



**Universiteit
Leiden**
The Netherlands

Bioinformatics

Speeding up the multiscale access in the
MycoDiversity DataBase

Haike van Thiel

Supervisors:

Irene Martorelli, Fons Verbeek & Richard van Dijk

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)

www.liacs.leidenuniv.nl

01/02/2022

Acknowledgement

Irene Martorelli and Aram Pooryousefi have provided an important foundation for this thesis. My gratitude goes out to Irene Martorelli, Fons Verbeek and Richard van Dijk for supervising this project. In addition, I would like to thank Jeremy Gobeil for his advise and additional support along the way.

Abstract

There are many species of fungi in the world. Some of them have essential roles in ecosystems. The amount of DNA barcoding information retrieved from fungi has increased exponentially over the last decade. The data is stored in online archives and they could be a rich source of information. Some researchers are very interested in how different fungi grow under different environmental conditions. The MycoDiversity DataBase was introduced to process the raw archive data and enable users to investigate Fungi in different environments. This thesis presents new methods for the MycoDiversity DataBase that speed up the multiscale access. The main approach for that was implementing a hierarchical tree based filter. In addition, this thesis introduces a new visualization method, a map, as well.

Contents

1	Introduction	1
1.1	Contributions	1
1.2	Thesis overview	1
2	Background	3
2.1	MDDB fungal biodiversity data	3
2.2	Database types	3
2.3	MDDB database	3
3	Methods	5
3.1	Software	5
3.1.1	MonetDB	5
3.1.2	ODBC	5
3.1.3	PHP	5
3.1.4	HTML5	5
3.1.5	CSS	5
3.1.6	UIKit	6
3.1.7	JavaScript	6
3.1.8	JQuery	6
3.1.9	Leaflet	6
3.2	MVC	6
3.3	Style conventions	7
3.4	SQL Queries	8
4	Implementation	9
4.1	Filter	10
4.1.1	Location/taxonomy	10
4.1.2	Environment	11
4.1.3	Filter button	12
4.2	Tabs	13
4.3	Results	14
4.3.1	Table	14
4.3.2	Map	15
5	Results	16
5.1	Use cases	16
5.1.1	Use case 1	16
5.1.2	Use case 2	18
5.1.3	Use case 3	19
5.2	Validation	21
5.3	Performance	22
6	Conclusions	25

7	Further Research	26
7.1	Other pages	26
7.2	More samples	26
7.3	Download	26
7.4	Efficiency	26
7.5	Other db	26
7.6	More tabs	26
7.7	More filter options	26
	References	28
A	Appendix use cases	29
B	Appendix page components	30

1 Introduction

Fungi are one of the most abundant organisms in the world. There are millions of different species with many different functions. Some live in symbiosis with plants, others break down organic material and some are pathogenic and dangerous to our health. In other words, they have an important role in many ecosystems [1]. The variation is one of the aspects which make fungi very interesting to research. There still is a lot unknown about them. Large quantities of fungal data are collected constantly and the results are stored online in biological archives e.g. National Center for Biotechnology Information (NCBI) [2]. The problem is, even though the storages are very rich in information, researchers might not be able to use the information directly.

The MycoDiversity DataBase (MDDB) was created to solve this issue [3]. It needs a proper interface in order for researchers to be able to work with it. At the start of this thesis there already existed some code, but it needed some changes to make processes more efficient. There was a long list of requirements which needed to be implemented as well. The thesis is a smaller project that focuses on the biodiversity search of the MDDB. It builds on a larger project, which is made with a content management system (CMS). The main page, the headers, footers, menu's etc are all defined using the CMS. The biodiversity search program will be loaded as an iframe within the Mycodiversity project.

Research aim: *This project aims to design and test algorithmic approaches for MDDB with relation to data retrieval over different scales. The efficiency of the different algorithms will be assessed and the best performing algorithms will be incorporated in the UI of the database. In addition the visualization of fungal biodiversity data will be investigated and implemented.*

1.1 Contributions

This thesis will deliver the following items

- software description and implementation for algorithmic approaches and assessment to filter the MDDB and visualize the results in a table and on a map
- Three use cases and their results
- evaluation and performance methods

1.2 Thesis overview

In this section the structure of this thesis is outlined.

This chapter 1 contains the introduction as well as the problem statement and the research aim. Chapter 2 provides some background information for the methods of this research. It will explain how the data is obtained. It will also state how the data is stored in the database and how the database was designed. Then it will go into more detail how the database was structured. And lastly it will explain which parts of the database will be used for this project.

Chapter 3 provides a description of the methods. It will give an overview of the software used. On top of that it shows the code structure and the interface structure.

Chapter 4 shows the implementation design and description

Chapter 5 gives an overview of the results alongside a discussion of these results.

