the resulting tree.

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

The pipeline starts with the typing data, which contains the DNA sequences of the species. This data is processed and a Hamming distance matrix is computed. These distances may be subject to a distance correction, that is based on some appropriate model of evolution (also known as substitution model). The resulting distance matrix is then used with a method to reconstruct a phylogenetic tree, using the Neighbor-Joining algorithm. After this, the tree is visualized using the tree generated in the previous step and the isolate data, if it is provided. If an isolate data is provided, an isolate data key must be provided as well.

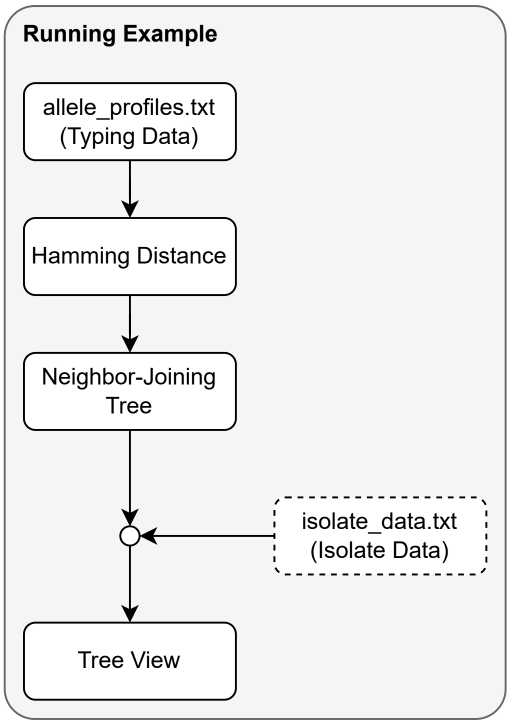

Running Example
allele_profiles.txt (Typing Data)
Hamming Distance
Neighbor-Joining Tree
isolate_data.txt (Isolate Data)
Tree View

Figure 2.9: Pipeline representation of the running example.

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 are used for the computations - various algorithms may yield different results, and thus reveal different aspects and relations of the profiles;
- 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;
- Provide a variety of customization options to the user, including the choice of what algorithms 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 an integration framework 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.

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 problems. Both of them do not allow the choice of the tree visualization layout. Other problem is that this 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, with an anticipated growth in the number of profiles in the future.

The platform should additionally offer enhanced configuration and customization options, specifically in terms of 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;
- The user can create personal projects and datasets;

- The user can upload files, namely typing data and isolate data to initialize the phylogenetic analysis.
- The user can start computations, generating distance matrices and inference trees, and visualize the trees.
- Provide a variety of customization options to the user, including the choice of what algorithms 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 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 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.

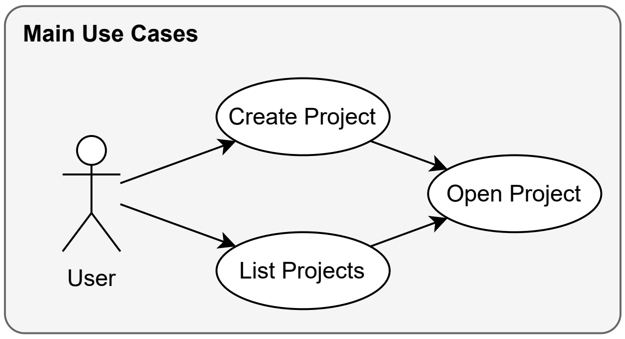


Figure 3.1: Main use cases.

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.

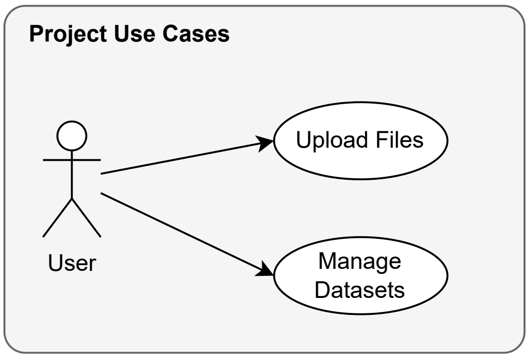


Figure 3.2: Project use cases.

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.

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.

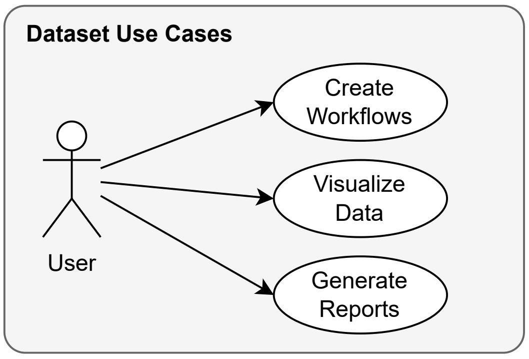


Figure 3.3: Dataset use cases.

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

In this chapter, we will explain the components of the system and how they interact. This chapter will also clarify what the project can accomplish, as well as the architecture, entities, and implementation blueprint that we have designed and developed for it.

4.1 Overview

Figure 4.1 is a diagram that shows the system's main components and their interactions. The system is composed of a backend application (server-side) and a frontend application (client-side).

The backend is composed of a set of microservices [22], which are responsible for the execution of the algorithms and the storage of the data.

The frontend is composed of a web application, which is responsible for the visualization of the data and the interaction with the user. The frontend communicates with the backend through a REST API, using the HTTP protocol [23], which is provided by the API Gateway.

