Universidade do Minho
Escola de Engenharia
Departamento de Informática

José Henrique Lopes Pereira

Development of MOSGUITO: a user-friendly graphical interface for meta-omics data analyses

October, 2022

Universidade do Minho
Escola de Engenharia
Departamento de Informática

José Henrique Lopes Pereira

Development of MOSGUITO: a user-friendly graphical interface for meta-omics data analyses

Master dissertation
Master Degree in Bioinformatics

Dissertation supervised by
Andreia Filipa Ferreira Salvador
Vítor Manuel Sá Pereira

October, 2022

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

AGRADECIMENTOS

Ao fazer esta tese, pude contar com o apoio de várias pessoas sem as quais este trabalho não teria chegado ao ponto em que chegou. Este é o meu obrigado a quem me apoiou e ajudou. À Andreia, pois sem uma boa orientadora fica bastante complicado realizar e concretizar uma boa tese, sendo que sem a sua ajuda e presistêcia nada disto era possível. Ao professor Vítor, mentor disponível para todos os momentos em que se requeria a sua participação e ajuda. Ao João, que foi das pessoas que mais me acompanhou e ajudou a desbloquear quando certas complicações apareciam. Aos meus colegas e amigos que me apoiaram e ajudaram sempre que precisei, estando sempre disponíveis para me ajudar e aturar em qualquer situação. Um grande obrigado à minha família principalmente aos meus pais que de tudo fizeram para apoiar o seu filho a ultrapassar mais uma etapa da sua vida. Por fim agradeço a Deus por todo o apoio e ajuda incondicional.

STATEMENT OF INTEGRITY

ABSTRACT

Complex microbial communities are essential to all ecosystems, and by linking microbial identity to function, meta-omics technologies facilitate the interpretation of the processes catalyzed by microorganisms. MOSCA is a command-line pipeline that performs bioinformatics analyses of metagenomics, metatranscriptomics, and metaproteomics. MOSGUITO is a web-based tool developed in React, which allows the configuration of MOSCA's workflow and the visualization of MOSCA outputs. Although the metadata and the configuration options of MOSCA could be easily customized and downloaded through MOSGUITO, MOSGUITO was unable to interact with MOSCA automatically. In this thesis, a third-tier client-server architecture was developed containing the Client MOSGUITO, the Server MOSCA, and a Database. MOSGUITO as a client-side can retrieve, store and delete data from the Database and start running analysis on MOSCA as a server. MOSCA as a server can receive files from the client-side and start an analysis run. The database can store results from MOSCA, input files from users, and respective user information from their login session. A full guide to how to utilize this new version of MOSGUITO is provided. MOSGUITO client-side can interact with MOSCA as a server using Flask APIs, end users don't need to have knowledge on command-line pipelines to use MOSCA, nor the computer resources to download it. Therefore users using MOSGUITO can optimize the usage and configuration of MOSCA, being able to analyze the data from omics experiments with a simple interaction with MOSGUITO.

Keywords: Meta-omics, MOSCA, Flask, MOSGUITO, API

RESUMO

Comunidades microbianas complexas são essenciais em todos os ecossistemas, as tecnologias metaómicas facilitam a interpretação dos processos catalisados pelos microrganismos, pois permitem ligar a identidade dos microrganismos à sua função. MOSCA é um pipeline que funciona à base de linha de comandos que realiza análises de bioinformática de metagenómica, metatranscriptómica e metaproteómica. O MOSGUITO é uma ferramenta web desenvolvida em React, que permite a configuração do fluxo de trabalho do MOSCA e a visualização dos resultados. Embora os meta dados e as opções de configuração do MOSCA possam ser facilmente personalizados e transferidas através do MOSGUITO, o MOSGUITO não conseguia interagir com o MOSCA automaticamente. Nesta tese, foi desenvolvida uma arquitetura cliente-servidor de terceiro nível contendo o Cliente MOSGUITO, o Servidor MOSCA e uma Base de Dados. O MOSGUITO como cliente pode recuperar, armazenar e excluir dados da base de dados e começar a executar análises no MOSCA como servidor. O MOSCA como servidor pode receber arquivos do lado do cliente e iniciar uma execução de análise. A base de dados pode armazenar resultados do MOSCA, ficheiros de input submetidos pelos utilizadores e respetivas informações da sessão de Login do utilizador. É apresentado um guia completo de como utilizar esta nova versão do MOSGUITO. O lado do cliente MOSGUITO pode interagir com o MOSCA como um servidor usando APIs construídas utilizando a framework Flask. Os Utilizadores finais não precisam ter conhecimento sobre linhas de comando para usar o MOSCA e sem a necessidade de recursos de computador para o transferir. Assim, os utilizadores do MOSGUITO otimizam o uso e a configuração do MOSCA, podendo analisar seus dados com uma simples interação com o MOSGUITO.

Palavras-Chave: Metaómica, MOSCA, Flask, MOSGUITO, API

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

API Aplication Programming Interface.

CLI command-line interface.
CRUD CRUD (Create, Read, Update and Delete).

dbms Database Management System.

MG metagenomics.
MOSCA Meta-Omics Software for Community Analysis.
MOSGUITO MOSca's Graphical User Interface (GUI) TO perform meta-omics analyses.
MP metaproteomics.
MT metatranscriptomics.

OAuth OAuthentication.

PSMs peptide-spectrum matches.

QC Quality Control.

URI Uniform Resource Identifier.
URL Uniform Resource Locator.

ACRONYMS

API	Aplication Programming Interface.
CLI	command-line interface.
CRUD	CRUD (Create, Read, Update and Delete).
dbms	Database Management System.
MG	metagenomics.
MOSCA	Meta-Omics Software for Community Analysis.
MOSGUITO	MOSca's Graphical User Interface (GUI) TO perform meta-omics analyses.
MP	metaproteomics.
MT	metatranscriptomics.
OAuth	OAuthentication.
PSMs	peptide-spectrum matches.
QC	Quality Control.
URI	Uniform Resource Identifier.
URL	Uniform Resource Locator.

1 INTRODUCTION

1.1 CONTEXT AND MOTIVATION

Complex microbial communities are essential to all ecosystems, and by linking microbial identity to function, omics technologies facilitate the interpretation of the processes catalyzed by microorganisms. Analysis of meta-omics data can be very challenging, and for that purpose, several bioinformatics pipelines have been developed. MOSCA is a command-line bioinformatics pipeline that performs MG, MT, and MP integrated analysis. A web interface for MOSCA will enable its utilization by users less familiarized with command-line interfaces. MOSCA workflow includes computationally intensive steps such as assembly and annotation, which demand a lot of computational resources such as Random access memory (RAM) and Central Process Unit (CPU). Integrating MOSCA as a web service alleviates the need for expensive resources dedicated to meta-omics analysis. The first version of MOSGUITO (available at https://iquasere.github. io/MOSGUITO) is a ReactJS application developed to build's configuration files and to facilitate the analysis of MOSCA results. The MOSCA results are presented in an organized structure, allowing users to interact with them. MOSGUITO helps in the configuration step by producing a file that is used for the customization of the MOSCA workflow, but the upload of MOSCA results to MOSGUITO remains limited. This could be overcome by implementing a web service with MOSGUITO as an interface and MOSCA running in the background. With that, users would not need to download the MOSCA pipeline and would interact only with MOSGUITO. Another improvement could be the separation of MOSCA into various stages of analyses, allowing to select the analysis step to perform without using a command prompt. Also very important would be the control of the access to the web service by using user accounts. The account could be used from a third-party organization using OAuthentication (OAuth), and consequently, the user would not need to create a new account and could be notified by email when the data analyses is complete.

1.2 OBJECTIVES

The objective of this thesis is to develop a web version of MOSCA. The specific objectives are the following:

1. Create a database that allows to organize data from various users.
2. Create a graphical user interface to visualize pertinent information such as inputs, running analysis status, and outputs.

To achieve these objectives, the following tasks were implemented:

1. Divide and organize MOSCA's steps in order users to perform different steps in the MOSCA analyses pipeline transforming it into a Micro-service Workflow.
2. Connect all the different containers in a Docker workflow Manager to manage all requests from the web service and utilize all the resources available for the respective requests.
3. Implement an API to manage all user requests; interact with docker workflow manager and database. The API will retrieve information from the database, namely the results files from the analyses, inputs, parameters used to configure the analyses, and state of all steps involved in the analyses.
4. Develop a Database to store and manage all users and their respective projects and files.
5. Develop a Login for the users. Users can log in using third-party accounts utilizing OAuth.Users Log-in with Google will allow to save the results in their Google Drive.
6. Integrate the web-server with front-end MOSGUITO, to directly link execution argument inputs and presentation of results.
7. Integrate this implementation in a server.

1.3 THESIS ORGANIZATION

This thesis presents the construction of MOSGUITO/MOSCA as a unified fully automated tool, providing the capabilities of the MOSCA pipeline through the graphical interface of MOSGUITO. In chapter two, a survey of the state-of-the-art is made regarding the general steps involved in the analysis of MG, MT, and MP data. A survey of web tools for omics analyses is also presented. Also included is an overview of different types of client-server applications and methods, giving a small introduction to their principle compositions like server side, client-side, the use of a Database and different types of user authentication. There is also an approach to web services the different types and an annotation of the Rest type and the different frameworks to develop them. Chapter 3 explains the architecture of MOSGUITO/MOSCA the steps involved in the construction of the application and the most important packages that were used in the development of MOSCA's web service. Chapter 4 offers a complete guide on how to work with the application. In Chapter 5 a discussion is present. Chapter 6 consists of the thesis's main conclusions, its limitations, and its future prospects.

2

STATE OF THE ART

2.1 META-OMICS ANALYSIS

2.1.1 *Metagenomics*

MG is the study of the genomes of the organisms present in an environmental sample. MG gives a general information on potentially novel biocatalysts or enzymes, genomic linkages between functions and phylogeny for uncultured organisms, and evolutionary profiles of community function and structure (Sleator et al., 2008). MG can be integrated with MT or MP approaches to describe the functional activity, i.e., levels of expression for genes of interest (Gilbert et al., 2008; Béja et al., 2000). MG pipelines usually contain the following steps: Preprocessing consisting mainly on removing adapters and spurious /erroneous reads; Assembly consisting on the alignment of reads into contigs that represent the sequences present in the original sample as closely as possible; Binning that is the process of sorting DNA sequences into groups that might represent an individual genome or genomes from closed related organisms; Annotation that identifies the predicted genes present in the study sample; and Quantification, that measures the relative abundance of organisms and genes (Carvalho and Irizarry, 2010). These steps are included in MOSCAs MP workflow (Fig.1). The MOSCA process for MG data will be described here:

Preprocessing in MOSCA receives reads (FastQ) as input. FastQC (Brown et al., 2017) is used for Quality Control (QC) of the reads, and information from the reports of FastQC are used to set the parameters of Trimmomatic (Bolger et al., 2014), for removal of adapters and spurious/erroneous reads. This phase contains one visible output, the FastQC reports, and prepossessed reads (FastQ) that will be used in assembly.

The Assembly step takes as input high-quality reads and uses de novo assemblers, MetaSPAdes (Nurk et al., 2017) or Megahit (Li et al., 2016), to assemble the reads into contigs (in FASTA format). The quality control of the assembly is performed by MetaQUAST (Mikheenko et al., 2016) and Bowtie2 (Langmead et al., 2009) which produce quality reports of the assembly in TSV format. Assembly is an optional step of MOSCA's workflow.

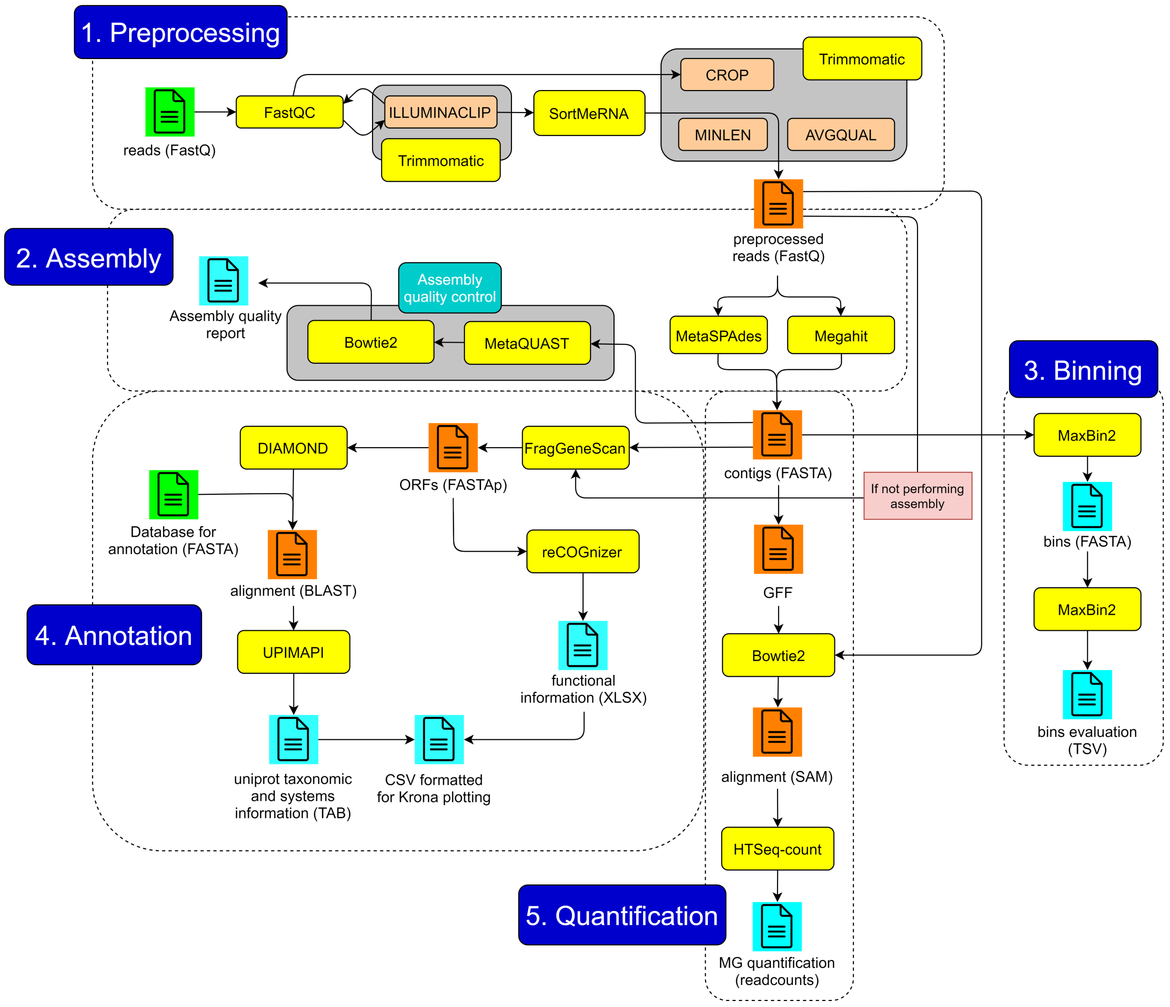


Figure 1: MOSCA MG architecture workflow. Blue icons: the main steps of the workflow; Yellow icons: the bioinformatics tools utilized in each step; Green icons: the inputs needed to perform the analysis; Orange icons: the intermediate files i.e., files outputted by one step that serve as input to another step; and Navy Blue icons: the output files.

Binning takes as input the contigs obtained from the assembly and sorts their sequences into groups that might represent an individual genome or genomes from closely related organisms. MaxBin2 (Wu et al., 2016) is used to build bins (in FASTA format) and bins evaluation with CheckM obtains reports in TSV format. If assembly is not executed, binning is not either.

Annotation of metagenomes takes as input either the contigs constructed in the assembly step or the preprocessed reads. Before Annotation, the gene calling takes place by using FragGeneScan (Rho et al., 2010), which provides protein sequences (in FASTA format) for homology-based annotation with UPIMAPI and domain-based annotation with reCOGnizer. Results from annotation are organized into TSV and EXCEL tables and Krona plots.

MG **Quantification** is the last stage of annotation in MOSCA. It outputs readcounts for each gene.

2.1.2 *Metatranscriptomics*

MT analysis enables an understanding of how the microbiome responds to the environment by studying the functional analysis of genes expressed by the microbiome. It can also estimate the microbial population's taxonomic composition. MT projects workflows are divided into the main steps: Preprocessing, Functional Annotation, Quantification, and Differential Expression Analysis. MOSCA includes these steps into its MT workflow (Fig. 2).

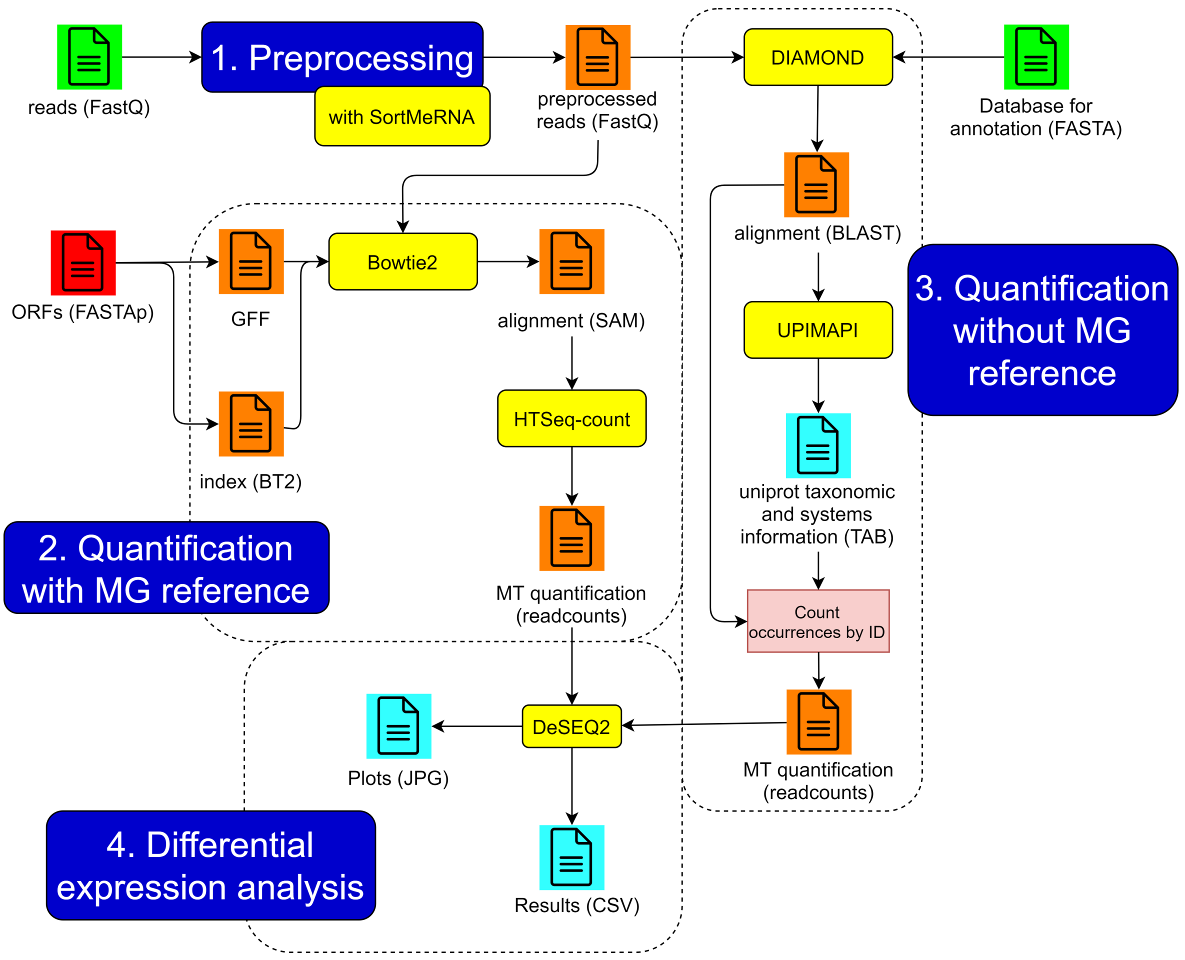

reads (FastQ)
1. Preprocessing
with SortMeRNA
preprocessed reads (FastQ)
DIAMOND
Database for annotation (FASTA)
alignment (BLAST)
ORFs (FASTAp)
GFF
Bowtie2
alignment (SAM)
UPIMAPI
3. Quantification without MG reference
HTSeq-count
index (BT2)
uniprot taxonomic and systems information (TAB)
2. Quantification with MG reference
MT quantification (readcounts)
Count occurrences by ID
Plots (JPG)
DeSEQ2
MT quantification (readcounts)
4. Differential expression analysis
Results (CSV)

Figure 2: MOSCA MT architecture Workflow. Blue icons: the main steps of the workflow; Yellow icons: the bioinformatics tools utilized in each step; Green icons: the inputs needed to perform the analysis; Orange icons: the intermediate files i.e., files outputted by one step that serve as input to another step; Red icons: the intermediate files needed to be produced by other analysis i.e., MG step; and Navy Blue icons: the output files.