Chapter 6 presents the conclusions this research provided. It will explain whether the aim is achieved.

Chapter 7 will give some future prospects as this project is just the tip of the iceberg.

2 Background

2.1 MDDB fungal biodiversity data

Fungi can be found all over the world and there is a lot of diversity between them. With the DNA barcoding technique their DNA can be sequenced. Metabarcoding focuses on identifying the compositions of species within a sample. This technique led to an estimation of 5.1 million species of fungi [4]. High throughput sequencing (HTS) technologies are used to study the fungal diversity on different scales. Some focus on small scales like country level or regional level, some on global scales and others look at environmental influences or relationships among fungal organisms. These HTS methods generate a lot of data which is often stored in a repository. To name a few: European Nucleotide Archive (ENA), NCBI, Sequence Read Archive (SRA) and the DNA DataBank of Japan Sequence Read Archive (DRA). The data is then obtained and stored in the MDDB using SRA mapping techniques and the NCBI SRA Toolkit [5].

2.2 Database types

The traditional way of storing records is through the use of a row oriented database (R-DB). A different database architecture was introduced in 2005 [6]. This database is called a column oriented database (C-DB). In a R-DB the data is organized by record whereas a C-DB organizes data by field. A small note: in order for the C-DB to have advantage, the database needs to be stored on separate disks. Writing in a R-DB is very easy as a new record will simply be added behind the last record in the database. In a C-DB there has to be a check on which disk the data actually needs to be stored. This means that for writing data the R-DB has an edge over the C-DB. The opposite occurs when reading data. If data is needed from one or multiple rows the R-DB is still fast. The problem is that for a lot of projects, including this one, aggregation is needed. This causes the process to be slower as additional data columns are added to the memory. The amount of disks the R-DB has to access is often higher. The C-DB structure allows to minimize the number of disks that need to be accessed. Also, the number of additional data columns that are put in the memory is kept as small as possible. This thesis mainly concerns itself with reading data. That is why at least for now, the logical choice was to use a column store database. [3] The original MDDB project compared two R-DBs with a C-DB. the C-DB had a much lower computation time. The included R-DBs were MySQL and SQLite. The C-DB that was used to compare was MonetDB [7].

2.3 MDDB database

The MDDB is designed with 20 tables, which can be divided in five main components: Study, Literature, Taxonomy, Sequence and Location. This is indicated in the UML diagram in figure 1. Data obtained from the NCBI is stored in the Literature and Study components. The processed HTS data are contained in the Sequence component. The Location and Taxonomy components give extra context. The Taxonomy labels the DNA sequence data from the Sequence component and the Location assists the geographical aspects of the Study and Literature. The location is associated to the study which is linked to the sequence data. The latter is also connected with the taxonomy that permits assigning scientific names belonging to fungi.