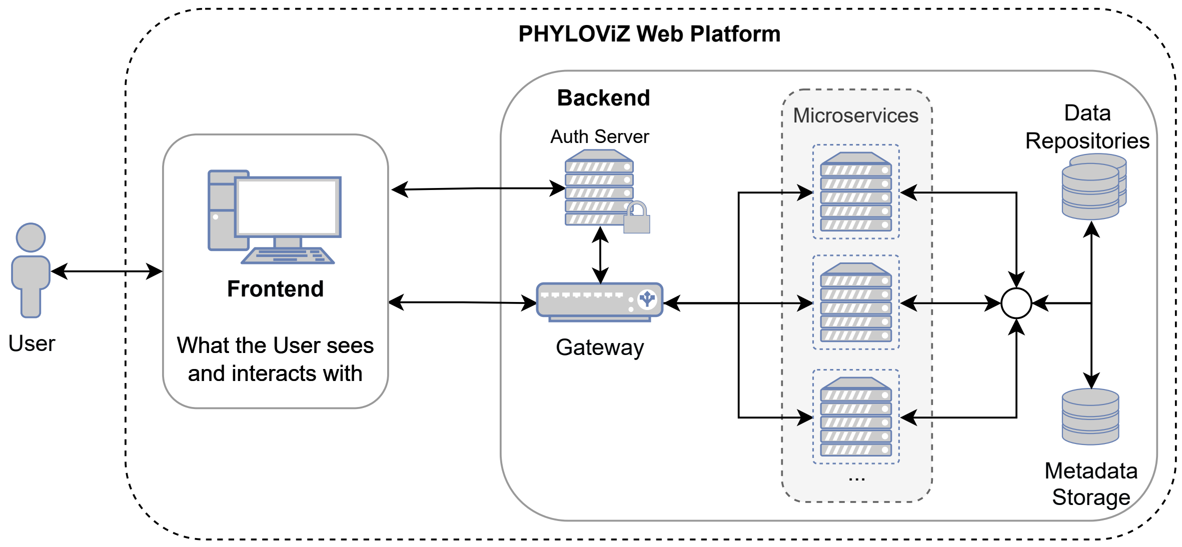


Figure 4.1: System architecture.

4.2 Data Model

The data in our system is categorized into two categories: metadata and phylogenetic data. This is observable in figure 4.1 with Metadata Storage and Data Repositories. Metadata describes the properties and relationships of the data, while data contains the actual data. In this section, we will explain all the entities that make up our data model, and then how they are stored and connected to each other by the metadata-data system.

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 database goes beyond mere descriptive information. It also captures and 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.

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, the metadata database enhances the overall functionality and usability of the system, providing researchers and users with a rich source of information for their analyses and investigations.

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 datasets and files.

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 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

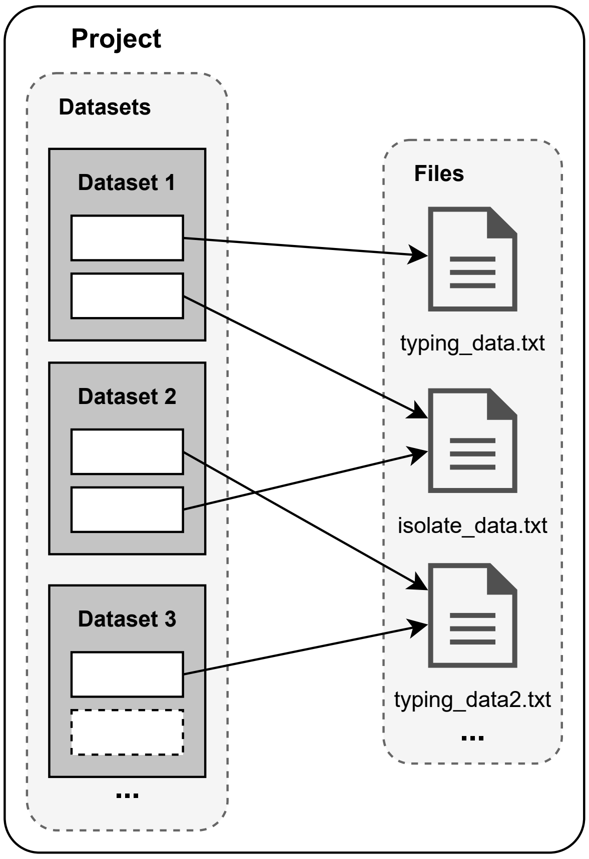


Figure 4.2: Relation between the project, datasets and files. The project has 3 datasets that share files. For example, the dataset 1 and the dataset 2 share the same isolate data file, named "isolate_data.txt".

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 uses a distance matrix generated dynamically.

- **Tree** - result of applying an inference algorithm, based on distances, generating a phylogenetic tree with proper hierarchical relationship 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 chosen, 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.

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.