Preprocessing is done as described for the MG, with the addition of removing reads identified as rRNA by SortMeRNA (Kopylova et al., 2012).

Quantification and normalization takes as input preprocessed MT reads and aligns them to a reference. If MG data is provided with MT, contigs from the MG workflow will be used as reference. Otherwise, MT assembly with Trinity will provide contigs that serve as estimations of the transcripts that generated MT reads, and those will be used for reading, mapping and quantification. MT normalization in MOSCA uses either TMM or RLE, with the option of which left to the user. These are two methods wisely used for RNA-Seq Analyses (Maza, 2016).

Differential expression analysis takes normalized readcounts coming from the Quantification phase of MOSCA MT, and performs statistical analysis to discover significant changes in expression level between experimental groups. The analysis is made using DeSEQ2 (Love et al., 2014), which estimates variance-mean dependence in count data from high-throughput sequencing assays and tests for differential expression based on a model using the negative binomial distribution. This phase produces two different outputs, several plots (in JPG format), and significance reports in CSV format.

2.1.3 *Metaproteomics*

MP identifies the main functions driving microbial behavior by performing the identification and quantification of proteins (Wilmes et al., 2015). MP pipelines workflows usually contain the following main steps: Database construction, Protein identification, Protein quantification and Differential expression analysis. These steps are included in MOSCA's MP workflow (Fig. 3).

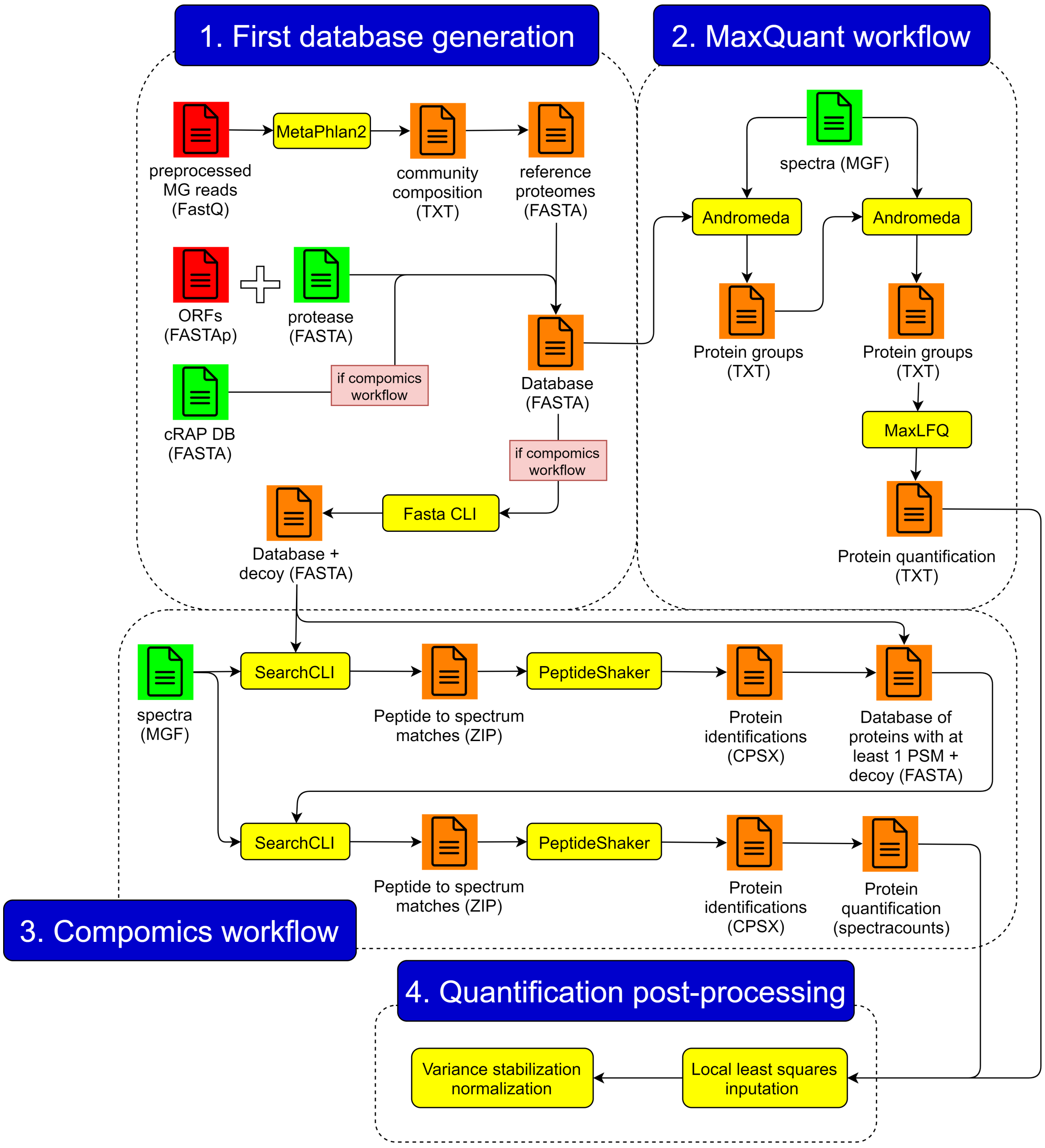

1. First database generation
preprocessed MG reads (FastQ)
MetaPhlan2
community composition (TXT)
reference proteomes (FASTA)
ORFs (FASTAp)
protease (FASTA)
cRAP DB (FASTA)
if compomics workflow
Database (FASTA)
if compomics workflow
Fasta CLI
Database + decoy (FASTA)
2. MaxQuant workflow
spectra (MGF)
Andromeda
Andromeda
Protein groups (TXT)
Protein groups (TXT)
MaxLFQ
Protein quantification (TXT)
3. Compomics workflow
spectra (MGF)
SearchCLI
Peptide to spectrum matches (ZIP)
PeptideShaker
Protein identifications (CPSX)
Database of proteins with at least 1 PSM + decoy (FASTA)
SearchCLI
Peptide to spectrum matches (ZIP)
PeptideShaker
Protein identifications (CPSX)
Protein quantification (spectracounts)
4. Quantification post-processing
Variance stabilization normalization
Local least squares inputation

Figure 3: MOSCA MP architecture workflow. Blue icons: the main steps of the workflow; Yellow icons: the bioinformatics tools utilized in each step; Green icons: the inputs needed to perform the analysis; Orange icons: the intermediate files i.e., files outputted by one step that serve as input to another step; and Red icons: the intermediate files utilized by tools produced in different analyses i.e., MG.

First database generation receives as input the preprocessed MG reads (in FastQ format) and the ORFs (FASTAp) from MG Annotation and builds a FASTA database for MP with all the possible sequences to be present in the dataset studied. MetaPhlan2 (Truong et al., 2015) characterizes taxonomically the preprocessed reads, and for the taxa obtained it retrieves reference proteomes, which are added to the database. The sequence of the protease used (in FASTA format) is also added to the database. Two different workflows are available. In the Compomics workflow, the cRAP database (common Repository of Adventitious Proteins, containing a set of proteins commonly found in proteomics experiments which are mainly contaminants) will be added, and a decoy database will be generated using FastaCLI from SearchCLI (Wang et al., 2012) from the complete collection of proteins. If using the MaxQuant workflow, the Database will be inputted directly to Protein identification, without decoy database generation. The main output of this step is either the decoy or the base database, in FASTA format.

MaxQuant workflow uses MaxQuant, a quantitative proteomics software designed for analyzing large mass-spectrometric datasets. MaxQuant workflow receives the respective Database generated from the previous phase and spectra (MGF) data files. It includes Andromeda (Cox et al., 2011), which is a search engine based on a probability calculation for the sorting of peptide-spectrum matches (PSMs). PSMs allow to relate spectra to peptides present in the database. MaxQuant uses Andromeda to determine mass-dependent recalibration function based on a preliminary database search and used after again to identify one or more fragmented peptides, transforming the previous inputs information into a Protein groups (TXT) file. The last step of the MaxQuant workflow, it uses MaxLFQ (Cox et al., 2014) a pipeline for proteome-wide quantification, transforming Protein groups files into Protein quantification (TXT) that is used in stage four Quantification post-processing.

Compomics workflow receives a decoy database from the previous stage, and spectra (MGF) files as input. It uses SearchCLI to match peptides to spectra and PeptideShaker and to identify the proteins from PSMs. Protein quantification is obtained as spectra counts from PepetideShaker.

Quantification-Preprocessing consists in the utilization of two different methods to realize statistical analyses, the first being the Local least squares imputation (Zhang et al., 2008), used to substitute missing values with estimations obtained from the available quantifications, and variance stabilization normalization is used to stabilize the variance of the data and normalize it (Lin et al., 2008).

2.2 AUTOMATION OF MOSCA PIPELINE

MOSCA is a tool that uses Snakemake to achieve complete automation (Mölder et al., 2021). Snakemake workflows are specified by decomposing them into steps represented as rules. Each rule specifies how to generate a set of output files from a set of input files. A shell command, a block of Python code, an external script (Python, R), a Jupyter notebook, or a wrapper can accomplish this. The first rule is only defined with inputs and no outputs, and Snakemake rollback rules until it finds that the inputs of a precedent rule. If those inputs are present, it will begin by running that rule until it reaches the last one, which is the rule with only inputs. As a result, if the workflow breaks in the middle of an analysis, Snakemake workflows can resume from the point where it broke without losing any information gathered in previous rules. Snakemake can accomplish this by utilizing the various files already generated in previous run rules and by searching the input files from the various rules it detects the last rule run, starting from that rules again. In the case of MOSCA pipeline the snakemake workflow contains 15 rules, ″all″, ″preprocess″, ″join reads″, ″assembly″, ″binning″, ″fastq2fasta″, ″annotation″, ″recognizer″, ″quantification analysis″, ″metaphlan″, ″protein report″, ″entry report″, ″differential expression″, ″keggcharter″ and ″report″. Rule ″all″ is the last rule of the workflow containing only inputs. ″join reads″ and ″fastq2fasta″ are intermediate rules, they work to transform files to be used by other rules. ″report″ is the rule producing three files containing the principal information obtained in all the steps of MOSCA. Any other step is a principal step for producing and obtaining results. Fig.4 shows the dependencies between each step of MOSCA.

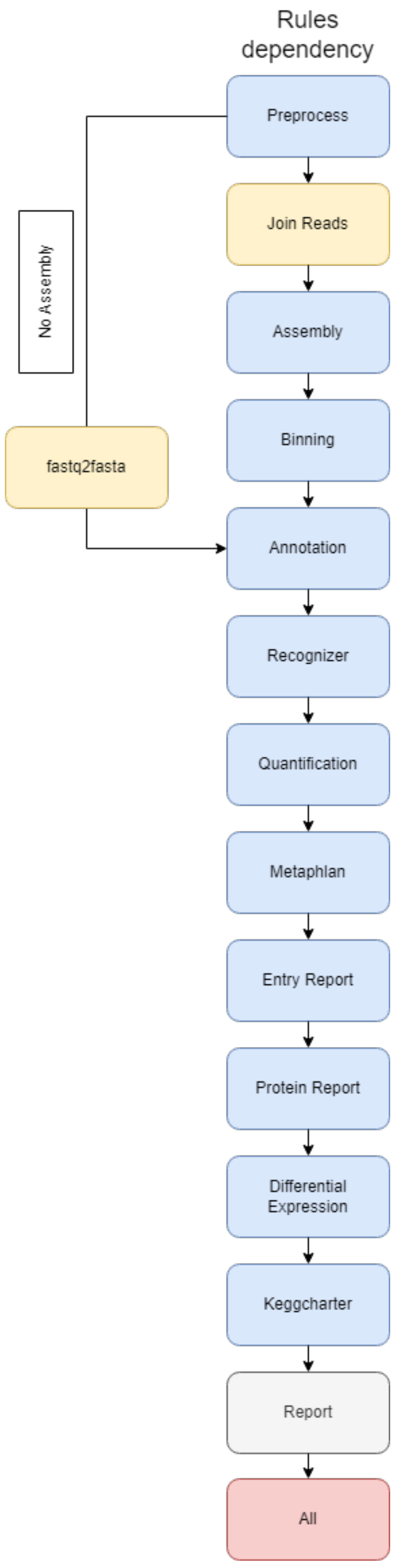


Figure 4: MOSCA rules dependencies, in blue is show the main steps of MOSCA; in yellow intermediate steps; grey a step to produce output files; and red the rule "all" to check that all rules have run with success.

2.3 MOSGUITO: INTERACTIVE WEB-BASED APPLICATION FOR MOSCA PIPELINE

MOSGUITO is an interactive web-based application implemented using ReactJS (*ReactJS.org, ' ReactJS official'. [Online]*, 2022). MOSGUITO is available at https://iquasere.github.io/MOSGUITO and is only able to build a configuration file to use in MOSCA and to upload a zip file with results from MOSCA analyses. MOSGUITO is organized into seven pages a "config" page that provides input fields to set general configurations affecting the entire workflow, especially the MG and MT workflows; an "experiments" page that provides a metadata table about each dataset; a "uniprotColumns", a "uniprotDatabases" and a "keggmaps" pages that provide selection panels where Uniprot columns and Uniprot databases for UPIMAPI and metabolic maps for KEGGCharter are set, respectively; a "proteomicsConfiguration" page which provides fields of input to configure the MP workflow; and a "results" page providing a button to upload a results ZIP file outputted by MOSCA at the end of its analysis, displaying the results inside MOSGUITO(Fig. 5).

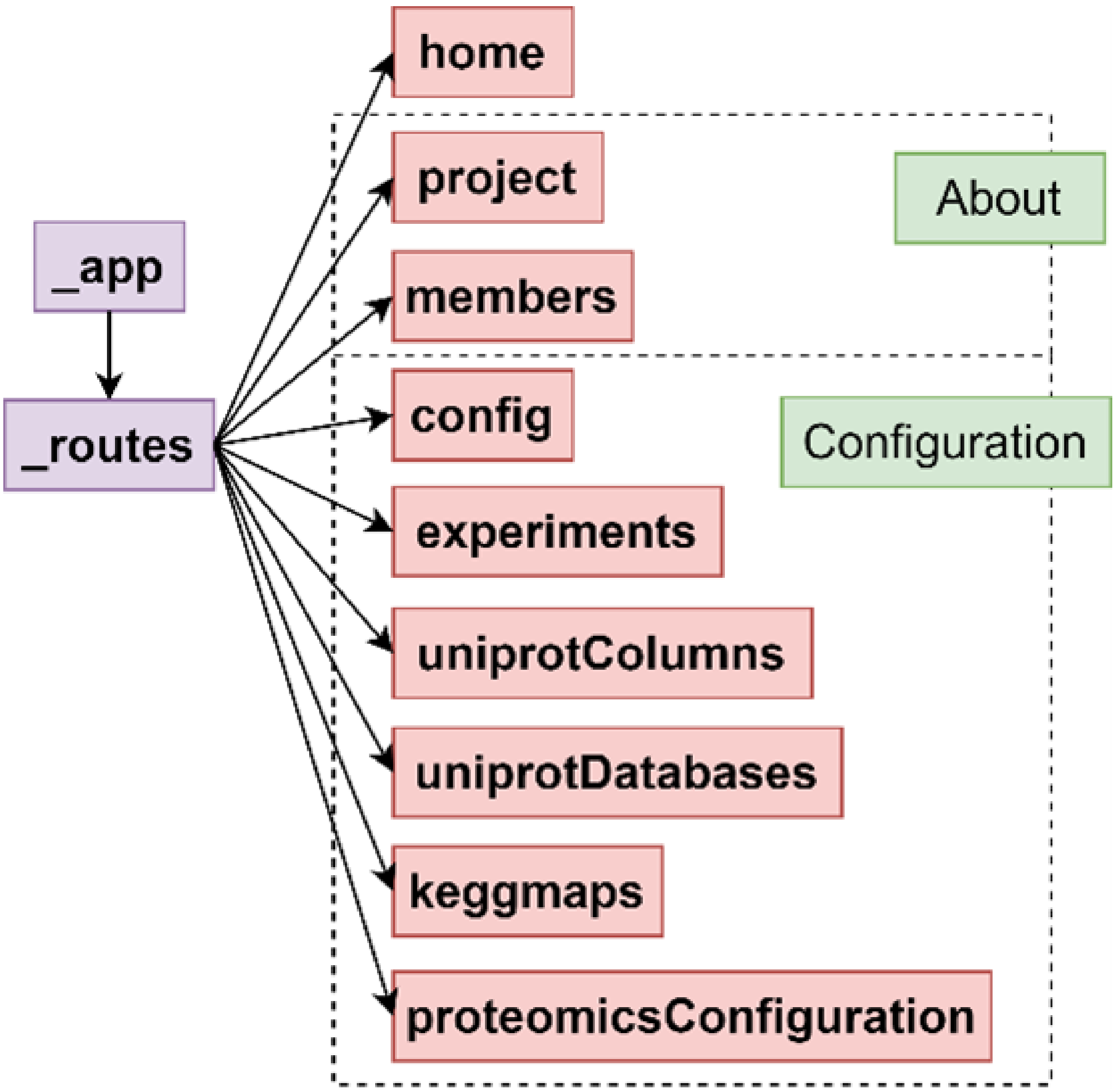


Figure 5: Application structure of MOSGUITO. Purple boxes represent the entry point to the application, red boxes represent the pages available for user interaction and green boxes represent the categories of pages, inside which the pages are nested in the side bar.