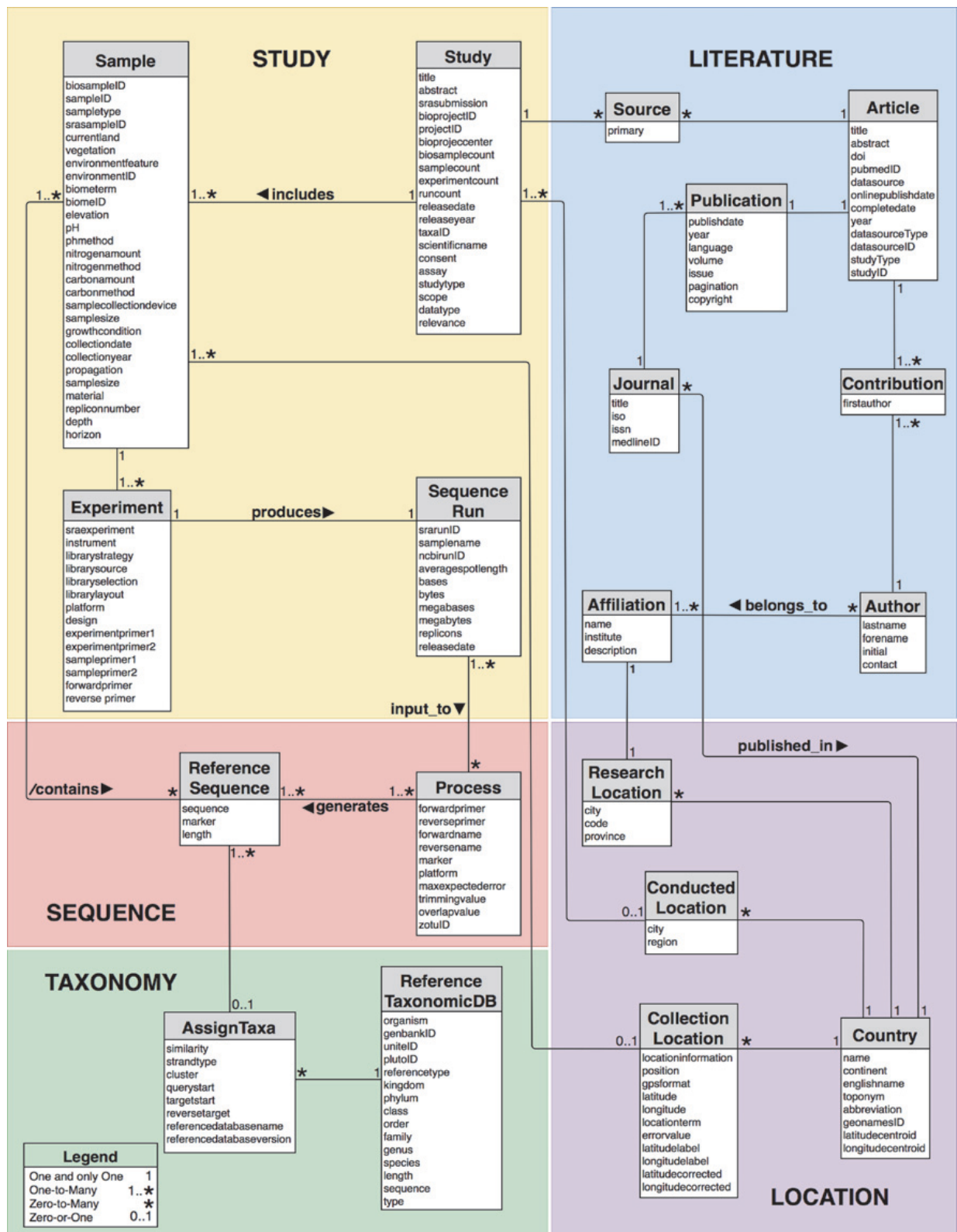


Figure 1: MDDB UML diagram, source:[3]

3 Methods

3.1 Software

This section gives an overview of all the software used with a little bit of background information and why they are relevant for this thesis.

3.1.1 MonetDB

The database-management system (DBMs) used is MonetDB. As mentioned before, MonetDB is column-oriented. It is an open source DBMS. It was specifically developed to handle large quantities of data. There were two versions of the MDDb, one in MonetDB and one in MySQL. MonetDB is used because it retrieves the data faster than MySQL.

3.1.2 ODBC

ODBC stands for Open Database Connectivity and it is produced by Microsoft and Simba Technologies [8]. It is an interface that allows access to data in a DBMS by an application. For this project it connects the server with the MonetDB.

3.1.3 PHP

The scripting language on the back end is Hypertext Preprocessor better known as PHP. It is a language which can be embedded into HTML and is very suitable for web development. Rasmus Lerdorf is the original creator of PHP who released it in 1995[9]. The latest version is PHP 8.0 released June 8th of 2020. The server the project currently runs on PHP 7.4. PHP is still a very dominant language for web development. For the websites whose server-side language is public, it is used in roughly 79.2% of them [10]. PHP has a large community and therefore provides a lot of support. There are other options besides PHP, Python for example. Both have their advantages and their disadvantages. The code made before the start of the thesis was all in PHP and that gave it the edge over Python.

3.1.4 HTML5

HTML is an abbreviation for Hyper Text Markup Language and was first released in 2008 [11]. It is a standard markup language and it is used to describe a webpage's structure. It does so by the use of tags like "paragraph" [12]. The HTML is responsible for the webpages structure.

3.1.5 CSS

CSS is short for Cascading Style Sheet and can be used for formatting the layout of a webpage. This includes colors, textstyles, borders and many more aspects. The first standard for CSS was published in 1996 and was developed by Håkon Wium Lie [13]. CSS is used to make the layout appealing to users and to help them understand parts by giving visual feedback.

3.1.6 UIKit

Currently version 3. The UIKit is a package which can be integrated in HTML code and is responsible for different parts of the layout [14]. UIKit components can also be manipulated through JavaScript if necessary. The UIKit is used to make the page more appealing for the user and to give it a more professional look.

3.1.7 JavaScript

The first release of JavaScript was in 1996 [15]. It is a very popular language used by an estimated 71.5% of professional developers. JavaScript is a scripting language that can be used to create intricate features for a webpage. It is supported by most web browsers and has many libraries. The language is used to manipulate elements, to define button actions and in some cases to make new HTML code.

3.1.8 JQuery

JQuery is a JavaScript library with multiple uses. In 2006 John Resig created it based on Dean Edwards' earlier cssQuery library [16]. It provides an API that handles things like Asynchronous JavaScript and XML (AJAX), event handling and HTML document manipulation much simpler. JQuery is used for the AJAX calls and to manipulate certain variables.