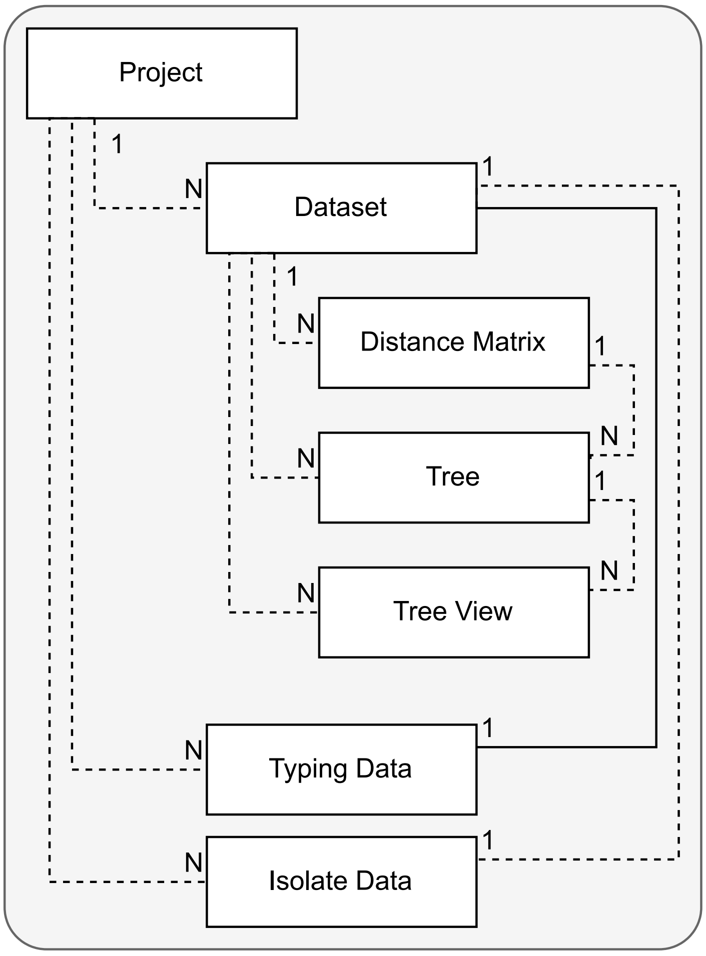


Figure 4.3: Entity overview diagram.

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;

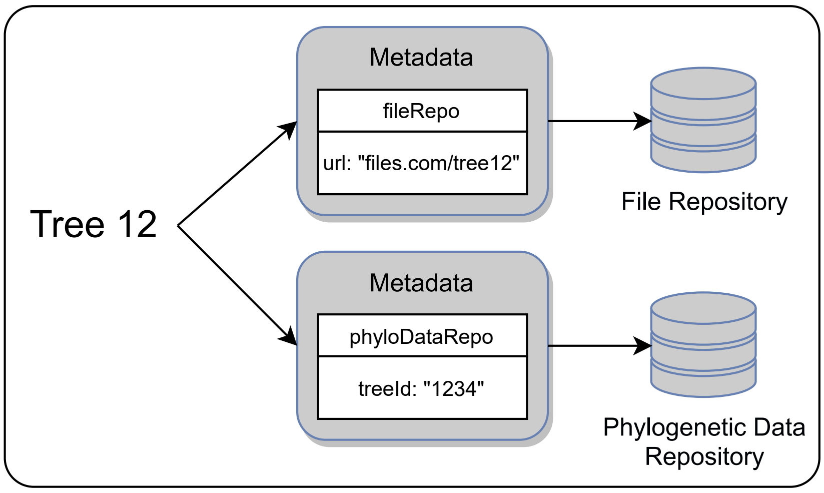


Figure 4.4: Metadata for multiple representations of a tree with the identifier ”12”.

Notice in Figure 4.4 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 ”12”, 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 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.

    - 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.

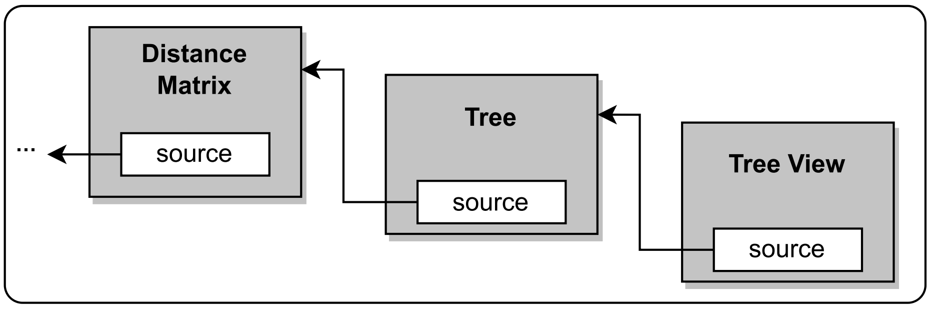
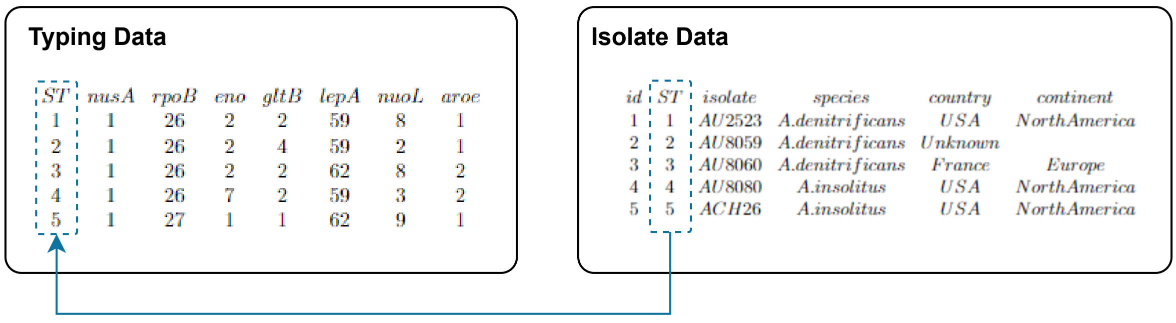


Figure 4.5: Example of source of resources. For example, a distance matrix can be the source of a tree, and subsequently, that tree can become the source of a tree view.

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.



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

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 isolated 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. 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. 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) [24];

- **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 fast 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 and accessible in each designated repository.

Consider a scenario where efficient storage and retrieval of distance matrices are desired. The chosen general phylogenetic data repository may not possess 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.

4.3 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.

The system is composed of the following microservices:

- **Administration microservice**: responsible for the projects and datasets management, including the creation, deletion, and edition of projects and datasets;
- **FileTransfer microservice**: responsible for the upload and download of files of the projects;
- **Compute microservice**: responsible for the execution of the phylogenetic analysis workflows, including the execution of the phylogenetic analysis tools, the alignment of sequences, 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 the phylogenetic tree, the visualization of the distance matrices, and the visualization of the alignment of sequences. Accessing data of the files in table form with pagination is also available here. 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 brief look at each microservice architecture.