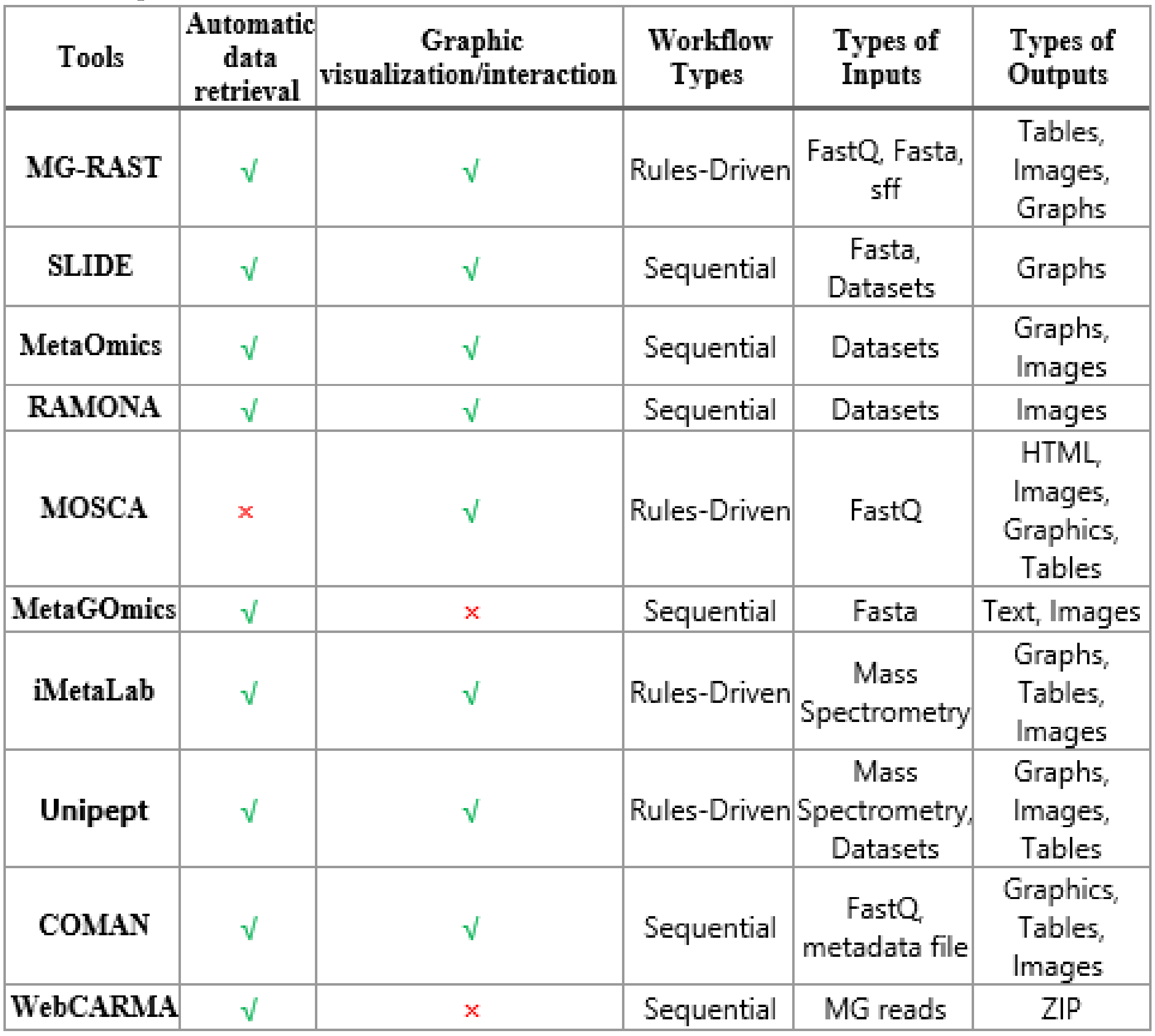
2.4 OVERVIEW OF WEB SERVICES PERFORMING ANALYSIS OF OMICS DATA

There are already several pipelines developed for the study of omics, many of them available online. Table. 1 presents a general comparison of some of the most popular. The pipelines workflows can be categorized into three different types: "Sequential" is frequently chart-based, progressing from one phase to the next, each stage is dependent on the previous phase's actions being completed; "State Machine" is a little more complex, and activities in these workflows frequently switch back and forth between steps as needed and; "Rules-

Driven" which is carried out in accordance with a sequential procedure governed by rules. MG-RAST (Wilke et al., 2015) is a web tool with a Rules-Driven workflow that receives raw sequence data as input. Users of MG-RAST can upload raw sequence data in the formats of fastq, fasta, and sff. MG-RAST outputs tables with taxonomic information, Images, graphs, for example, compare annotation data, PCoA, heatmaps, and dendrograms it let users download files resulting from each step and processing. SLIDE (Ghosh et al., 2018) is an open-source, web-based tool to visualize large-scale-omics data interactively. SLIDE accepts a text-delimited input data file containing a matrix of expression values (comma, tab, space, semi-colon, pipe). SLIDE contains a Sequential workflow, outputs a quantitative visualization of omics data in expression-based heatmaps on a standard web browser, and allow users to move around the heatmaps interactively and make sub-analyses of specific feature groups. MetaOmics (Ma et al., 2018) is a comprehensive analytical pipeline and browser-based software suitable to meta-analyze multiple transcriptomic studies for various biological purposes, including quality control, differential expression analysis, differential co-expression network analysis, clustering, prediction, pathway enrichment analysis, and dimension reduction. MetaOmics contains a Sequential workflow, receives as input tables in TXT and CSV format, outputs different graphs in JPG format with Enrichment pathways and diagnostic plots and tables with functional annotation and Differential expressed genes to visualize in the local web. RAMONA (Sass et al., 2015) is a Web application for multilevel omics data gene set analysis. RAMONA contains a Sequential workflow, receives as input a list of genes and outputs interactive visualizations of the inferred active terms in the context of their respective pathways or ontology hierarchy. MetaGOmics (Riffle et al., 2018) is a Web application that automates the quantitative functional (using Gene Ontology) and taxonomic analysis of metaproteomics data. MetaGOmics contains a Sequential workflow, receives as input FASTA files and outputs a text report or images comparing the ratios of GO terms in the two experiments, the end users are only able to download the results. iMetaLab (Liao et al., 2018) is a cloud-based platform with a user-friendly web interface that allows general users to quickly acquire peptide and taxonomic abundance information as well as protein function annotation from raw mass spectrometry data. iMetaLab contains a Rules-Driven workflow, receives as input mass spectrometry data and outputs interactive visualizations mass spectrometry performance, identified unique peptides and proteins, taxonomy profile and function annotation. Unipept (Mesuere et al., 2016) is an open source web application that is designed for MP analysis with a focus on interactive data visualization. Unipept contains a Rules-Driven workflow, receives as input mass spectrometry, dataset containing peptide analysis and outputs interactive visualizations heatmaps, tables and graphics containing taxonomy information. COMAN (Ni et al., 2016) is a web-based application for functional characterisation and comprehensive analysis of high-throughput MT data. COMAN contains a Sequential workflow and receives as input

Illumina paired-end sequencing reads in FASTQ format and a metadata file specifying the sample conditions for comparative statistical analysis and outputs interactive visualizations KeggMaps, Graphics containing statistic information and tables. WebCARMA (Gerlach et al., 2009) is a web application for the taxonomic and functional classification of unassembled (ultra-)short reads from MG communities. WebCARMA contains a Sequential workflow and receives as input MG reads and outputs ZIP file containing taxonomic and functional information.

Table 1: Comparison of different web services for meta-omics analysis. Comparing Automatic data retrieval, if the tool can retrieve the results from the pipeline to the web; workflow types, if the workflow Rules-Driven, Sequential, or a State Machine; Types of inputs; and types of outputs.



Tools	Automatic data retrieval	Graphic visualization/interaction	Workflow Types	Types of Inputs	Types of Outputs
MG-RAST	√	√	Rules-Driven	FastQ, Fasta, sff	Tables, Images, Graphs
SLIDE	√	√	Sequential	Fasta, Datasets	Graphs
MetaOmics	√	√	Sequential	Datasets	Graphs, Images
RAMONA	√	√	Sequential	Datasets	Images
MOSCA	×	√	Rules-Driven	FastQ	HTML, Images, Graphics, Tables
MetaGOmics	√	×	Sequential	Fasta	Text, Images
iMetaLab	√	√	Rules-Driven	Mass Spectrometry	Graphs, Tables, Images
Unipept	√	√	Rules-Driven	Mass Spectrometry, Datasets	Graphs, Images, Tables
COMAN	√	√	Sequential	FastQ, metadata file	Graphics, Tables, Images
WebCARMA	√	×	Sequential	MG reads	ZIP

2.5 CLIENT-SERVER

Client-server is a system architectural model in information technology that consists of two parts: client and server systems that communicate across a computer network. A client-server application is a type of distributed system that includes both client and server software. The client-server application allows for more efficient task distribution. The client process is constantly connecting to the server, while the server process is still waiting for requests from any client. A client is a computer hardware device that runs software and connects to a server to access a service. A server is a computer with dedicated software that runs on it and provides services to other machines. The client-server concept described how a server makes services and resources available to one or more clients. A one-to-many relationship exists between server and client, which means that a single server can give internet resources to several clients at the same time. When a client requests a link to a server, the server might accept or refuse the request. If the link is permitted, the server will create and maintain a connection with the client using a specified protocol.

2.5.1 *Client-Server architecture*

The client-server architecture can be divided into four different categories: one-tier, two-tier, three-tier, and N-tier (*Software Architecture: One-tier, Two-tier, Three Tier, N Tier*, n.d.) (Fig.6). A one-tier application, commonly referred to as a standalone application, unifies all of the software's levels, including the display, business, and data access layers. Client applications (client tier), often referred to as client-server applications, and databases are separated into separate components of application design by two-tier architecture (data tier). The presentation layer (sometimes referred to as the client tier), the application layer (often referred to as the business tier), and a database layer make up three-tier architecture, or web-based application architecture (data tier). A three-tier design is similar to an N-tier architecture, but the number of application servers is increased and distributed over more levels to better accommodate business logic. Client-Server system are divided into various components (Kumar, 2019):

- **Client** is process or program known as a "client application" delivers a job request to a server across a communication network.
- **Server** is a group of programs and watches for client requests that are sent through the communication network.
- **Application Server** is a component-based product found in the middle tier of a server-client architecture.

- **Database Server** is a particular kind of server that offers access and retrieval of data from databases.

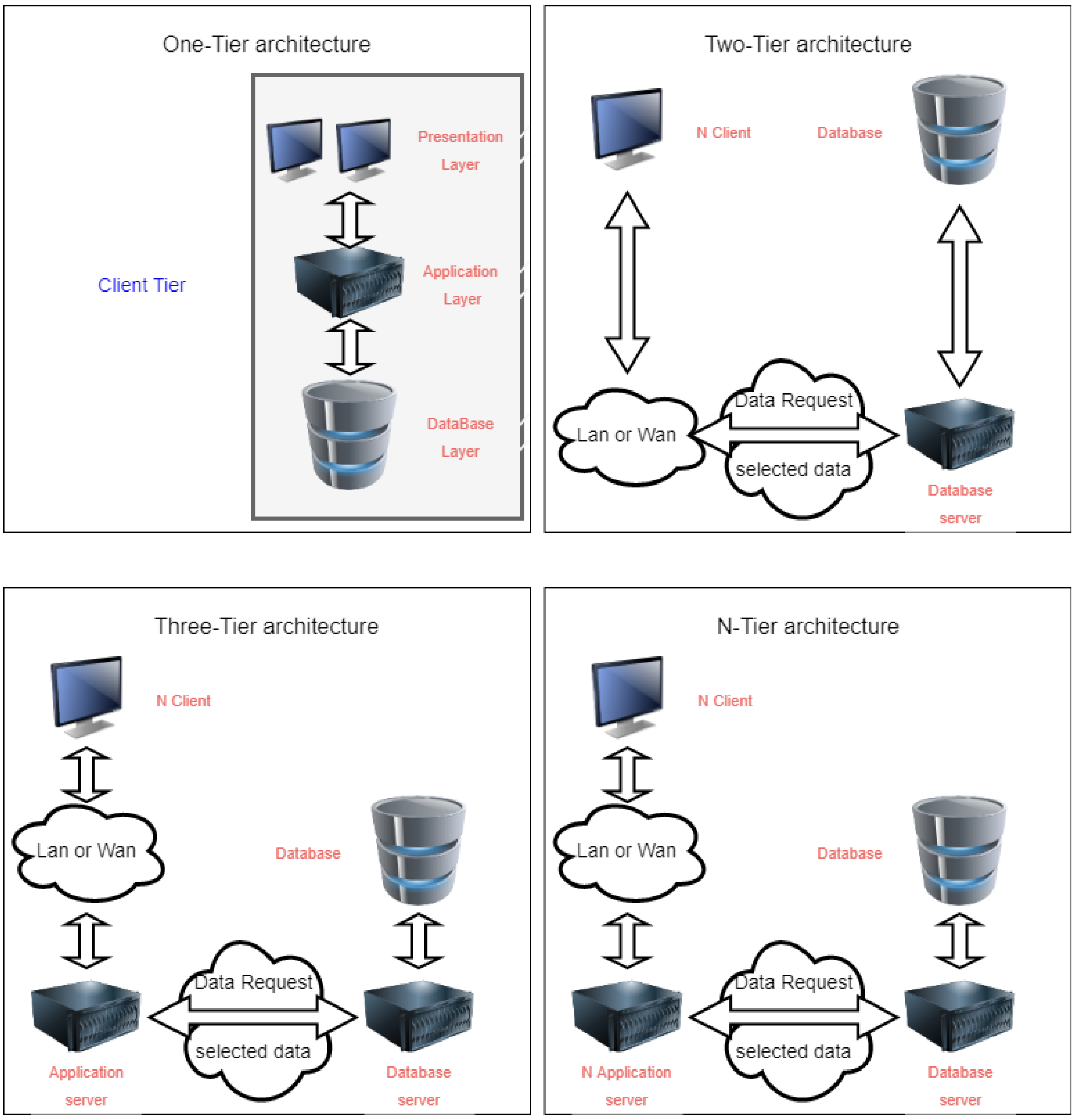


Figure 6: Different categories of client-server architectures one-tier, two-tier, three-tier and N-tier.

2.5.2 *Server*

A Server with multiple applications to run independently and to not overcharge a server, normally are deployed inside Docker containers Merkel (2014).

Docker

When applications are deployed inside containers an additional layer of deployment is added on top of the container environment where the programs are virtualized and run. Docker is made to provide a speedy, light environment where code can run effectively, and it also offers the additional capability of a professional work procedure to remove the code from the computer for testing before to production. Docker enables you to test your code and deploy it into the production environment as quickly as feasible. To be able to manage, scale and automate deployment into containerized applications is necessary the use of a complex tool called Kubernetes Tesliuk et al. (2019).

Kubernetes

The open-source orchestrator Kubernetes is utilized to develop, maintain, and deploy containerized applications (microservices). It offers scaling with decoupled design and container APIs with rapid shipping. It serves as a self-service platform that gives the developer team access to a hardware abstraction layer. Developer can immediately access the resources and manage the increased workload as a result. Fig.7 shows the principal concept of a kubernetes architecture containing nodes with a service and pods, a pod is a virtual server and a service is responsible for the interaction with pods, the CTRL plane shows the Kubernetes API containing backing store data about pods and respective services.

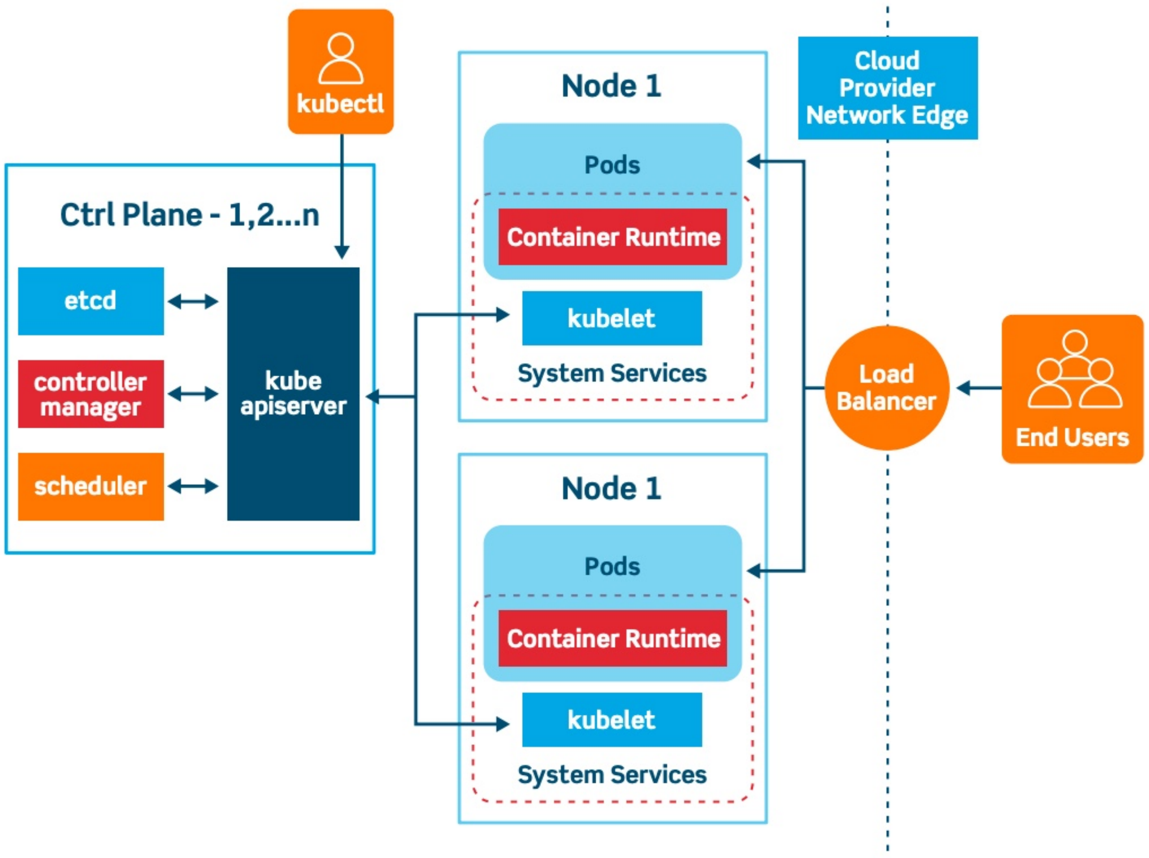


Figure 7: Kubernetes architectures concept, divided into nods containing an instance to run, a Load Balancer to connect the users outside of the cluster to the respective nodes, and a control API that stores information from the nodes and services.

2.5.3 *Database Management System*

A Database Management System (dbms) is software to construct and manage databases by running queries on the data performing CRUD (Create, Read, Update and Delete) (CRUD) operations, Create operation is used to create new records to a database, Read operation is used to retrieve information from the database, Update operation is used to modify a record inside the database and Delete operation is used to remove a record from a database. A database is a structure built with related lists. It serves to store information. Without a dbms, it is impossible to retrieve information from the database. Databases can be categorized into different types: Hierarchical Databases that use a parent-child model to store data allowing the data to appear to be in a tree format with one object on the top; Object-oriented databases, which are based on object programming, being the data tied to a unique object, these databases are managed by oriented programming languages; and Relational databases, the data is stored in tables that are related, and using a dbms is possible to perform CRUD operations to them. Python frameworks support Object-oriented databases, e.g., MongoDB (Membrey et al., 2010), Relational databases, e.g., My SQL (Svehring, 2021), and relational-object-oriented databases e.g., PostgreSQL (Obe and Hsu, 2017).