4.5.1 Administration Microservice

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

Figure 4.7 is a diagram that shows the architecture of the Administration microservice.

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

The following operations are supported:

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

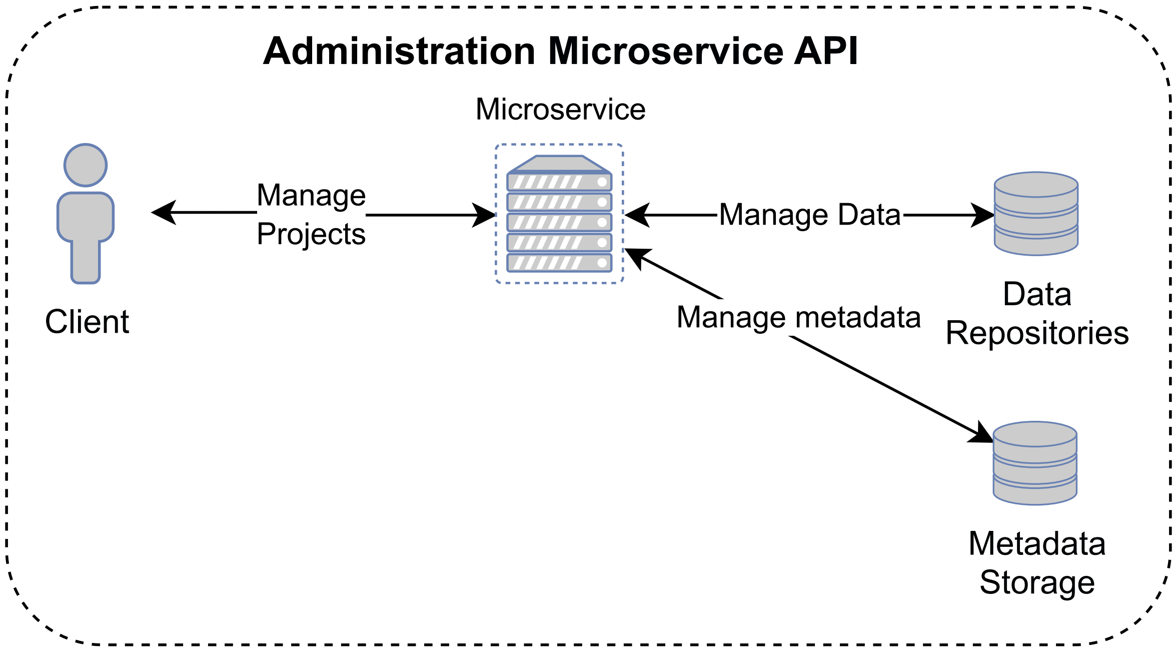


Figure 4.7: Administration microservice architecture.

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 resource is prevented if it is a dependency of another resource, meaning it is marked as a source of another resource.

Deleting a resource results not only in the deletion of its metadata, but also its data from the data repository 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 4.8 is a diagram that shows the architecture of the FileTransfer microservice.

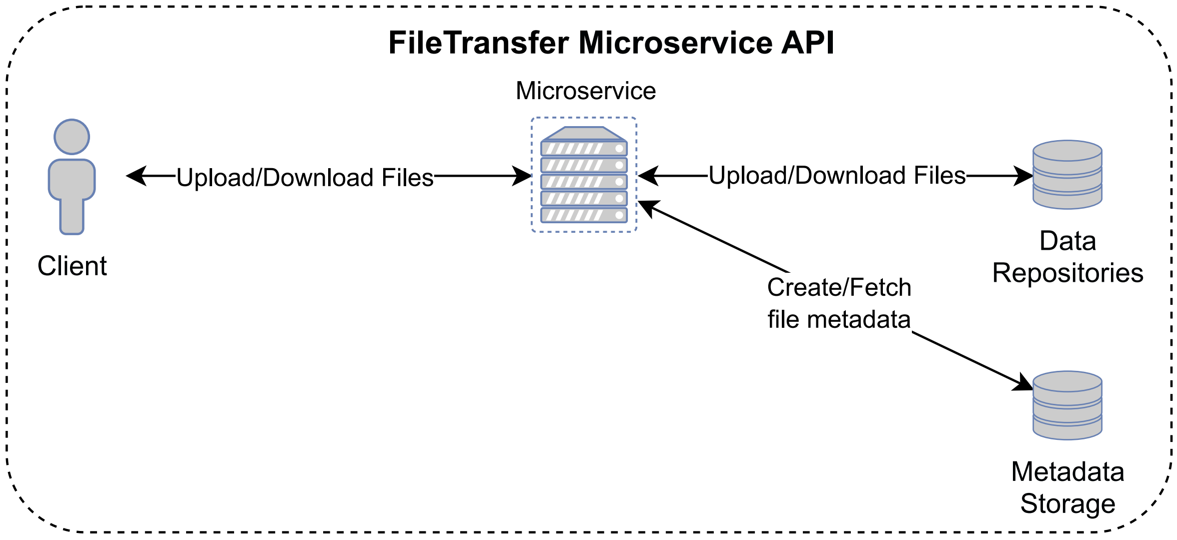


Figure 4.8: FileTransfer microservice architecture.

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.

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 4.9 is a diagram that shows the architecture of the Compute microservice.

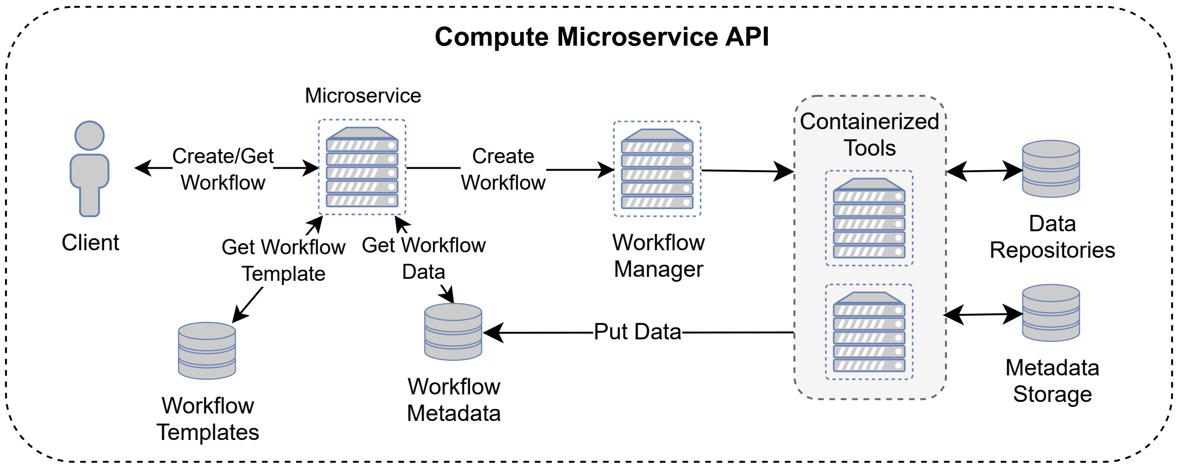


Figure 4.9: Compute microservice architecture.

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 can be defined as a procedure composed by groups of tasks or steps, which are usually intrinsically dependent with each other - an output of a certain task may serve as an input of a future task. This pattern repeats until the final task produce an output - the procedure ends. The procedure's structure and flow resembles a pipeline or a flowchart, being this the reason why these procedures are called pipelines or, as mentioned in this document, *workflows*.

When creating a workflow, the compute microservice first validates the parameters of the request to confirm that the workflow type exists and verify that the parameters have valid values. For example, a parameter of type 'string' should have a set of accepted values. Depending on implementation, this validation may be needed to prevent code injections, in case the parameters are passed directly into the command arguments of the tools to execute.

After validating the parameters, the microservice retrieves the template of the workflow, fills its placeholders with the parameters, and creates the workflow, passing it into the manager. The manager then executes this workflow, initializing the tasks in containerized environments. The tasks will then 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 put data into the workflow instance metadata, by incorporating additional data. 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.

To check the workflow status and its data, the microservice checks the workflow instance metadata for any extra data added by the tasks and communicates with the workflow manager to get the workflow status (RUNNING, SUCCESS, FAILED).

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

The workflows should be easily configurable by the administrator, to allow new tools and workflows to be defined. Therefore, the API for the Compute Microservice should stick to validating the initial parameters and launching the workflows, while the rest of parameter validation and more intrinsic behavior are part of the workflow itself.

The frontend should be synchronized appropriately with changes to the workflows:

- Added a new workflow (added a new workflow template) - implement the operation and/or view to call it.
- Renamed an existing workflow - rename it in the frontend too.
- Changed input parameters of an existing workflow - change parametrization in the views.
- Changed tasks - no need to change anything.

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 4.10 is a diagram that shows the architecture of the Visualization microservice.

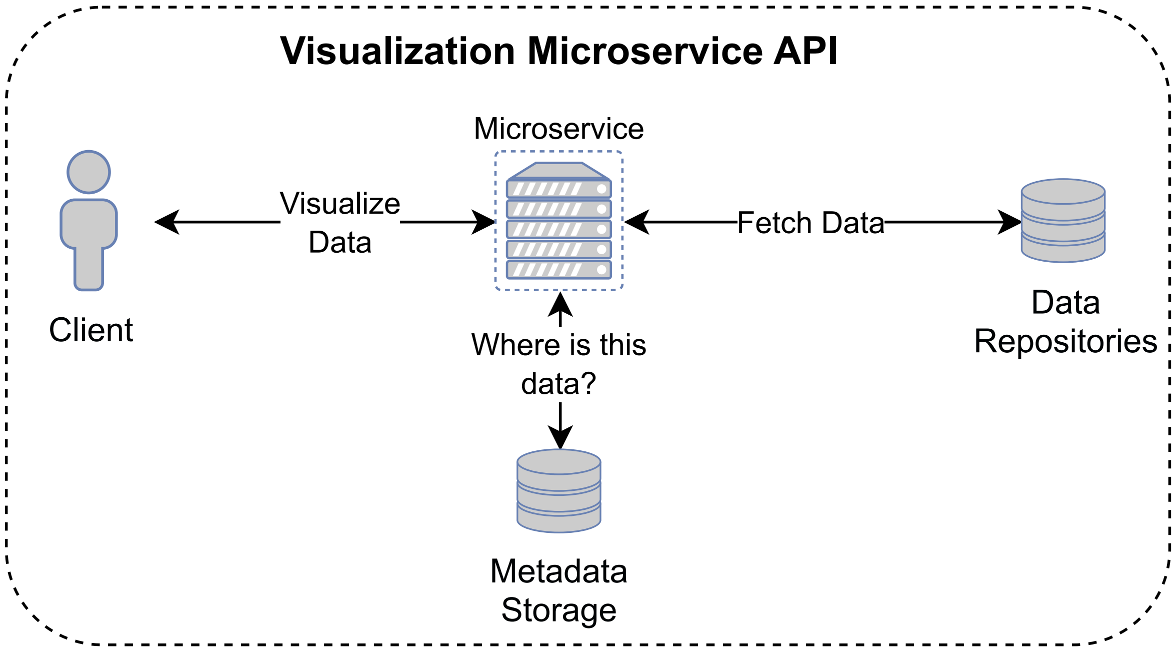


Figure 4.10: Visualization microservice architecture.