2.6 WEB SERVICE

Web services are a software that uses defined messaging protocols and is made accessible for usage by a client or other web-based programs through an application service provider's web server. To interface with diverse applications, web services are constructed utilizing open standards and protocols (Ankolekar et al., 2002). Web services employ the following protocols:

- **Extensible Markup Language (XML)** - This is used for data tagging, coding, and decoding (Bray et al., 1997).
- **Simple Object Access Protocol (SOAP)** - The data is transferred through this. The SOAP protocol was created to make it simple and quick for various computer languages to communicate with one another (Box et al., 2000).
- **Web Services Description Language (WSDL)** - This is used to inform the client application about the web service's contents and connection details (Christensen et al., 2001).
- **Universal Description, Discovery and Integration (UDDI)** - This is used to list the services that are offered by a certain application. Additionally, it enables other services to find web services (Richards, 2006).
- **Representational State Transfer (REST)** - Although not all web services use the REST protocol, programs created with RESTful APIs are lighter, easier to control, and more scalable (Ong et al., 2015).

2.6.1 *Types of Application program interface*

An API is a way for two computer applications to trade information over a network using a common language that they both understand. An API can provide a hook for colleagues, partners, or third-party developers to access data and services. Some APIs are open to any developer, while others are open only to partners or are used internally to help run a business better and facilitate collaboration between teams. APIs can be private, partner, or public (Nordic, n.d.):

- **Private APIs** pair two processes directly. These APIs usually provide access to data and can also provide integration components.
- **Partner APIs** represent business relationships APIs for trading information between different private or public entities.
- **Public APIs**, also often referred to as open APIs, are publicly available and there is no limitation in accessing them.

Public means that the API is available to almost anyone with little or no contractual arrangement with the API provider. Private APIs and Partner are used in a variety of ways, whether to support internal API efforts or a partner's use of the API.

RESTfull API definition

For an API to be considered a RESTful API, it needs to follow some architectural constraints:

- Needs to have a uniform interface. Interface uniformization consists of having an identification of constraints, meaning the use of Uniform Resource Identifier (URI) that is a Uniform Resource Locator (URL) plus a name of the resource, Manipulation of resources using HTTP requests with GET to retrieve information from the respective resource, Self-descriptive messages, using MIME types, description in code of a file type e.g., txt (test/plain), image(image/gif) in the HTTP request so the client can find data and last Hypermedia as the engine of application state creating URI templates, this URI is common, only changing a few characters in the respective URI from case to case.
- Needs to be Stateless, this means the server-side of the software can't store any user information, needing that all the information needed must be passed in the URI.
- Needs to have a Cache system, that way the software can store frequently accessed data in several places in the request-response path, that way the response of the system can be faster if the request is present in the cache that way the request doesn't need to reach the service.
- Needs a Client-Server, a client-server is the interface of the page web, it doesn't store any information, only the user state.
- Needs to have a Layered System, which means having different parts of the current software in different servers, database in a server, server-side in other and client-side in other, that way a user never knows to which server he is connected, improving security (Fielding, 2000).

If any of these constraints are not implemented, hypothetically an API or software can't be called RESTful.

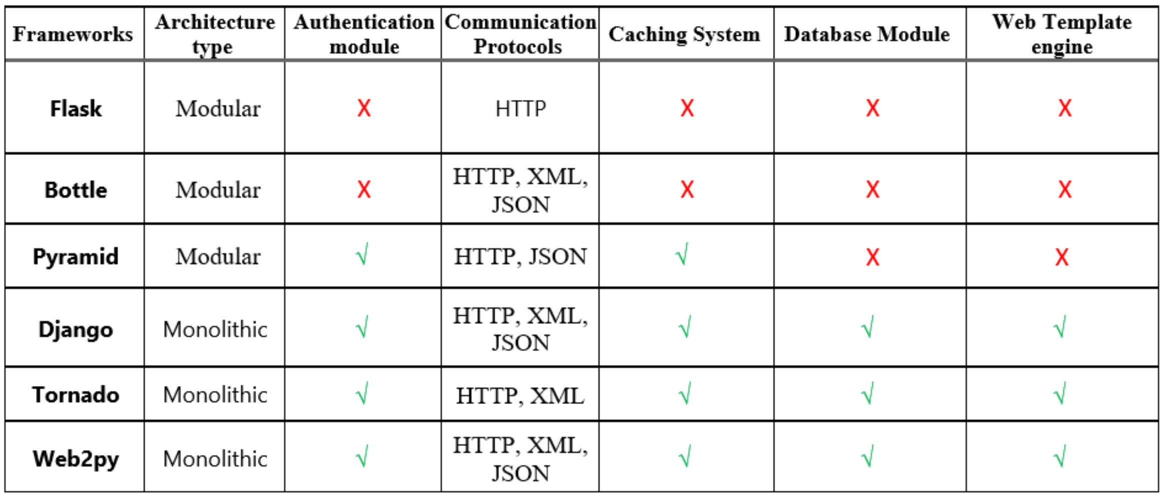
Python Frameworks to developed RESTful APIs

To facilitate the development of APIs, Python web Frameworks have been developed. A web framework is a collection of packages and modules which allows developers to write Web applications. Frameworks can be divided into two great groups, Micro Frameworks, frameworks that lack functionality like authentication methods, a web template engine, being less complex and easy to learn. E.g., Flask(Aslam et al., 2015) has a Modular architecture meaning is a framework to provide structural support, does not contain any Authentication, database and engine template modules in the out-of-the-box version, and per default is only able to support HTTP requests and responses; Bottle(Hellkamp et al., 2016) has a Modular architecture meaning is a framework to provide structural support, does not contain any Authentication, database and engine template modules in the out-of-the-box version, and per default is able to support HTTP, XML and JSON requests and responses; and Pyramid(McDonough, 2021) has a Modular architecture meaning is a framework to provide structural support, does contain Authentication module and a caching system, but doesn't contain a database and engine template modules in the out-of-the-box version, and per default is only able to support HTTP and JSON requests and responses. The other group is Full-stack frameworks. E.g., Django(Forcier et al., 2008) has a Monolithic architecture meaning is a framework with a vast codebase and structure, does contain Authentication, database and engine template modules and a caching system in the out-of-the-box version, and per default is able to support HTTP, XML and JSON requests and responses; Tornado (Dory et al., 2012) has a Monolithic architecture meaning is a framework with a vast codebase and structure, does contain Authentication, database and engine template modules and a caching system in the out-of-the-box version, and per default is able to support only HTTP and XML requests and responses; and Web2py (Di Pierro, 2013) has a Monolithic architecture meaning is a framework with a vast codebase and structure, does contain Authentication, database and engine template modules and a caching system in the out-of-the-box version, and per default is able to support HTTP, XML and JSON requests and responses, these frameworks are able to use a combination of a database module, authentication module, and a web template engine. The next table compares the different frameworks. (Table. 2)

2.6.2 *Authentications Methods for web Services*

Usually a Web Service is present with authentication process. The authentication process is divided into three interrelated concepts: 'Identification' corresponds to the communication of user identity with an Information system, 'authentication' corresponds to an acceptance that the user identity is correct, this authentication can be used with email or other personal information, 'authorization' corresponds to the privileges of the user and the respective use of their information. Exist different types of API authentication methods, being the

Table 2: Comparison of different python frameworks to develop RESTful APIs. Comparing framework architecture, modular or monolithic; Authentication Method, if the out-of-the-box framework contains it; It shows if the out-of-the-box framework contains an Authentication Method, a caching system, a database module, and a web template engine; It shows the principal communication protocols the framework support, for the requested and receiving messages.



Frameworks	Architecture type	Authentication module	Communication Protocols	Caching System	Database Module	Web Template engine
Flask	Modular	X	HTTP	X	X	X
Bottle	Modular	X	HTTP, XML, JSON	X	X	X
Pyramid	Modular	√	HTTP, JSON	√	X	X
Django	Monolithic	√	HTTP, XML, JSON	√	√	√
Tornado	Monolithic	√	HTTP, XML	√	√	√
Web2py	Monolithic	√	HTTP, XML, JSON	√	√	√

most used HTTP Authentication Schemes Franks et al. (1999), API Keys(Farrell, 2009), OAuth(Hardt et al., 2012) and OpenID Connect(Sakimura et al., 2014).

- **HTTP Authentication Schemes** - HTTP authentication Schemes are divided into basic authentication and bearer authentication. The simplest and most direct way is basic authentication. The sender adds a username and password using this method to the request header. In order to ensure secure transmission, the login and password are encoded using Base64, a method of encoding that reduces the two pieces of information to a string of 64 characters. Bearer Authentication can be understood as "give access to the bearer of this token." The bearer token allows access to a certain resource or URL and most likely is a cryptic string, usually generated by the server in response to a login request.

- **API Keys** - In an attempt to address the early authentication problems with systems like HTTP Basic Authentication, API Keys were developed. Each first-time user in this procedure is given a special generated value, indicating that the person is recognized. The user's unique key, which is sometimes produced based on their hardware configuration and IP information and other times randomly generated by the server that knows them, is used to verify that they are the same user as before when they try to re-enter the system.

- **Oauth** - OAuth consists of using third-party applications to Identify the user, this method uses the third-party application to retrieve user authentication and authorization without the need to interact with user credentials.
- **OpenID Connect** - On top of the OAuth protocol, OpenID Connect adds a straightforward identity layer that enables computer clients to access a user's basic profile information and confirm their identity based on the authorization server's authentication. A client application can authenticate a user and acquire information (or "claims") about that user, such as the user name, email address, and other details, using the sign-in procedure defined by OpenID Connect. A secure JSON Web Token (JWT) called ID token contains user identify information that has been encoded. JWT is an open, accepted mechanism for securely representing claims between two parties that is defined by RFC 7519. You can generate, decode, and verify JWT. Although JWT is a standard, it was created by Autho, a firm that manages identities and authentication through APIs.

3 MATERIALS AND METHODS

MOSGUITO was developed as a web service application with a client server architecture joining MOSCA and MOSGUITO in one application. It was built as a three-tier application having a server, a corresponding client, and a Database to store files and user related data. Flask was used as the main framework to develop the corresponding APIs because is a Modular framework being able to provide structural support to the server MOSCA and is easier to learn and build APIs without the need of an extensive understanding of the respective libraries and vast code like monolithic frameworks.

3.1 SERVER

In order to communicate with the Snakemake scripts containing the workflow of MOSCA which work as a unit starting at one rule and terminating in another, a API was built using Flask. The main goal of the server is to make each rule in the workflow of MOSCA independent. To achieve this, we built input functions for each rule of Snakemake to check if the corresponding input files of the rules exist in a folder where only files from outside the server are inserted. If those files exist, the rule can run independently if necessary, keeping the workflow intact and independent.

3.2 MOTHER API

Mother API serves as an intermediary between the client and the server. The Mother API responsible for connecting to the server can interact with the Database API to insert, delete and retrieve files from the respective user and can send them to the server to start the analyses, by storing and building temporary files from physic files and send them to the respective requester. Cause every input in the MOSCA is defined by a constant name, the Mother API is responsible to rewrite the name of the temporary files that are sent to the server, avoiding server errors. Every function in the Mother API requires a cookie that exists when a user successfully logs in, every time the user makes a request the cookie is sent to the Mother API. If no cookie is sent from the browser to the Mother API containing the user information, the Mother API will respond with an error, and will redirect the user to the Login Page, we will talk more about the cookie in a next section. The Mother API uses Requests *Requests lib* (n.d.) a library that makes it incredibly simple to send HTTP/1.1 requests.

3.3 DATABASE

The database was built using PostgreSQL because is a relational object-oriented database being a combination of an Object-oriented database model and a Relational database model, and in order to create the various tables, columns, and unique sequences, it was built using conventional programming in Python using SQLAlchemy (Myers and Copeland, 2015). SQLAlchemy is a Python library that bridges the gap between conventional programming and real-world databases by providing an Objective-relational mapping. If a cookie is sent from the client-side containing user information, the API of the database is always able to retrieve information from the database. If a cookie is not sent from the client-side containing user information, there is no user to retrieve information from the database, and it is not allowed to pick any information from the Database. The database API can upload files from the output directory's, convert them to temporary files, and send them to the Mother API, which can then send them to the server or to the client-side, allowing the user to analyze or download the required files.

3.4 CLIENT-SIDE

MOSGUITO was transformed to be able to adapt to all the requirements from the servers and from the mother API. Two new pages were added to MOSGUITO related to file management, one that let users upload input files into the database each input file must be categorized by a type, the other let end users consult their inputs files already inputted into the database, and let them delete or download their inputs files. To begin analyses on the server, another page was developed the page is divided in three major steps: the first is selecting the analyses to perform, the second is selecting the input files that will be used in the respective analyses, and the third is the overall name the end user provides to the individual analyses. Previously, results of MOSCA could be uploaded to MOSGUITO; now, users will be able to download all of the files created by the analyses, or display the results in the browser. For the client-side to send information to our Mother APIs using HTTP protocol it has used axios *Axios* (n.d.) , Axios is a nodeJS and browser-based promise-based HTTP client. It is isomorphic (meaning it can run in both the browser and nodeJS using the same codebase).

It uses the native nodeJS HTTP module on the server, and XMLHttpRequests on the client (browser).

3.5 AUTHENTICATION

User oriented web applications usually include a Login authentication, and for the privacy of analyses and results we decided to build an OpenID authentication by utilizing cookies as the main login method. The Authentication is done by the database API, verifying if the credentials coming from the client-side match the user credentials. If credentials are valid the server will send an encoded cookie to the client-side. The presence of the cookie in browser lets user access other URLs from MOSGUITO. The cookie contains user logged in information and the date the cookie was created with an expiration time of one hour, which means after one hour the cookie will expire and send the end user to Login page again, to encode all the information stored in the cookies is used JWT to do the encoding. With the insertion of authentication some adjustments have been done to the APIs, a user every time it requests something to the Mother API a function runs checking if the cookie exists.

4 RESULTS

MOSGUITO as a web service can interact with MOSCA as a server and a database to create and store information from results of MOSCA.

Fig.8 shows the full architecture design of MOSGUITO divided into four main components, the server that uses MOSCA pipeline; the Database using PostgreSQL; the Web Services APIs the ones responsible to connect the MOSGUITO client application to the database and the corresponding server; and the Client-side transformation with the integration of authentication methods using OpenID Connect.

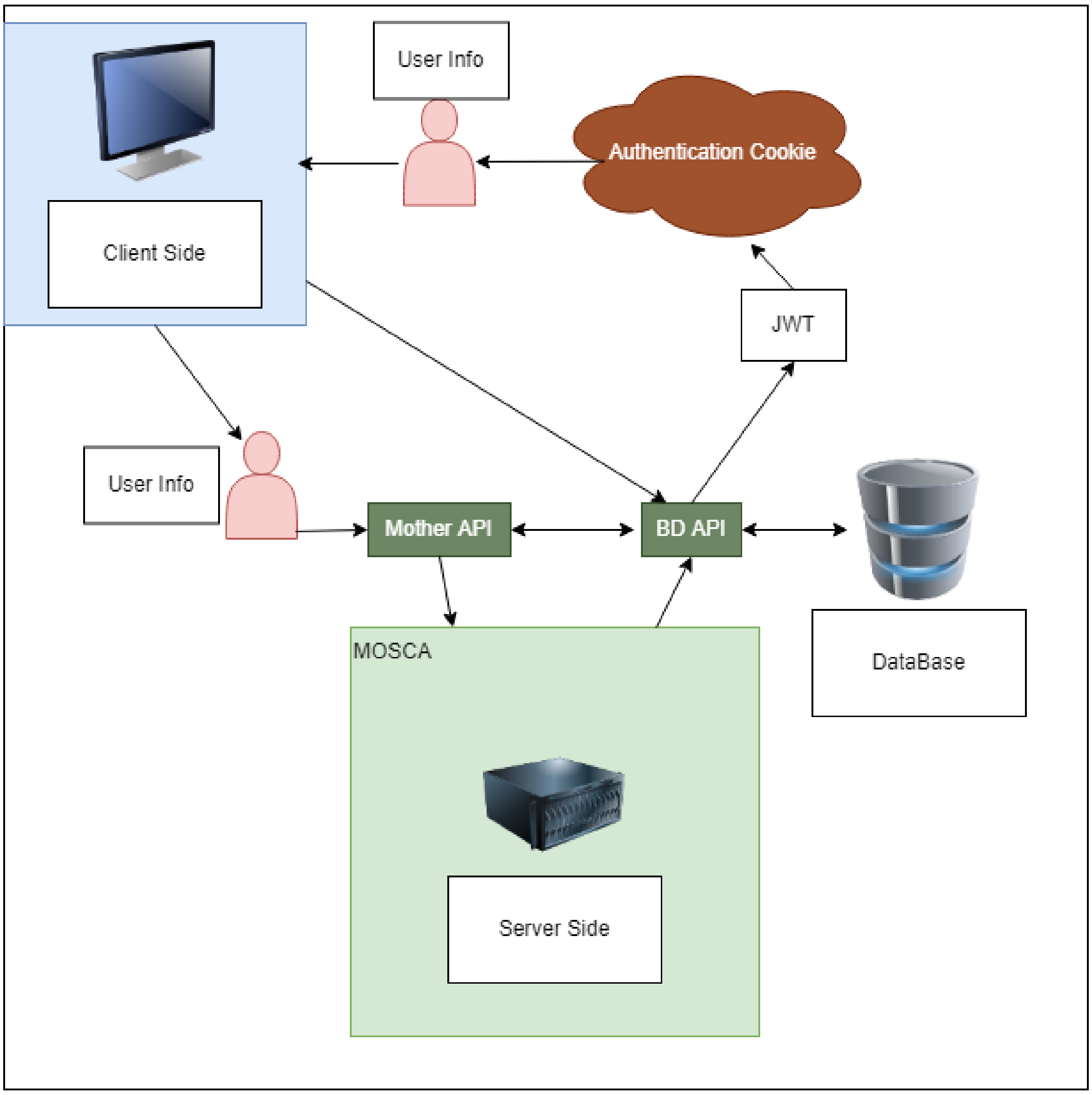


Figure 8: MOSGUITO as a web service implemented architecture, with the construction of the server utilizing MOSCA, the interconnection using APIs and the respective MOSGUITO as the client-side.

MOSCA as a server was transformed, Fig.9 showcase a comparison between the past and current versions of MOSCA workflow and the different inputs each rule is able to receive.