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 [25], 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) [26] - Spring, Docker, React, Material-UI, Webpack, GitHub Actions;
- Systems Virtualization Techniques (Técnicas de Virtualização de Sistemas) [27] - Docker;
- Software Laboratory (Laboratório de Software) [28] - Spring, Docker, GitHub Actions;
- Software Development Techniques (Técnicas de Desenvolvimento de Software) [29] - 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 delve 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 [30], 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 [31] 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 [32] 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 [33] 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 [34] 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 [35] 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 [36] 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 [37] 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 [37].

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 and reduce latency [38].

OpenStack

OpenStack [39, 40] 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 [41], 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 [24], 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 [42, 43] 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 instance metadata; 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. 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 [44, 45] 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 [46]. 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 [47] 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 [48] 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 [49] 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 [50] 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 [51].

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 [52, 53] and GitHub [54, 55] 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 [56] 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 [57, 58] 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.

diagrams.net

diagrams.net/draw.io [59] 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 [60] 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.

For abstraction of the fetch call, a function called `apiFetch` was created. This function takes in the request parameters and returns a promise that resolves to the response of the request. It handles sending the request and also handles the response, checking if it is valid or not, and throwing an error if necessary. Additionally, other functions were created to further abstract the `apiFetch` function for each request method used in the application, such as GET, POST, PUT, and DELETE.

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 pass props 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` [61] 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` [62] 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.

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 used is cosmos.js, which 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 libraries, 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.

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.

By combining these components and leveraging the capabilities of the `chart.js` library and `cosmos.js`, the `TreeView` component provides an interactive and customizable tree visualization experience for users within the application.

6.1.3 User Interface and Navigation

...

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 the following subsections we describe in more detail each microservice of the system.

Each microservice is implemented using Spring Boot, which is a framework that makes it easy to create stand-alone, production-grade Spring based applications.

6.3.1 Common Implementation

...

6.3.2 Administration Microservice

The Administration microservice is responsible for managing the projects of the system.

...

6.3.3 FileTransfer Microservice

The FileTransfer microservice is responsible for uploading and downloading files to and from the system.

...

6.3.4 Compute Microservice

The Compute microservice is responsible for executing the computation tasks of the system, such as calculation of distance matrices, phylogenetic trees and visualizations.

...

Workflows

...

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.

...

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 [35].

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. Open the `application.yml` file in `shared` folder and the gateway;
2. Replace the client-secret property with the "Secret" from your clipboard.

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

6.5 Deployment

...

Chapter 7

Testing and Evaluation

In this chapter we show and analyse differences in performance between various implementations, to reinforce why certain decisions were made and features implemented. We will also outline the differences in performance and features between PHYLOViZ Web Platform and its competitors.

7.1 Performance Testing

...

7.2 Discussion

Chapter 8

Final Remarks

...

8.1 Conclusions

...

8.2 Future Work

There are several possible continuations of the work done throughout this project. One could be ... . Other potential development could be ....

References

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

[2] D. H. Huson, R. Rupp, and C. Scornavacca, *Phylogenetic networks: concepts, algorithms and applications.* Cambridge University Press, 2010.

[3] A. P. Francisco, C. Vaz, P. T. Monteiro, J. Melo-Cristino, M. Ramirez, and J. A. Carriço, "Phyloviz: phylogenetic inference and data visualization for sequence based typing methods," *BMC bioinformatics*, vol. 13, pp. 1–10, 2012.

[4] M. Nascimento, A. Sousa, M. Ramirez, A. P. Francisco, J. A. Carriço, and C. Vaz, "Phyloviz 2.0: providing scalable data integration and visualization for multiple phylogenetic inference methods," *Bioinformatics*, vol. 33, no. 1, pp. 128–129, 2017.

[5] B. Ribeiro-Gonçalves, A. P. Francisco, C. Vaz, M. Ramirez, and J. A. Carriço, "Phyloviz online: web-based tool for visualization, phylogenetic inference, analysis and sharing of minimum spanning trees," *Nucleic acids research*, vol. 44, no. W1, pp. W246–W251, 2016.

[6] A. Francisco, C. Vaz, P. Monteiro, J. Melo-Cristino, M. Ramirez, and J. A. Carriço, "Phyloviz," www.phyloviz.net, 2014. [Online]. Available: https://www.phyloviz.net/

[7] B. Lourenço, "A framework for large scale phylogenetic analysis," *arXiv:2012.13363 [q-bio]*, 01 2021. [Online]. Available: https://arxiv.org/abs/2012.13363

[8] M. Luis and C. Vaz, "Flowviz: Framework for phylogenetic processing," *arXiv:2211.15282 [cs]*, 11 2022. [Online]. Available: https://arxiv.org/abs/2211.15282

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

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

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

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

[13] N. I. of Health *et al.*, "What are single nucleotide polymorphisms (snps)," *Genetics Home Reference-NIH. US National Library ofMedicine. URL: https://ghr. nlm. nih. gov/primer/genomicresearch/snp*, 2019.

[14] C. Vaz, M. Nascimento, J. A. Carriço, T. Rocher, and A. P. Francisco, "Distance-based phylogenetic inference from typing data: a unifying view," *Briefings in Bioinformatics*, vol. 22, no. 3, p. bbaa147, 2021.

[15] A. P. Francisco, M. Bugalho, M. Ramirez, and J. A. Carriço, "Global optimal eburst analysis of multilocus typing data using a graphic matroid approach," *BMC bioinformatics*, vol. 10, pp. 1–15, 2009.

[16] B. Ribeiro-Gonçalves, A. P. Francisco, C. Vaz, M. Ramirez, and J. A. Carriço, "Phyloviz online," online.phyloviz.net. [Online]. Available: https://online.phyloviz.net/

[17] L. Silva, "Library of efficient algorithms for phylogenetic analysis," *arXiv:2012.12697 [cs]*, 12 2020. [Online]. Available: https://arxiv.org/abs/2012.12697

[18] D. H. Huson and D. Bryant, "Application of Phylogenetic Networks in Evolutionary Studies," *Molecular Biology and Evolution*, vol. 23, no. 2, pp. 254–267, 10 2005. [Online]. Available: https://doi.org/10.1093/molbev/msj030

[19] A. Dereeper, V. Guignon, G. Blanc, S. Audic, S. Buffet, F. Chevenet, J.-F. Dufayard, S. Guindon, V. Lefort, M. Lescot, J.-M. Claverie, and O. Gascuel, "Phylogeny.fr: robust phylogenetic analysis for the non-specialist," *Nucleic Acids Res*, vol. 36, no. Web Server issue, pp. W465–9, Apr. 2008.

[20] F. Lemoine, D. Correia, V. Lefort, O. Doppelt-Azeroual, F. Mareuil, S. Cohen-Boulakia, and O. Gascuel, "NGPhylogeny.fr: new generation phylogenetic services for non-specialists," *Nucleic Acids Research*, vol. 47, no. W1, pp. W260–W265, 04 2019. [Online]. Available: https://doi.org/10.1093/nar/gkz303

[21] Z. Zhou, N.-F. Alikhan, M. J. Sergeant, N. Luhmann, C. Vaz, A. P. Francisco, J. A. Carriço, and M. Achtman, "GrapeTree: visualization of core genomic relationships

among 100,000 bacterial pathogens," *Genome Res*, vol. 28, no. 9, pp. 1395–1404, Jul. 2018.

[22] S. Li, H. Zhang, Z. Jia, C. Zhong, C. Zhang, Z. Shan, J. Shen, and M. A. Babar, "Understanding and addressing quality attributes of microservices architecture: A systematic literature review," *Information and software technology*, vol. 131, p. 106449, 2021.

[23] R. Fielding and J. Reschke, "Hypertext transfer protocol (http/1.1): Semantics and content," Internet Engineering Task Force (IETF), Tech. Rep., 2014.

[24] AWS, "Cloud object storage — store & retrieve data anywhere — amazon simple storage service," Amazon Web Services, Inc., 2018. [Online]. Available: https://aws.amazon.com/s3/

[25] D. Hardt, "The oauth 2.0 authorization framework," Internet Engineering Task Force (IETF), Tech. Rep., 2012.

[26] "Web application development — instituto superior de engenharia de lisboa," www.isel.pt. [Online]. Available: https://www.isel.pt/en/leic/web-application-development

[27] "Systems virtualization techniques — instituto superior de engenharia de lisboa," www.isel.pt. [Online]. Available: https://www.isel.pt/en/leic/systems-virtualization-techniques

[28] "Software laboratory — instituto superior de engenharia de lisboa," www.isel.pt. [Online]. Available: https://www.isel.pt/en/leic/software-laboratory

[29] "Software development techniques — instituto superior de engenharia de lisboa," www.isel.pt. [Online]. Available: https://www.isel.pt/en/leic/software-development-techniques

[30] K. Arnold, J. Gosling, and D. Holmes, *The Java programming language.* Addison Wesley Professional, 2005.

[31] S. Samuel and S. Bocutiu, *Programming kotlin.* Packt Publishing Ltd, 2017.

[32] C. Walls, *Spring in Action, Sixth Edition.* Simon and Schuster, 04 2022.

[33] ——, *Spring Boot in action.* Simon and Schuster, 2015.

[34] J. Carnell and I. H. Sánchez, *Spring microservices in action.* Simon and Schuster, 2021.

[35] S. Thorgersen and P. I. 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.

[36] D. Recordon and D. 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.

[37] J. Nickoloff and S. Kuenzli, *Docker in Action.* Simon and Schuster, 10 2019.

[38] "Deploy a registry server — docker," Docker Documentation. [Online]. Available: https://docs.docker.com/registry/deploying/

[39] O. Sefraoui, M. Aissaoui, M. Eleuldj *et al.*, "Openstack: toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, pp. 38–42, 2012.

[40] openstack, "Build the future of open infrastructure." OpenStack, 2010. [Online]. Available: https://www.openstack.org/

[41] "Home — biodata.pt," biodata.pt. [Online]. Available: https://biodata.pt/

[42] K. Banker, D. Garrett, P. Bakkum, and S. Verch, *MongoDB in action: covers MongoDB version 3.0.* Simon and Schuster, 2016.

[43] MongoDB, "The most popular database for modern apps," MongoDB. [Online]. Available: https://www.mongodb.com/

[44] J. Carlson, *Redis in action.* Simon and Schuster, 2013.

[45] Redis, "Redis," redis.io. [Online]. Available: https://redis.io/

[46] G. Bierman, M. Abadi, and M. 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.

[47] M. Thomas, *React in Action.* Simon and Schuster, 05 2018.

[48] MUI, "Mui: The react component library you always wanted," mui.com. [Online]. Available: https://mui.com/

[49] "webpack," webpack. [Online]. Available: https://webpack.js.org/

[50] N. Rokotyan, O. Stukova, and D. Ovsyannikov, "Cosmograph: GPU-accelerated Force Graph Layout and Rendering," Jun. 2022. [Online]. Available: https://github.com/cosmograph-org/cosmos

[51] “Cosmos — comparison with other network graph visualization libraries,” www.youtube.com. [Online]. Available: https://www.youtube.com/watch?v=HWk78hP8aEE

[52] S. Chacon and B. Straub, *Pro git.* Springer Nature, 2014.