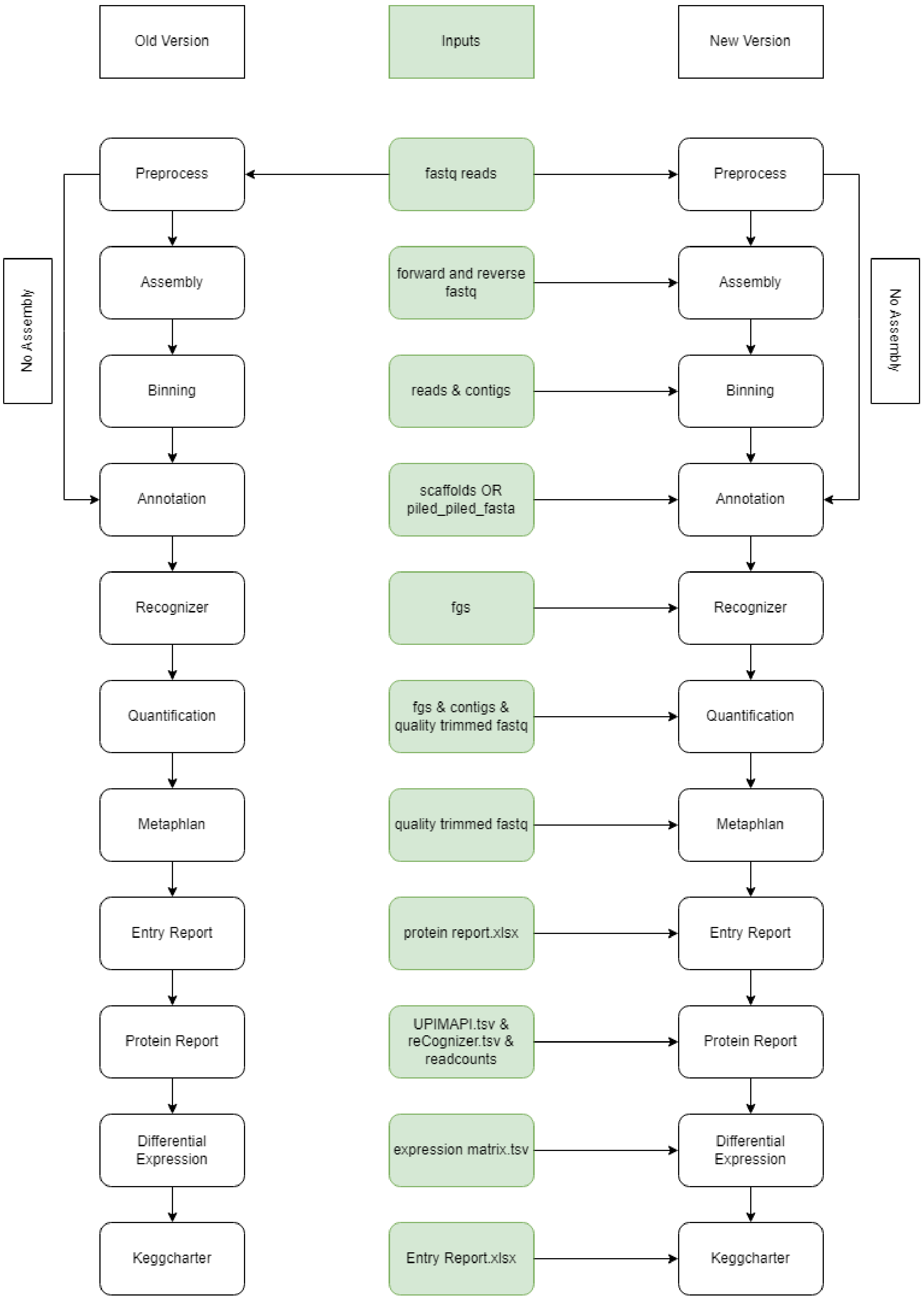


Figure 9: Comparison of the snakemake workflow, in the first version of MOSCA and in the actual version of MOSCA. The first one only receive inputs for the preprocess step, while in the new version is able to receive inputs in all rules, making every rule independent or not from others depending on the presence of their inputs.

For the Database we build the table hierarchy Fig.10 using Python and defined three separate tables one main table and two child ones. As the main table the User Table containing information from end users, is formed by several columns, ″id″ as a primary key; ″google id″ as a primary key to be populated if the account is logged in from OAuth using Google; ″User id″ and ″email″ containing the end user's email; ″Password″ containing the password for the corresponding user, the password in the database is stored as an hashcode, the hashcode is building by hashing the password, hashing uses an encryption algorithm to convert your password (or any other piece of data) into a short string of letters and/or numbers, the hashing of the password is done using werkzeug.security (*werkzeug*, n.d.). As a child table the Input Files containing information about all Input Files, is formed by several columns, the ″parent User Id″ from the user table; ″id″ as a primary key to give each input file a unique key; ″data″ containing the date when the file was stored in the database; ″Type″ containing the type of the file, each type is essential for different rules in the server; The physical files are not stored in the Database, they are stored inside different directory's of the server, the directory's are separated by end user Id. As other child table the Output Files containing information about all the results obtained from the server, is formed by different columns, ″Analyses Name″ that end user gave to their analyses; ″Rules run″ containing all rules run in the analyses, ″parent User id″ coming from the User table, both ″Analyses Name″ and ″user id″ are primary Key and Foreign Key respectively, each user can only have one analyse with a respective name; ″hashcode output″ generated to be attached to each directory containing the results from the analyses, ″hashcode output″ is a unique key, when MOSCA finishes the analyses, a ZIP containing all the results from the analyses are unzipped inside a directory containing ″Output hashcode output″ storing all files inside.

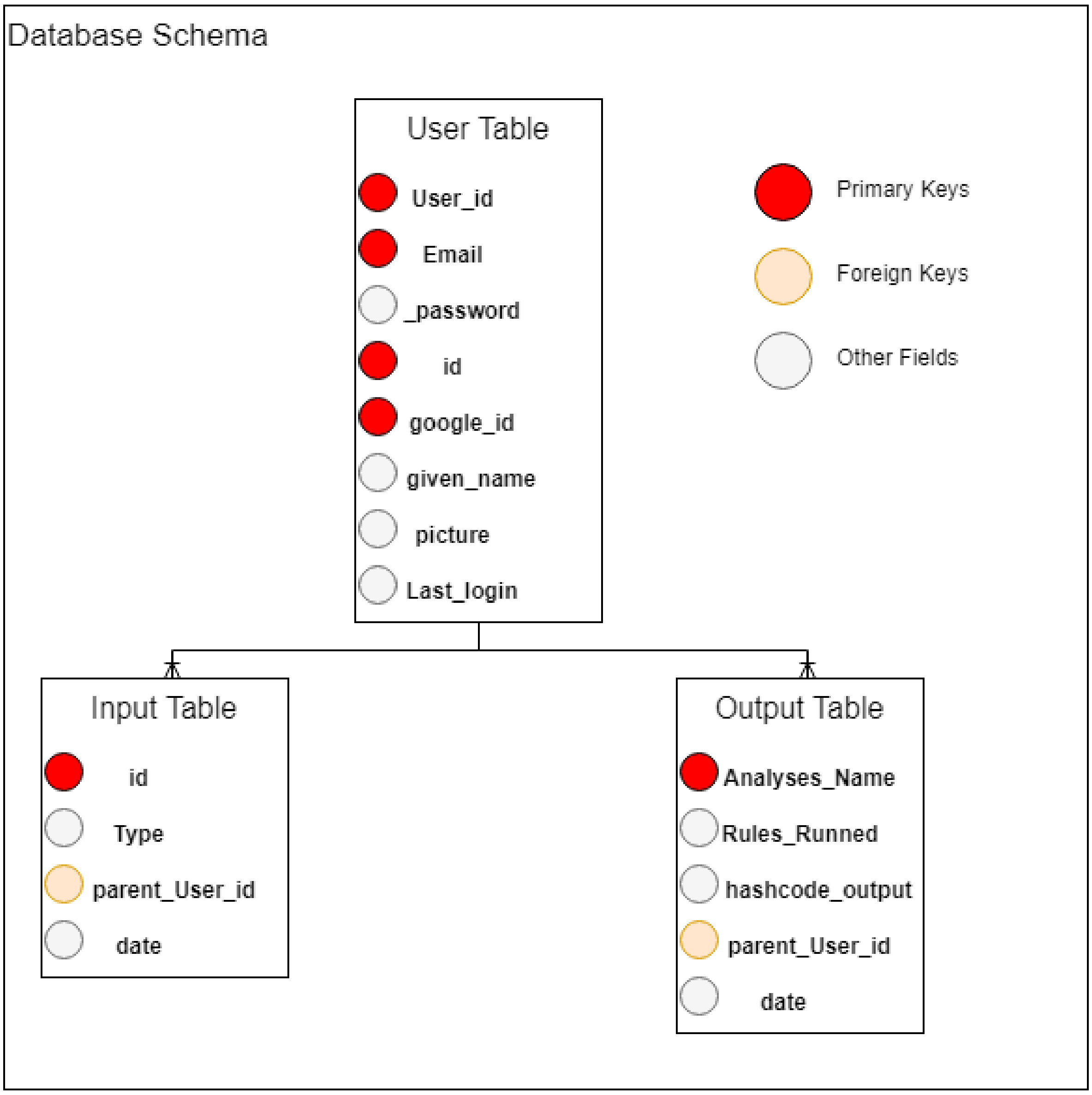


Figure 10: Database Schema with all fields and respective father and child relationships tables, containing the information about primary and foreign keys.

MOSGUITO was tested using a generic user. As a first step it is necessary to Login in the MOSGUITO application using the interface displayed in Fig.11. The interface also allows for new users registration.

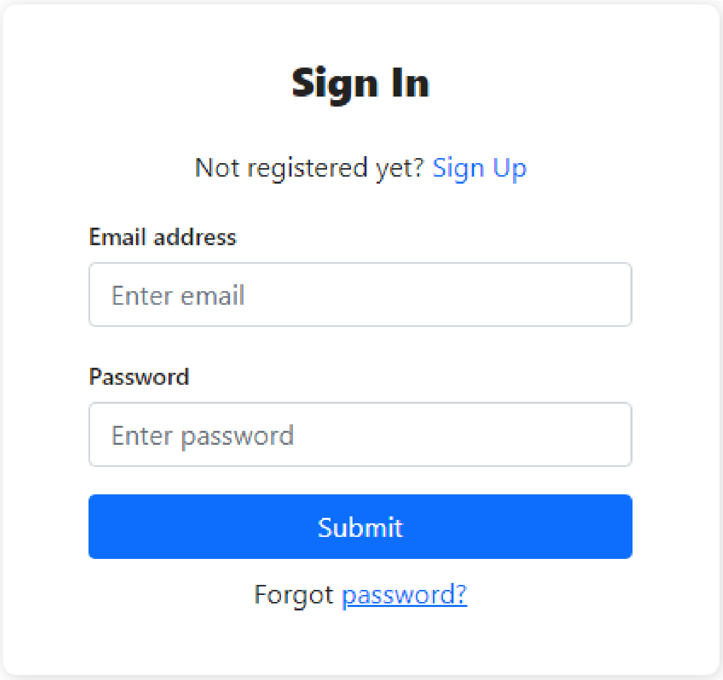
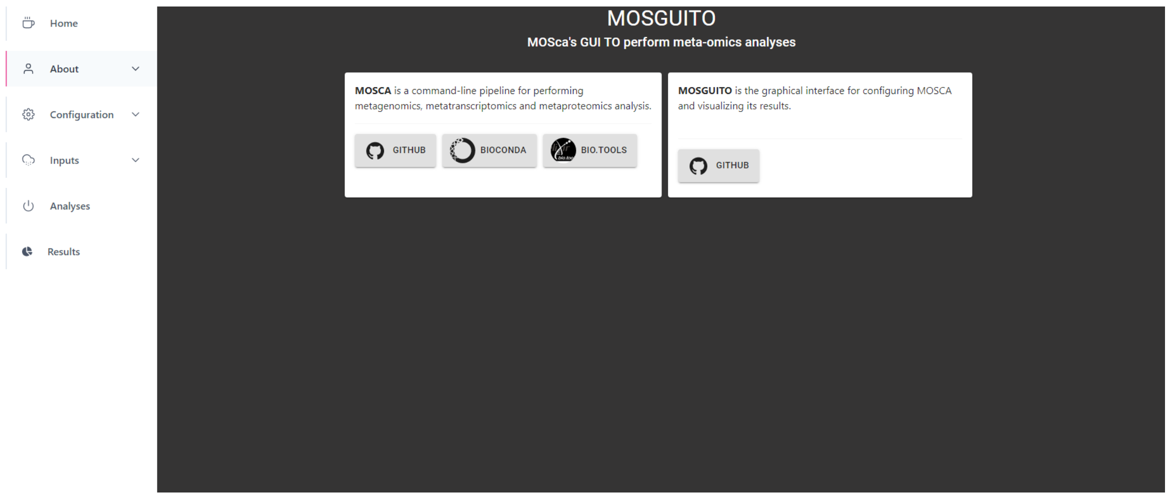


Figure 11: ”Login page” from MOSGUITO

After the authentication authorization, a Home Page Fig.12 appears containing information about MOSCA and MOSGUITO,



MOSGUITO

MOSca's GUI TO perform meta-omics analyses

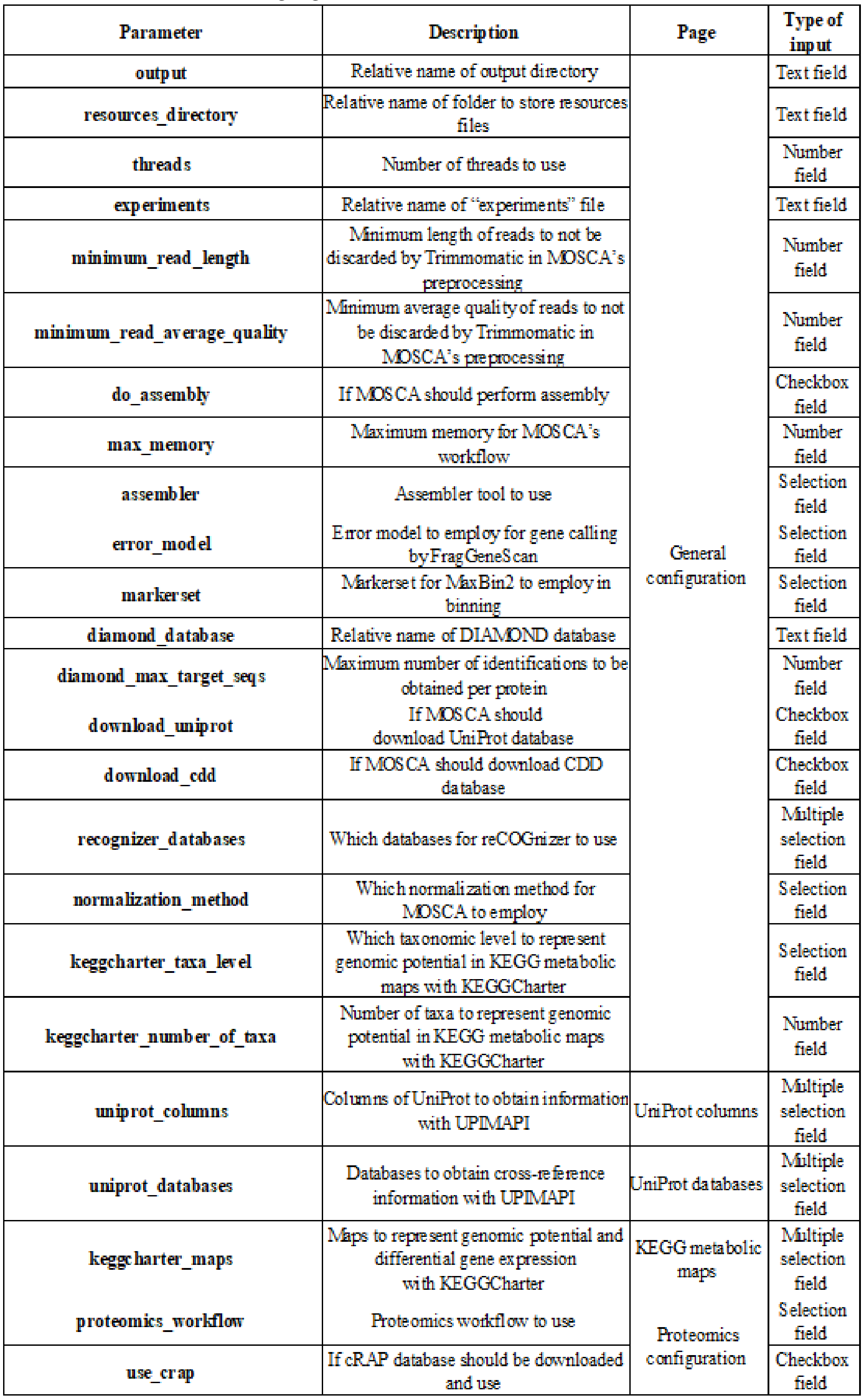
MOSCA is a command-line pipeline for performing metagenomics, metatranscriptomics and metaproteomics analysis.

MOSGUITO is the graphical interface for configuring MOSCA and visualizing its results.

Figure 12: "Home page" from MOSGUITO containing information about the project.

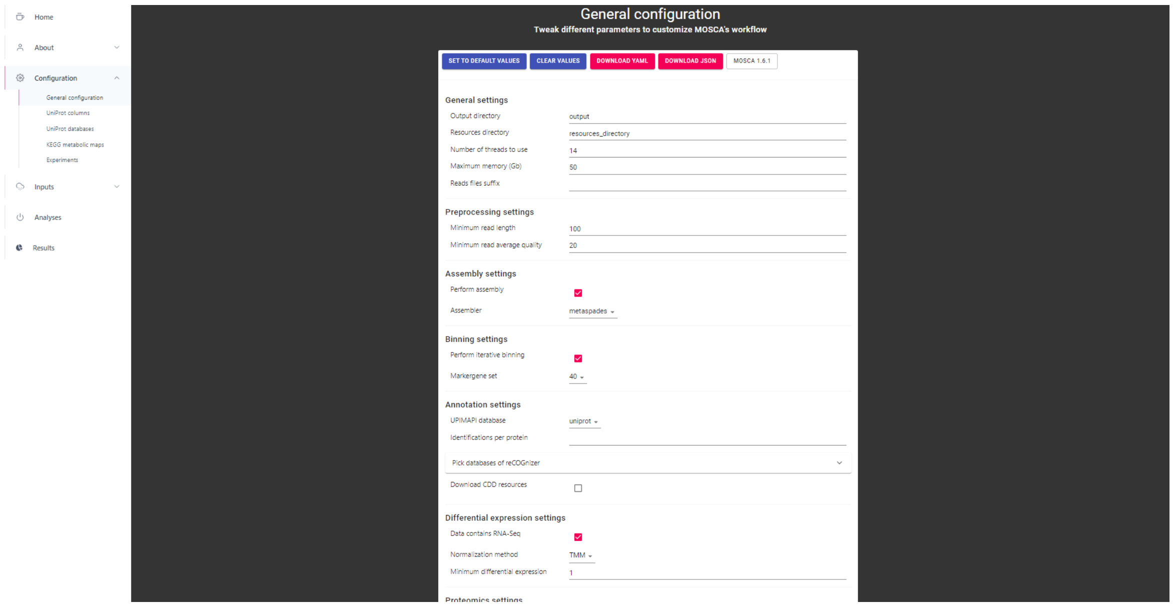
there is another page containing information about the developers and intervention of both applications. The configuration is tweaked in five pages: "General configuration", "UniProt columns", "UniProt databases", "KEGG metabolic maps" and "Proteomics configuration" (Table.3).

Table 3: Parameters of the configuration file and the respective page of MOSGUITO and type of input available for changing its value on the client-side.