[53] Git, “Git,” Git-scm.com. [Online]. Available: https://git-scm.com/

[54] L. Dabbish, C. Stuart, J. Tsay, and J. 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.

[55] GitHub, “Github,” GitHub, Inc. [Online]. Available: https://github.com/

[56] “Planning and tracking with projects,” GitHub Docs. [Online]. Available: https://docs.github.com/en/issues/planning-and-tracking-with-projects

[57] C. Chandrasekara, P. Herath, C. Chandrasekara, and P. Herath, “Introduction to github actions,” *Hands-on GitHub Actions: Implement CI/CD with GitHub Action Workflows for Your Applications*, pp. 1–8, 2021.

[58] “Features - github actions,” GitHub, Inc., 2018. [Online]. Available: https://github.com/features/actions

[59] “Diagram software and flowchart maker,” www.diagrams.net. [Online]. Available: https://www.diagrams.net/

[60] “Fetch api - web apis — mdn,” developer.mozilla.org. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

[61] “Tree view react component - material ui,” mui.com. [Online]. Available: https://mui.com/material-ui/react-tree-view/

[62] “React data grid component - mui x,” mui.com. [Online]. Available: https://mui.com/x/react-data-grid/

Appendices

Appendix A

Authentication Sequence

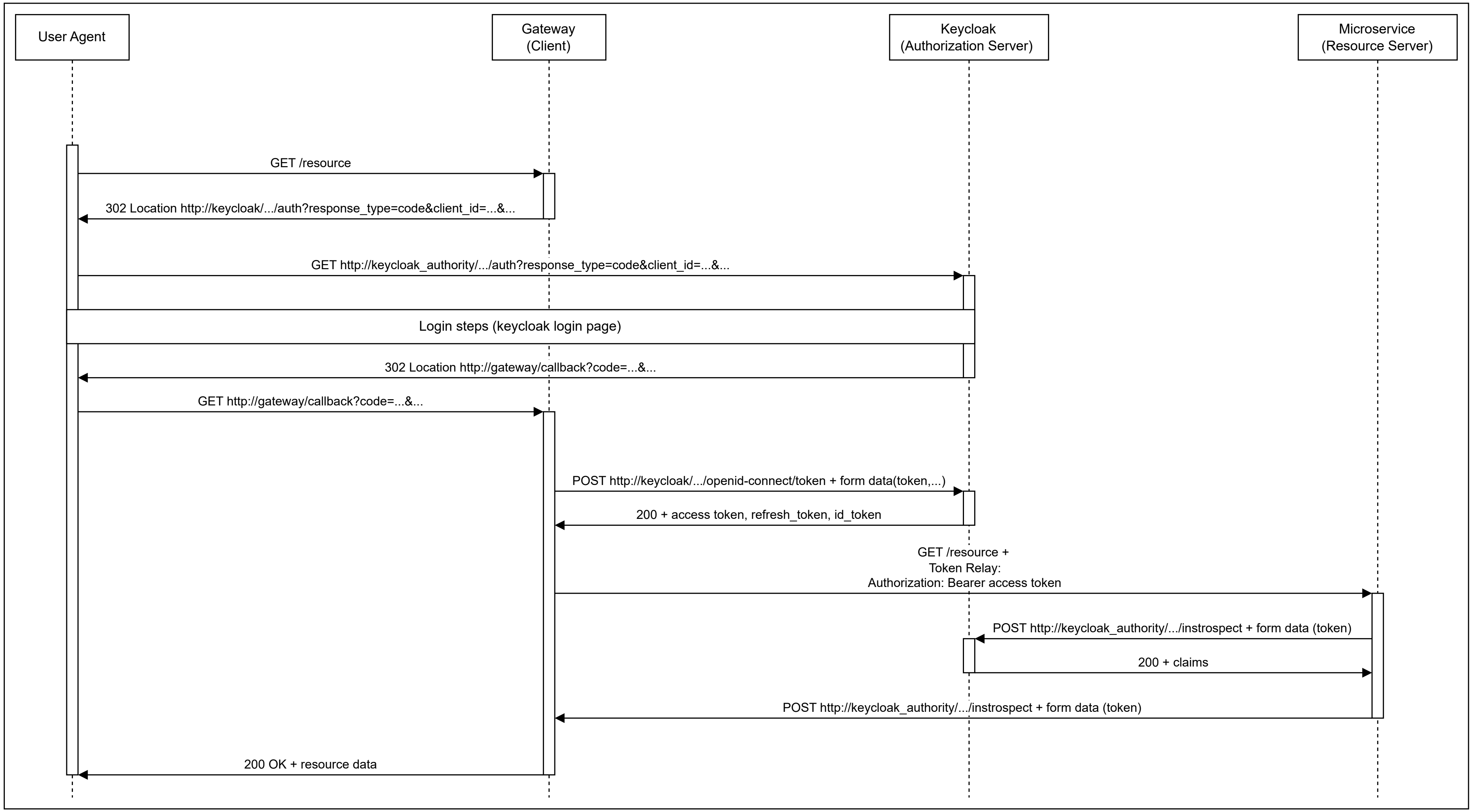

User Agent
Gateway (Client)
Keycloak (Authorization Server)
Microservice (Resource Server)
GET /resource
302 Location http://keycloak/.../auth?response_type=code&client_id=...&...
GET http://keycloak_authority/.../auth?response_type=code&client_id=...&...
Login steps (keycloak login page)
302 Location 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.