Parameter	Description	Page	Type of input
output	Relative name of output directory	General configuration	Text field
resources_directory	Relative name of folder to store resources files	General configuration	Text field
threads	Number of threads to use	General configuration	Number field
experiments	Relative name of "experiments" file	General configuration	Text field
minimum_read_length	Minimum length of reads to not be discarded by Trimmomatic in MOSCA's preprocessing	General configuration	Number field
minimum_read_average_quality	Minimum average quality of reads to not be discarded by Trimmomatic in MOSCA's preprocessing	General configuration	Number field
do_assembly	If MOSCA should perform assembly	General configuration	Checkbox field
max_memory	Maximum memory for MOSCA's workflow	General configuration	Number field
assembler	Assembler tool to use	General configuration	Selection field
error_model	Error model to employ for gene calling by FragGeneScan	General configuration	Selection field
markerset	Markerset for MaxBin2 to employ in binning	General configuration	Selection field
diamond_database	Relative name of DIAMOND database	General configuration	Text field
diamond_max_target_seqs	Maximum number of identifications to be obtained per protein	General configuration	Number field
download_uniprot	If MOSCA should download UniProt database	General configuration	Checkbox field
download_cdd	If MOSCA should download CDD database	General configuration	Checkbox field
recognizer_databases	Which databases for reCOGnizer to use	General configuration	Multiple selection field
normalization_method	Which normalization method for MOSCA to employ	General configuration	Selection field
keggcharter_taxa_level	Which taxonomic level to represent genomic potential in KEGG metabolic maps with KEGGCharter	General configuration	Selection field
keggcharter_number_of_taxa	Number of taxa to represent genomic potential in KEGG metabolic maps with KEGGCharter	General configuration	Number field
uniprot_columns	Columns of UniProt to obtain information with UPIMAPI	UniProt columns	Multiple selection field
uniprot_databases	Databases to obtain cross-reference information with UPIMAPI	UniProt databases	Multiple selection field
keggcharter_maps	Maps to represent genomic potential and differential gene expression with KEGGCharter	KEGG metabolic maps	Multiple selection field
proteomics_workflow	Proteomics workflow to use	Proteomics configuration	Selection field
use_crap	If cRAP database should be downloaded and use	Proteomics configuration	Checkbox field

Parameters in text fields take as input short text, which should correspond to the right values; number fields take as input a positive number, with buttons on the side to increase and decrease the value by 1; checkbox fields can be ticked and unticked for values of true and false, respectively; multiple selection fields allow to pick an option from a limited number of choices. Fig.13 shows a snapshot of the “config” page of MOSGUITO.



General configuration

Tweak different parameters to customize MOSCA's workflow

General settings

Output directory	output
Resources directory	resources_directory
Number of threads to use	14
Maximum memory (Gb)	50
Reads files suffix	

Preprocessing settings

Minimum read length	100
Minimum read average quality	20

Assembly settings

Perform assembly	✓
Assembler	metaspades

Binning settings

Perform iterative binning	✓
Markergene set	40

Annotation settings

UPIMAPI database	uniprot
Identifications per protein	
Pick databases of reCOGnizer	
Download CDD resources	☐

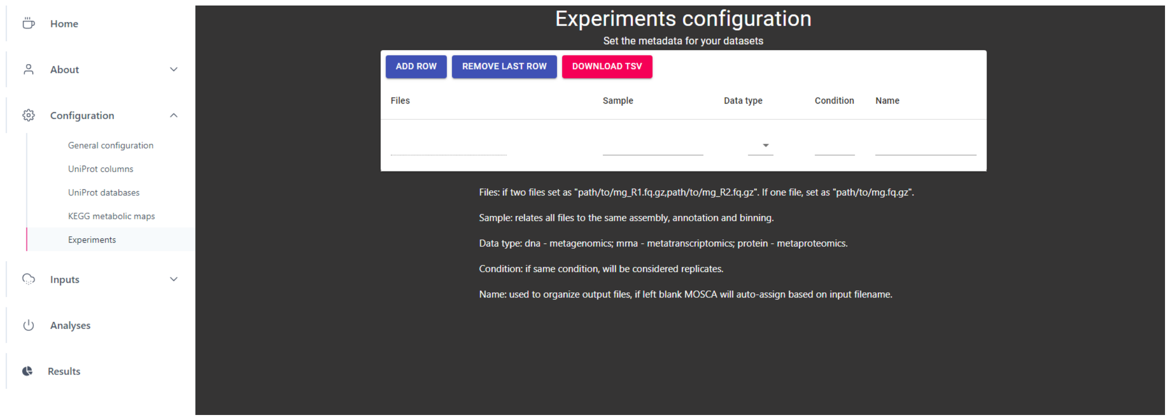
Differential expression settings

Data contains RNA-Seq	✓
Normalization method	TMM
Minimum differential expression	1

Proteomics settings

Figure 13: ”General configuration” page of MOSGUITO, with the side bar on the left, showing the main sections of MOSGUITO and the pages of the “Configuration” section.

The experiments table is set in the “Experiments” page, inside MOSGUITO (Fig.14). The buttons provide the following functionalities: “ADD ROW” adds a new line for inputting a new dataset, “REMOVE LAST ROW” removes the last line of the table, “DOWNLOAD TSV” downloads the data in TSV format. In each row, the “Files”, “Sample”, “Condition” and “Name” take a short text as input, and “Data type” allow for a selection between “DNA”, “mRNA” and “protein”. “Files” are read only and are populated with the name of the files that served as input. Datasets with the same “Sample” value are assembled together, so it should be set for datasets coming from the same communities. Data type determines the type of analysis MOSCA will perform: “DNA”, “mRNA” and “protein” will be analyzed through the MG, MT and MP workflows, respectively.



Files: if two files set as "path/to/mg_R1.fq.gz,path/to/mg_R2.fq.gz". If one file, set as "path/to/mg.fq.gz".

Sample: relates all files to the same assembly, annotation and binning.

Data type: dna - metagenomics; mrna - metatranscriptomics; protein - metaproteomics.

Condition: if same condition, will be considered replicates.

Name: used to organize output files, if left blank MOSCA will auto-assign based on input filename.

Figure 14: “Experiments” page of MOSGUITO. Blue buttons add or remove one line at the end of the table, pink button downloads the data in the table in TSV format, and for each line, information concerning each dataset can be inputted: the files paths, the “sample” category, the type of data, the “condition” category, and a name to be associated in the outputs.

The end user must upload files in order for them to be used in server analyses. The Input page allows users to upload their files and categorize them by type; each type is required for specific analyses. The user is only able to input one file at a time Fig.15.

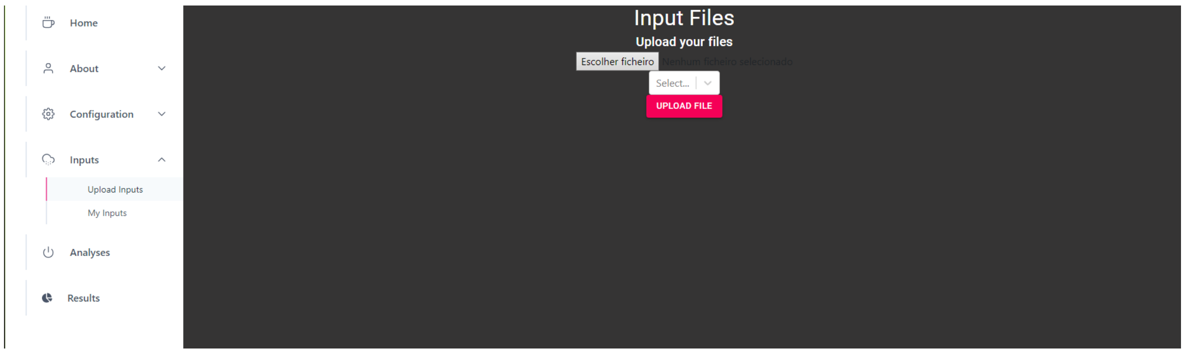
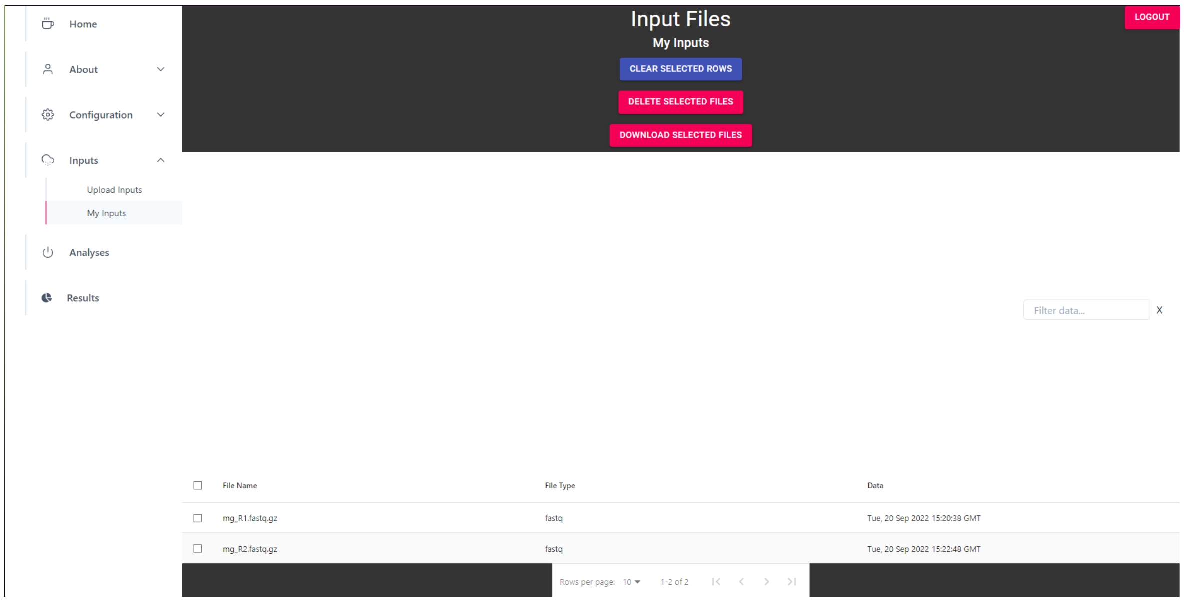


Figure 15: “Upload Inputs” page of MOSGUITO. Red buttons to upload the file, and a dropdown list to select the type of file associated with the input.

A user can also download and delete files from the database using information about their uploaded files obtained in ”My inputs” page from MOSGUITO; each user has a memory size of 20GB for files saved on the server Fig.16.



Input Files

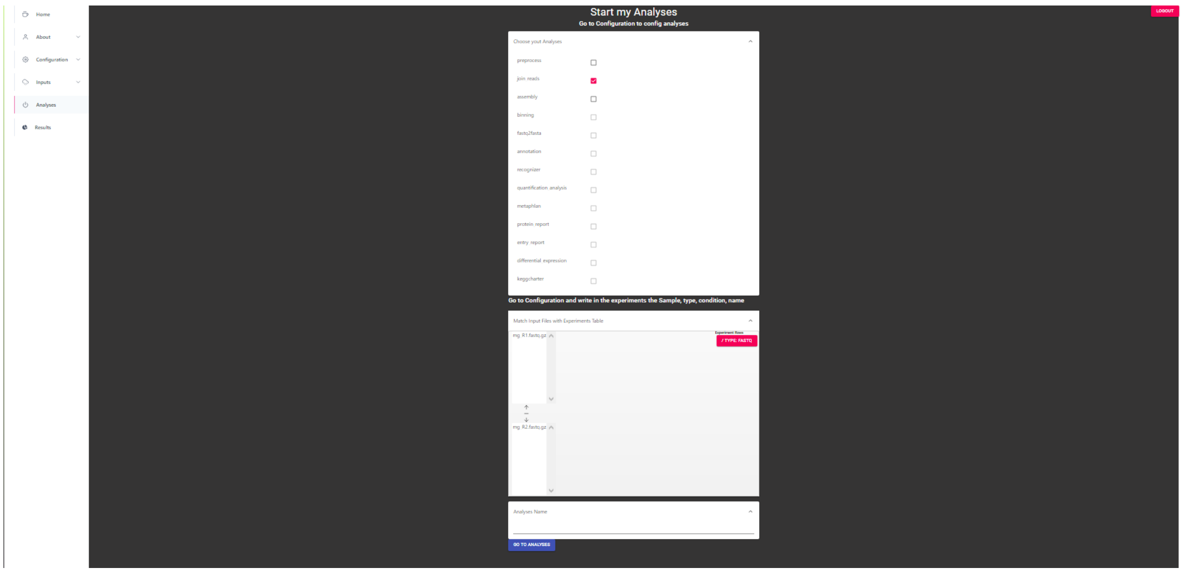
My Inputs

File Name	File Type	Data
mg_R1.fastq.gz	fastq	Tue, 20 Sep 2022 15:20:38 GMT
mg_R2.fastq.gz	fastq	Tue, 20 Sep 2022 15:22:48 GMT

Rows per page: 10 1-2 of 2

Figure 16: “My Inputs” page of MOSGUITO. Red buttons to download and delete files selected in the table, and a blue one to clear select files.

A user can start an analyses in MOSCA using ”Analyses” page. The page is divided into three major components, in the first one users can select the analyses to run, in the second the experiments table needs to be populated, using the information from the experiment files, for every row a input button will appear representing different inputs necessary’s for each analyses, clicking in the buttons files categorized by the type necessary for the analyses appear and the end user matches them to each of the rows, in the third step is to give a Name to the analyses, and send the analyses to the server by clicking the button send to analyses Fig.17.



Start my Analyses

Go to Configuration to config analyses

Choose yout Analyses

- [ ] preprocess
- [x] join reads
- [ ] assembly
- [ ] binning
- [ ] fastq2fasta
- [ ] annotation
- [ ] recognizer
- [ ] quantification analysis
- [ ] metaphlan
- [ ] protein report
- [ ] entry report
- [ ] differential expression
- [ ] keggcharter

Go to Configuration and write in the experiments the Sample, type, condition, name

Match Input Files with Experiments Table

mg R1.fastq.gz

mg R2.fastq.gz

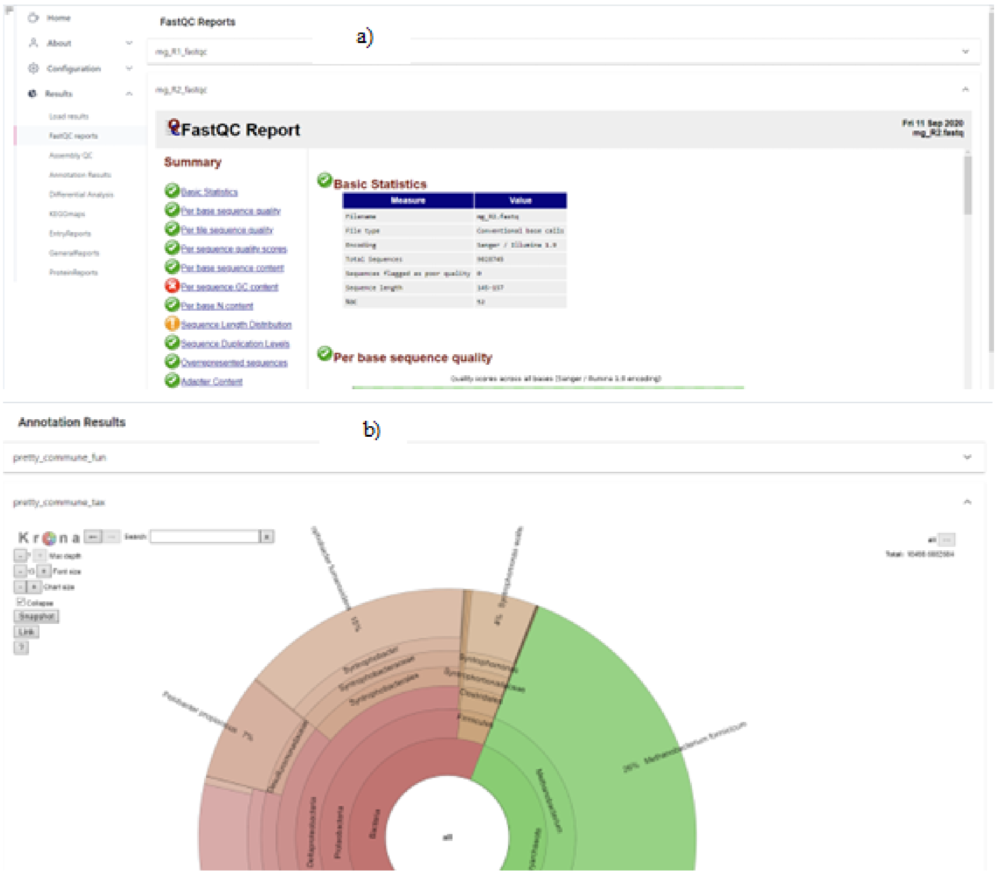
Experiment Rows

/ TYPE: FASTQ

Analyses Name

Figure 17: "Analyses" page of MOSGUITO is divided into three major steps one to select the analyses to realize, another to match the inputs files to the experiments dataset and another to give a name to the corresponding analyses.

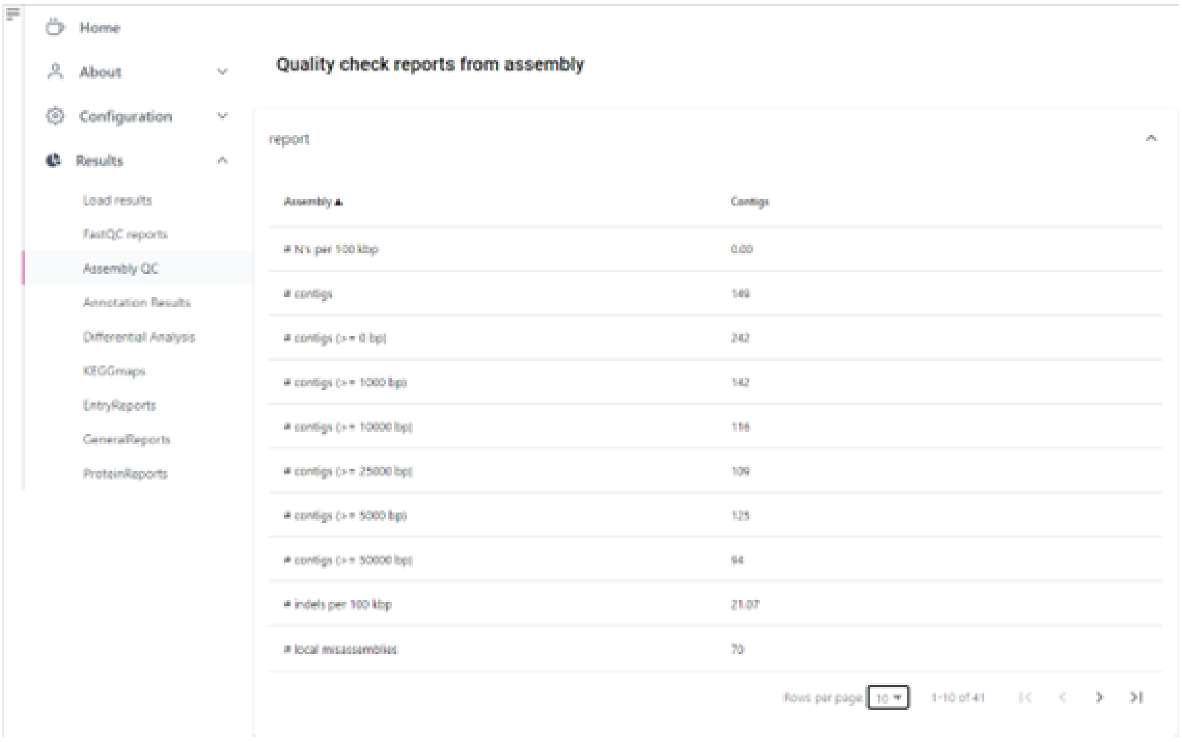
If the user needs to see the results from their analyses, going to the "Results" page they can see in a table format all the Analyses they submit in the server, sorted by date and containing "Analyses Name" and to other columns containing a button to download all the results produced by the server the file comes in ZIP format, and another column with a button to let users watch the results in the client-side, only a few files are displayed in the client-side, notably "FastQC Reports", "AssemblyQC", "Annotation Results", "Differential Analysis", "KEGGMaps", "EntryReports", "General Reports" and "Protein Reports". In the FastQC reports page and Annotation Results page for each different results you will have an "Accordion" with the names of the respective FastQC/Krona Plot of annotation Result, and if you click on them, you will see the FastQC with all the interactions the HTML of the FastQC/Krona Plot offer. Fig.18 shows a snapshot of the "Accordion" and the FastQC report/Krona Plot pages.



Measure	Value
Filename	mg_R2.fastq
File type	Conventional base calls
Encoding	Sanger / Illumina 1.9
Total Sequences	9828745
Sequences flagged as poor quality	0
Sequence length	145-157
%GC	52

Figure 18: In the section a) shows a snapshot of "FastQC reports" page of MOSGUITO. For each result will be a "Accordion" with the name of the report, and inside that "Accordion", the FastQC Report with all the interactions possible a FastQC have. In the section b) shows a snapshot of "Annotation Results" page, for each annotation result it will present an "Accordion" with the respective Krona Plot and all the interactions associated with it.

In the ”AssemblyQC” page a report on the quality of contigs obtained is presented, showing the results in a table, for each different quality contigs in the results an accordion with assembly table name will appear, selecting them will display the respective table on the page. Fig.19 shows a snapshot of the AssemblyQC page.



Quality check reports from assembly

report

Assembly ▲	Contigs
# N's per 100 kbp	0.00
# contigs	149
# contigs (>= 0 bp)	242
# contigs (>= 1000 bp)	142
# contigs (>= 10000 bp)	116
# contigs (>= 25000 bp)	109
# contigs (>= 5000 bp)	125
# contigs (>= 50000 bp)	94
# indels per 100 kbp	21.07
# local misassemblies	70

Rows per page: 10 1-10 of 41

Figure 19: “AssemblyQC” page of MOSGUITO. For each assembly table will be an “Accordion” with the name of the table, the table can be sorted by clicking in the column names. And in the rows for page, you can select how many rows per page you want to show.

In the ”Differential Analysis” page heatmaps on the differential analyses obtained is present, showing every Heatmap produced by the analyses, being displayed in the browser with their names. Fig.20 shows a snapshot of the Differential Analysis page.

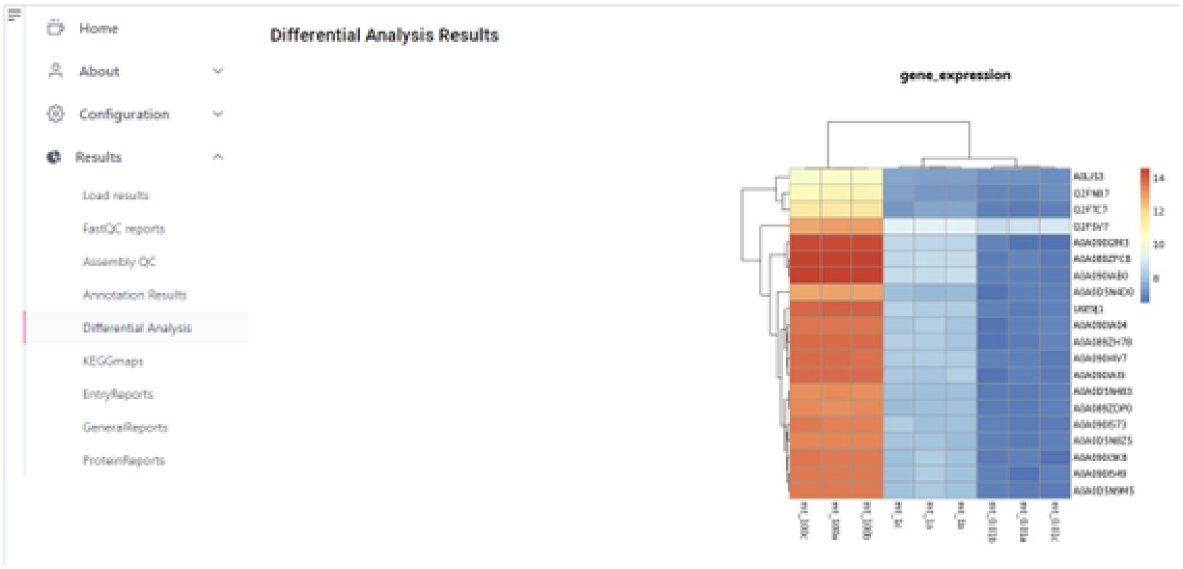
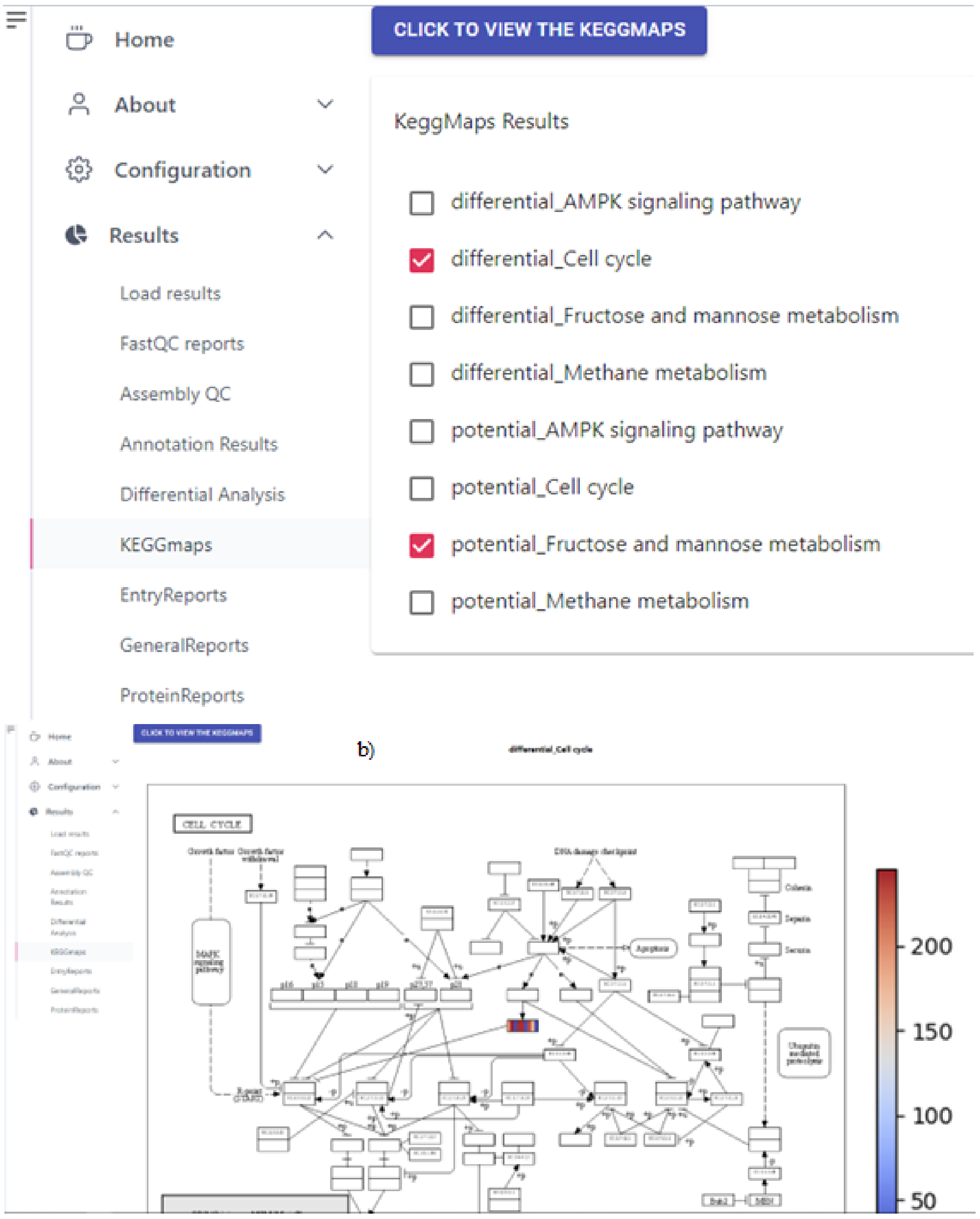


Figure 20: "Differential Analysis" page of MOSGUITO. For each Heatmap in the differential analysis of MOSCA, you will get the label and the corresponding heatmap, when you click on the image, it will stand out from the webpage.

In the "KEGGmaps" page a KEGGmap on the keggcharter obtained is present, showing every KEGGmap requested in presented in the configuration. MOSCA can produce more that 300 KEGGmaps, so to not overload the browser, the client-side is able only to show 6 KEGGmaps at a time, to do that in an initial state of page you have a accordion with all the KEGGmaps produced by your analyses with a checkbox. By selecting the checkbox and the button the selected KEGGmaps are displayed in the "page". Fig.21 shows a snapshot of KEGGmaps page.

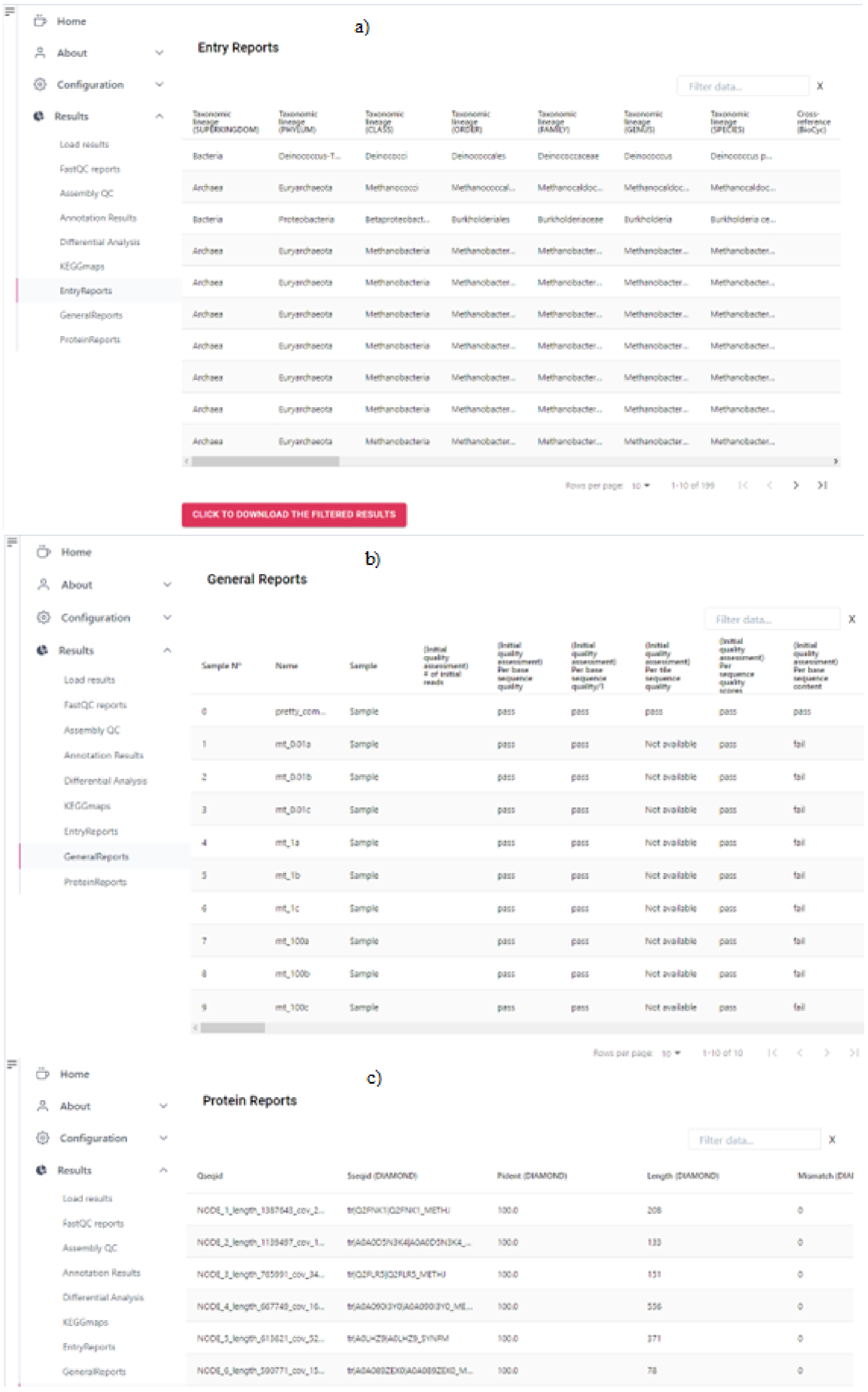


KeggMaps Results

- [ ] differential_AMPK signaling pathway
- [✓] differential_Cell cycle
- [ ] differential_Fructose and mannose metabolism
- [ ] differential_Methane metabolism
- [ ] potential_AMPK signaling pathway
- [ ] potential_Cell cycle
- [✓] potential_Fructose and mannose metabolism
- [ ] potential_Methane metabolism

Figure 21: "KEGGmaps" page of MOSGUITO. In the a) section a snapshot of an Accordion with all the KEGGmaps possible to visualize, you can choose a max of 6, other way it will show a warning message, click on the blue bottom to show the selected KEGGMaps. In the b) section a snapshot of the KEGGmaps results, the selected one, clicking on the image will have the image stand out from the website, the blue button will make the KEGGmaps page return to the a) section.

In the ″EntryReports″, ″GeneralReports″, ″ProteinReports″ pages tables on entry report and protein report respectively obtained is present, showing the respectable tables, containing the report information. Fig.22 shows a snapshot of all this pages.



Entry Reports

Taxonomic lineage (SUPERKINGDOM)	Taxonomic lineage (PHYLUM)	Taxonomic lineage (CLASS)	Taxonomic lineage (ORDER)	Taxonomic lineage (FAMILY)	Taxonomic lineage (GENUS)	Taxonomic lineage (SPECIES)	Cross-reference (BioCyc)
Bacteria	Deinococcus-T...	Deinococci	Deinococcales	Deinococcaceae	Deinococcus	Deinococcus p...	
Archaea	Euryarchaeota	Methanococci	Methanococcal...	Methanocaldoc...	Methanocaldoc...	Methanocaldoc...	
Bacteria	Proteobacteria	Betaproteobact...	Burkholderiales	Burkholderiaceae	Burkholderia	Burkholderia ce...	
Archaea	Euryarchaeota	Methanobacteria	Methanobacter...	Methanobacter...	Methanobacter...	Methanobacter...	
Archaea	Euryarchaeota	Methanobacteria	Methanobacter...	Methanobacter...	Methanobacter...	Methanobacter...	
Archaea	Euryarchaeota	Methanobacteria	Methanobacter...	Methanobacter...	Methanobacter...	Methanobacter...	
Archaea	Euryarchaeota	Methanobacteria	Methanobacter...	Methanobacter...	Methanobacter...	Methanobacter...	
Archaea	Euryarchaeota	Methanobacteria	Methanobacter...	Methanobacter...	Methanobacter...	Methanobacter...	
Archaea	Euryarchaeota	Methanobacteria	Methanobacter...	Methanobacter...	Methanobacter...	Methanobacter...	
Archaea	Euryarchaeota	Methanobacteria	Methanobacter...	Methanobacter...	Methanobacter...	Methanobacter...	

1-10 of 199

General Reports

Sample Nº	Name	Sample	(Initial quality assessment) # of initial reads	(Initial quality assessment) Per base sequence quality	(Initial quality assessment) Per base sequence quality/1	(Initial quality assessment) Per tile sequence quality	(Initial quality assessment) Per sequence quality scores	(Initial quality assessment) Per base sequence content
0	pretty_com...	Sample		pass	pass	pass	pass	pass
1	mt_0.01a	Sample		pass	pass	Not available	pass	fail
2	mt_0.01b	Sample		pass	pass	Not available	pass	fail
3	mt_0.01c	Sample		pass	pass	Not available	pass	fail
4	mt_1a	Sample		pass	pass	Not available	pass	fail
5	mt_1b	Sample		pass	pass	Not available	pass	fail
6	mt_1c	Sample		pass	pass	Not available	pass	fail
7	mt_100a	Sample		pass	pass	Not available	pass	fail
8	mt_100b	Sample		pass	pass	Not available	pass	fail
9	mt_100c	Sample		pass	pass	Not available	pass	fail

1-10 of 10

Protein Reports

Qseqid	Sseqid (DIAMOND)	Pident (DIAMOND)	Length (DIAMOND)	Mismatch (DIA
NODE_1_length_1387643_cov_2...	tr|Q2FNK1|Q2FNK1_METHJ	100.0	208	0
NODE_2_length_1139487_cov_1...	tr|A0A0D5N3K4|A0A0D5N3K4_...	100.0	133	0
NODE_3_length_765991_cov_34...	tr|Q2FLR5|Q2FLR5_METHJ	100.0	151	0
NODE_4_length_687749_cov_16...	tr|A0A090I3Y0|A0A090I3Y0_ME...	100.0	556	0
NODE_5_length_613621_cov_52...	tr|A0L4Z9|A0L4Z9_SYNFM	100.0	271	0
NODE_6_length_590771_cov_15...	tr|A0A089ZEX0|A0A089ZEX0_M...	100.0	78	0

Figure 22: In the section a) shows a snapshot of the “EntryReport” page, the page presents a search bar in the right top, the search bar presents a button (“x”) to clear the search bar faster, and in the bottom a red button to download the current state of the data in the table. In section b) shows a snapshot of the “GeneralReport” page, the page presents the table and the search bar. In section c) shows a snapshot of “ProteinReport” page, the pagent presents a search bar in the right top with a button (“x”) to clear the search bar, and in the bottom a red button to download the current state of data in the table.

Cause MOSCA/MOSGUITO is not yet implement in a physical server so to work with the current version of the web service, the suggested installation method for MOSCA is:

1. Install Python version 3.10 going to https://www.python.org, and all dependencies by typing:

    pip install Flask

    pip install flask-sqlalchemy

    pip install psycopg2

    pip install Flask-Cors

    pip install flask-restful

    pip install pyOpenSSL

    pip install PyJWT

    pip install Flask-JWT-Extended

    pip install requests

    pip install python-magic-bin

2. Install PostGresSQL using guide in https://www.postgresqltutorial.com/postgresql-getting-started/install-postgresql/, Open SQl shell, and type:

    CREATE DATABASE Teses

3. Download APIs and scripts from github using command-line:

    svn checkout https://github.com/JosePereira97/Tese/trunk/Full_Job/Back_end

4. Build DataBase inside PostgresSQL using:

    from app_BD import db

    python

    db create_all()

5. Downloading and installing MOSCA to a linux environment. Installation of MOSCA requires mamba, which can be installed with *conda install -c conda-forge mamba*. To install MOSCA from source code, run:

    svn checkout https://github.com/JosePereira97/MOSCA

    bash MOSCA/workflow/envs/install.bash

    conda activate mosca

    In Snakemake.main() we need to subtitute sys.exit() with return to be able to get responses from Snakemake.

6. Downloading client-side MOSGUITO by:

   svn checkout https://github.com/JosePereira97/MOSGUITO

7. Last Step consists in running all different application using different command line instances by:

   python app_BD.py

   python app.py

   mosca.py

   npm start

   Start testing the Web service using MOSGUITO. Because MOSGUITO only works remotely because the lack of a server, the application is only able to do one analysis at a time. With the implementation in a server using kubernetes, the server is able to do more analysis in the same time, using the replication of docker servers.

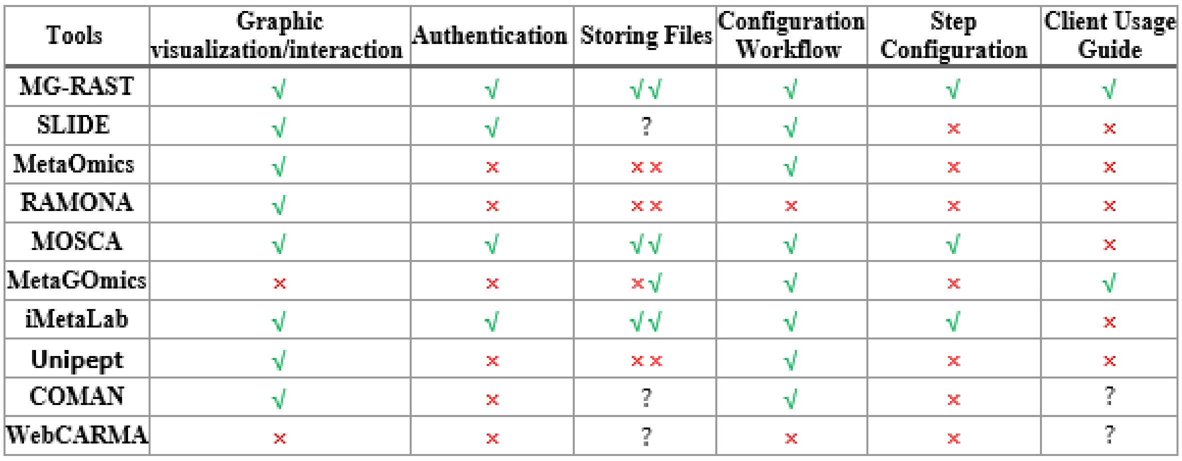
5

DISCUSSION OF THE RESULTS

MOSGUITO as a web service is divided four different components. Initially, the server MOSCA divided each step into different containers, all linked together, but using the Snakemake workflow, that was already implemented in MOSCA, it was possible to turn each step of MOSCA independent, without dividing them in different docker containers. MOSCA was constructed to be stored in docker containers inside a Kubernetes cluster, which is not yet implemented. Usually this kind of applications servers are stored inside physical servers, but is not yet the case for MOSCA. That way it was not possible to compare speed performances between different applications servers. To build APIs it was decided to use Flask cause it was easy to learn and utilize, but in complex APIs is harder cause it needs supplement packages to be able interact with other application like a Database. Our APIs use HTTP protocol to trade information between client and server, but usually this kind of applications use HTTPS protocol, because it's safer. But to use HTTPS protocol it is necessary to have credential servers, and because MOSCA is not implemented in a server there is no credential HTTPS protocol.

MOSGUITO/MOSCA is a unique application, being a web tool doing all meta-omics analysis, i.e., MG, MT and MP. Tab.4 shows a comparison of different features and usability in different meta-omics web tools. MG-RAST contains authentication methods, is able to store input and output files from users into a Database, end users can make their results public, users can download or visualize their results in the browser, it's an easy to use application with a lot of help in each step to start an analysis, and end users are able to configure their analysis. SLIDE contains authentication method cause is publicly available under BSD license, users can download and visualize their information, more information is unavailable about the interface cause is private, expensive and there is no more information available. In MetaOmics and RAMONA web services end users can visualize their results, only MetaOmics users can configure the Workflow, users are unable to Authenticate, store files in server cause the web service is a local one, users can only access client side by installing the application, unable to configure the steps of the pipeline, and client side does not contain any Usage guide. In MOSCA users can visualize outputs, store input and output files in server database, configure MOSCA workflow and steps, but lack in a User guide in the Client side for easy understanding. In MetaGOmics users can only download their results, only output files are stored and shared with end user by an URL scented to a specific email, users can Configure the workflow and client-side contains every information needed to help end users. In iMetaLab users can visualize outputs, store input and output files in server database, configure iMetaLab workflow and steps, but lack in a User guide in the Client side for easy understanding. In Unipept web services end users can visualize their results and configure their analysis, users are unable to Authenticate, store files in server cause the web service is a local one, users can only access client side by installing the application, unable to configure the steps of the pipeline, and client side does not contain any Usage guide. In COMAN web services end users can visualize their results and configure their analysis, users are unable to Authenticate, and configure the steps, and there in no information about the feature of storing files and client usage guide, COMAN client side is down in the time being so some information is unknown. In WebCARMA users can only download their results, client side does not contain authentication module, end users are unable to configure workflow and analysis steps, and there in no information about the feature of storing files and client usage guide, WebCARMA client side is down in the time being so some information is unknown. In the end, every web tool except MetaGOmics and WebCARMA let users visualize and interact with their results on the web. Only MG-RAST, SLIDE, MOSCA, and iMetaLab offers an Authentication module to end users. Only iMetaLab, MOSCA, and MG-RAST can store input and output files from end users, being MetaGOmics the only one able to store only Output files. Every web tool except RAMONA and WebCARMA let users configure the pipeline workflow. Only MG-RAST, MOSCA, and iMetaLab let end users select and configure the pipeline steps. Only MG-RAST and MetaGOmics web tools contain a guide built into the client side.

Table 4: Comparison of different web services for meta-omics analysis. Comparing features in the client-side of the application. Each column represents a different feature: Graphical visualization/interaction, if client-side of meta-omics tools can display results; Authentication, if client-side contains authentication module or not to protect end user data; Storing Files, if web service can store input and output files, 1° signal for inputs 2° signal for outputs, i.e., in MetaGOmics doesn't store input files, but stores output files; Configuration Workflow, if end-user are able to configure the workflow of meta-omics pipeline from the web; Step Configuration, if end-users can select each rule to run without previous rules; Client Usage Guide, if in the respective Client-Side exist hints to help end users work with the application. "?" represents tools that do not have the feature reported in the applications or that the client-sides are difficult to access.



Tools	Graphic visualization/interaction	Authentication	Storing Files	Configuration Workflow	Step Configuration	Client Usage Guide
MG-RAST	√	√	√√	√	√	√
SLIDE	√	√	?	√	×	×
MetaOmics	√	×	××	√	×	×
RAMONA	√	×	××	×	×	×
MOSCA	√	√	√√	√	√	×
MetaGOmics	×	×	×√	√	×	√
iMetaLab	√	√	√√	√	√	×
Unipept	√	×	××	√	×	×
COMAN	√	×	?	√	×	?
WebCARMA	×	×	?	×	×	?

6 CONCLUSIONS AND FUTURE PERSPECTIVES

MOSGUITO was developed as a web service, to automate the use of MOSCA. It uses Python Flask for the APIs and ReactJS for the client-side. Major improvements from the first implementation of MOSGUITO is that end users don't need to download MOSCA; because of that end users don't need to interact to command line interfaces to be able to customize and work with MOSCA. MOSCA can run any step without the need to run previous rules. Yet, there is space for other improvements. MOSCA as a server is not installed in a deploy server, and is not yet available to be used by the public in general. Also, MOSGUITO does not contain any online version connected with the Mother API. An important next step will be to host the web service in a physical server to be able to test and open the application to MOSGUITO. Implementation to reset passwords and retrieve their accounts is something that can be done. A queue system needs to be implemented if MOSCA is stored in a physical server, because of the time MOSCA needs to do a complete a run.

BIBLIOGRAPHY

Ankolekar, A., Burstein, M., Hobbs, J. R., Lassila, O., Martin, D., McDermott, D., McIlraith, S. A., Narayanan, S., Paolucci, M., Payne, T. et al. (2002), Daml-s: Web service description for the semantic web, *in* 'International semantic web conference', Springer, pp. 348–363.

Aslam, F. A., Mohammed, H. N., Mohd, J. M., Gulamgaus, M. A. and Lok, P. (2015), 'Efficient way of web development using python and flask', *International Journal of Advanced Research in Computer Science* **6**(2), 54–57.

Axios (n.d.).
URL: *https://axios-http.com/docs/api_intro*

Béja, O., Aravind, L., Koonin, E. V., Suzuki, M. T., Hadd, A., Nguyen, L. P., Jovanovich, S. B., Gates, C. M., Feldman, R. A., Spudich, J. L. et al. (2000), 'Bacterial rhodopsin: evidence for a new type of phototrophy in the sea', *Science* **289**(5486), 1902–1906.

Bolger, A. M., Lohse, M. and Usadel, B. (2014), 'Trimmomatic: a flexible trimmer for illumina sequence data', *Bioinformatics* **30**(15), 2114–2120.

Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S. and Winer, D. (2000), 'Simple object access protocol (soap) 1.1'.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E. and Yergeau, F. (1997), 'Extensible markup language (xml)', *World Wide Web Journal* **2**(4), 27–66.

Brown, J., Pirrung, M. and McCue, L. A. (2017), 'Fqc dashboard: integrates fastqc results into a web-based, interactive, and extensible fastq quality control tool', *Bioinformatics* **33**(19), 3137–3139.

Carvalho, B. S. and Irizarry, R. A. (2010), 'A framework for oligonucleotide microarray preprocessing', *Bioinformatics* **26**(19), 2363–2367.

Christensen, E., Curbera, F., Meredith, G., Weerawarana, S. et al. (2001), 'Web services description language (wsdl) 1.1'.

Cox, J., Hein, M. Y., Luber, C. A., Paron, I., Nagaraj, N. and Mann, M. (2014), 'Accurate proteome-wide label-free quantification by delayed normalization and maximal peptide ratio extraction, termed maxlfq', *Molecular & cellular proteomics* **13**(9), 2513–2526.

Cox, J., Neuhauser, N., Michalski, A., Scheltema, R. A., Olsen, J. V. and Mann, M. (2011), 'Andromeda: a peptide search engine integrated into the maxquant environment', *Journal of proteome research* **10**(4), 1794–1805.

Di Pierro, M. (2013), *web2py*, Lulu. com.

Dory, M., Parrish, A. and Berg, B. (2012), *Introduction to Tornado: Modern Web Applications with Python*, " O'Reilly Media, Inc.".

Farrell, S. (2009), 'Api keys to the kingdom', *IEEE Internet Computing* **13**(5), 91–93.

Fielding, R. T. (2000), *Architectural styles and the design of network-based software architectures*, University of California, Irvine.

Forcier, J., Bissex, P. and Chun, W. J. (2008), *Python web development with Django*, Addison-Wesley Professional.

Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A. and Stewart, L. (1999), Http authentication: Basic and digest access authentication, Technical report.

Gerlach, W., Jünemann, S., Tille, F., Goesmann, A. and Stoye, J. (2009), 'Webcarma: a web application for the functional and taxonomic classification of unassembled metagenomic reads', *BMC bioinformatics* **10**(1), 1–10.

Ghosh, S., Datta, A., Tan, K. and Choi, H. (2018), 'SLIDE – a web-based tool for interactive visualization of large-scale – omics data', *Bioinformatics* **35**(2), 346–348.
URL: *https://doi.org/10.1093/bioinformatics/bty534*

Gilbert, J. A., Field, D., Huang, Y., Edwards, R., Li, W., Gilna, P. and Joint, I. (2008), 'Detection of large numbers of novel sequences in the metatranscriptomes of complex marine microbial communities', *PloS one* **3**(8), e3042.

Hardt, D. et al. (2012), 'The oauth 2.0 authorization framework'.

Hellkamp, M. et al. (2016), 'Bottle: Python web framework', *URL: https://bottlepy. org* .

Kopylova, E., Noé, L. and Touzet, H. (2012), 'Sortmerna: fast and accurate filtering of ribosomal rnas in metatranscriptomic data', *Bioinformatics* **28**(24), 3211–3217.

Kumar, S. (2019), 'A review on client-server based applications and research opportunity', *International Journal of Scientific Research* **10**, 33857–33862.

Langmead, B., Trapnell, C., Pop, M. and Salzberg, S. L. (2009), 'Ultrafast and memory-efficient alignment of short dna sequences to the human genome', *Genome biology* **10**(3), 1–10.

Li, D., Luo, R., Liu, C.-M., Leung, C.-M., Ting, H.-F., Sadakane, K., Yamashita, H. and Lam, T.-W. (2016), 'Megahit v1. 0: a fast and scalable metagenome assembler driven by advanced methodologies and community practices', *Methods* **102**, 3–11.

Liao, B., Ning, Z., Cheng, K., Zhang, X., Li, L., Mayne, J. and Figeys, D. (2018), 'iMetaLab 1.0: a web platform for metaproteomics data analysis', *Bioinformatics* **34**(22), 3954–3956.
URL: *https://doi.org/10.1093/bioinformatics/bty466*

Lin, S. M., Du, P., Huber, W. and Kibbe, W. A. (2008), 'Model-based variance-stabilizing transformation for illumina microarray data', *Nucleic acids research* **36**(2), e11–e11.

Love, M., Anders, S. and Huber, W. (2014), 'Differential analysis of count data–the deseq2 package', *Genome Biol* **15**(550), 10–1186.

Ma, T., Huo, Z., Kuo, A., Zhu, L., Fang, Z., Zeng, X., Lin, C.-W., Liu, S., Wang, L., Liu, P., Rahman, T., Chang, L.-C., Kim, S., Li, J., Park, Y., Song, C., Oesterreich, S., Sibille, E. and Tseng, G. C. (2018), 'MetaOmics: analysis pipeline and browser-based software suite for transcriptomic meta-analysis', *Bioinformatics* **35**(9), 1597–1599.
URL: *https://doi.org/10.1093/bioinformatics/bty825*

Maza, E. (2016), 'In papyro comparison of tmm (edger), rle (deseq2), and mrn normalization methods for a simple two-conditions-without-replicates rna-seq experimental design', *Frontiers in genetics* **7**, 164.

McDonough, C. (2021), 'The pyramid web framework'.

Membrey, P., Plugge, E., Hawkins, T. and Hawkins, D. (2010), *The definitive guide to MongoDB: the noSQL database for cloud and desktop computing*, Springer.

Merkel, D. (2014), 'Docker: lightweight linux containers for consistent development and deployment', *Linux journal* **2014**(239), 2.

Mesuere, B., Willems, T., Van der Jeugt, F., Devreese, B., Vandamme, P. and Dawyndt, P. (2016), 'Unipept web services for metaproteomics analysis', *Bioinformatics* **32**(11), 1746–1748.
URL: *https://doi.org/10.1093/bioinformatics/btw039*

Mikheenko, A., Saveliev, V. and Gurevich, A. (2016), 'Metaquast: evaluation of metagenome assemblies', *Bioinformatics* **32**(7), 1088–1090.

Mölder, F., Jablonski, K. P., Letcher, B., Hall, M. B., Tomkins-tinch, C. H., Sochat, V., Forster, J., Lee, S., Twardziok, S. O., Kanitz, A., Wilm, A., Holtgrewe, M., Rahmann, S., Nahnsen, S. and Köster, J. (2021), 'Sustainable data analysis with Snakemake [version 1 ; peer review : 1 approved , 1 approved with reservations]', *F1000Research* (May), 1–25.

Myers, J. and Copeland, R. (2015), *Essential SQLAlchemy: Mapping Python to Databases, "* O'Reilly Media, Inc.".

Ni, Y., Li, J. and Panagiotou, G. (2016), 'Coman: a web server for comprehensive metatranscriptomics analysis', *BMC genomics* **17**(1), 1–7.

Nordic, A. (n.d.), 'Developing the api mindset: a guide to using private, partner, & public apis (2015)'.

Nurk, S., Meleshko, D., Korobeynikov, A. and Pevzner, P. A. (2017), 'metaspades: a new versatile metagenomic assembler', *Genome research* **27**(5), 824–834.

Obe, R. O. and Hsu, L. S. (2017), *PostgreSQL: Up and Running: a Practical Guide to the Advanced Open Source Database, "* O'Reilly Media, Inc.".

Ong, S. P., Cholia, S., Jain, A., Brafman, M., Gunter, D., Ceder, G. and Persson, K. A. (2015), 'The materials application programming interface (api): A simple, flexible and efficient api for materials data based on representational state transfer (rest) principles', *Computational Materials Science* **97**, 209–215.

ReactJS.org, ' ReactJS official'. [Online] (2022).
URL: *https://reactjs.org*

Requests lib (n.d.), https://docs.python-requests.org/en/latest/. Accessed: 2022-01-21.

Rho, M., Tang, H. and Ye, Y. (2010), 'Fraggenescan: predicting genes in short and error-prone reads', *Nucleic acids research* **38**(20), e191–e191.

Richards, R. (2006), Universal description, discovery, and integration (uddi), *in* 'Pro PHP XML and Web Services', Springer, pp. 751–780.

Riffle, M., May, D. H., Timmins-Schiffman, E., Mikan, M. P., Jaschob, D., Noble, W. S. and Nunn, B. L. (2018), 'Metagomics: A web-based tool for peptide-centric functional and taxonomic analysis of metaproteomics data', *Proteomes* **6**(1).
URL: *https://www.mdpi.com/2227-7382/6/1/2*

Sakimura, N., Bradley, J., Jones, M., De Medeiros, B. and Mortimore, C. (2014), 'Openid connect core 1.0', *The OpenID Foundation* p. S3.

Sass, S., Buettner, F., Mueller, N. S. and Theis, F. J. (2015), 'Ramona: a web application for gene set analysis on multilevel omics data', *Bioinformatics* **31**(1), 128–130.

Sleator, R. D., Shortall, C. and Hill, C. (2008), 'Metagenomics', *Letters in applied microbiology* **47**(5), 361–366.

Software Architecture: One-tier, Two-tier, Three Tier, N Tier (n.d.).
URL: *https://www.softwaretestingmaterial.com/software-architecture/*

Svehring, S. (2021), 'My sql bible'.

Tesliuk, A., Bobkov, S., Ilyin, V., Novikov, A., Poyda, A. and Velikhov, V. (2019), Kubernetes container orchestration as a framework for flexible and effective scientific data analysis, *in* '2019 Ivannikov Ispras Open Conference (ISPRAS)', pp. 67–71.

Truong, D. T., Franzosa, E. A., Tickle, T. L., Scholz, M., Weingart, G., Pasolli, E., Tett, A., Huttenhower, C. and Segata, N. (2015), 'Metaphlan2 for enhanced metagenomic taxonomic profiling', *Nature methods* **12**(10), 902–903.

Wang, Q., Shrestha, D. L., Robertson, D. and Pokhrel, P. (2012), 'A log-sinh transformation for data normalization and variance stabilization', *Water Resources Research* **48**(5).

werkzeug (n.d.).
URL: *https://werkzeug.palletsprojects.com/en/2.2.x/utils/*

Wilke, A., Bischof, J., Harrison, T., Brettin, T., D'Souza, M., Gerlach, W., Matthews, H., Paczian, T., Wilkening, J., Glass, E. M., Desai, N. and Meyer, F. (2015), 'A restful api for accessing microbial community data for mg-rast', *PLoS computational biology* **11**(1), e1004008.
URL: *https://europepmc.org/articles/PMC4287624*

Wilmes, P., Heintz-Buschart, A. and Bond, P. L. (2015), 'A decade of metaproteomics: where we stand and what the future holds', *Proteomics* **15**(20), 3409–3417.

Wu, Y.-W., Simmons, B. A. and Singer, S. W. (2016), 'Maxbin 2.0: an automated binning algorithm to recover genomes from multiple metagenomic datasets', *Bioinformatics* **32**(4), 605–607.

Zhang, X., Song, X., Wang, H. and Zhang, H. (2008), 'Sequential local least squares imputation estimating missing value of microarray data', *Computers in biology and medicine* **38**(10), 1112–1120.