3.1.9 Leaflet

Leaflet is an open-source JavaScript library to make interactive maps [17]. The maps are based on the OpenStreetMap maps. It was originally created by Volodymyr Agafonkin. Leaflet was first released back in 2011. The library is used to make the map and to present results in the form of markers with InfoWindows. An alternative to make interactive maps is the Google maps API. Both are easy to work with and provide solid maps. The Leaflet maps however seems to load faster. Another disadvantage of google maps is that they will charge when using more than 25.000 page loads a day.[18]. As the project grows and demands more and more loads, a larger number might be necessary.

3.2 MVC

The code was built on existing work. One could divide code into different components. These are parts which handle the interaction with the database, parts which handle user requests and parts which deal with what the user gets to see. In the old code these parts could be found mixed in single files. This works but in the long run it can be difficult to work with. That is why the new code is structured with the Model-View-Controller (MVC) philosophy. This separates the three components and makes it more clear what parts of the code are responsible for certain actions. The MVC design pattern is used by many well known frameworks like Django (a python framework) and Laravel (a PHP framework). These known frameworks do not support MonetDB unfortunately and existing integrations were deprecated. In addition, they seemed to slow the program down. The models folder contains a database file. This file contains a function which connects to the database. The other model files extend on the class with this function to execute SQL queries. The

other model files have functions in which the SQL query parts which do not change are defined and they can be adjusted with parameters which it receives from the controller.

The controllers take the information from the user, often information sent with a POST request. It uses this information to retrieve information from a model and it either calls a PHP function which makes a view or it sends the information back in Json format. In the latter case, the user's view is updated with JavaScript.

A view is what the user sees and interacts with. This can be made with PHP when for example a table has to be made. Sometimes it is necessary to make the view with JavaScript for example when making a map.

3.3 Style conventions

Using style conventions is important to keep the code consistent and easy to work with. For the naming conventions, the CodeIgniter was followed [19]. the most important aspects include:

- File: A class file name starts with a capital letter. Other files are in all lower letters. In both cases words separated with an underscore.
- Class: has the same name as its file.
- Variable: lower letters with words separated with an underscore describing the variable.
- Functions: lower letters with words separated with an underscore describing the function.

The main file structure is as follows:

- js
 - all js files
- css
 - all css files
- src
 - controllers
 - * all controller files
 - models
 - * all model files
 - views
 - * all view files
- index.php

3.4 SQL Queries

The biodiversity search part of the research uses five tables directly. The association of the tables are explicitly important to provide information regarding the sequences, where they have been collected and observed and to what taxonomic names they belong to or something more simplistic but in line with this. Some queries are done in one single table like getting the continent or country. The main outline of the queries are predefined. These parameters can be determined by a user who clicks on a specific filter e.g. a country.

The tables used for these single table queries are “Sample” and “RefTaxonomicDB” entities.

A basic query would be: `SELECT $field FROM $table WHERE $condition`. Most queries are based on the filter and need to combine five tables (Sample, Contain, RefTaxonomicDB, RefSequence and AssignTaxa). The queries only contain AND operations in the WHERE section of the query and not OR operations.

A basic query would be: `SELECT $fields FROM "Sample" SM, "RefSequence" RS, "Contain" RSM, "RefTaxonomicDB" URS, "AssignTaxa" AT WHERE 1 = 1 AND URS.Refsequence_taxonomic_pk = AT.Refsequence_taxonomic_pk AND AT.refsequence_pk = RS.refsequence_pk AND RS.refsequence_pk = RSM.refsequence_pk AND RSM.sample_pk = SM.sample_pk AND $condition`

SELECT	FROM	WHERE
\$field	"Sample" / "RefTaxonomicDB"	1 = 1 \$field = \$condition

Table 1: Query template for one table

SELECT	FROM	WHERE
\$fields	"Sample" SM "RefSequence" RS "Contain" RSM "RefTaxonomicDB" URS "AssignTaxa" AT	1 = 1 URS.Refsequence_taxonomic_pk = AT.Refsequence_taxonomic_pk AT.refsequence_pk = RS.refsequence_pk AT.refsequence_pk = RS.refsequence_pk RS.refsequence_pk = RSM.refsequence_pk RSM.sample_pk = SM.sample_pk \$condition

Table 2: Query template for multiple tables

4 Implementation

This section contains flowcharts to illustrate the dynamics of the program. The program could have one large flowchart, but it was split up into smaller components to make the separate components more understandable. The interface consists of three main components. The filter component, the tab navigation and the result display. After interacting with one of the three, the user can immediately make a new decision meaning it is a continuous loop. Table 3 is the legend with all the symbols a flowchart may contain with a description. The grey circle symbols contain one or more letters. They illustrate a connection point between two figures and the letter has something to do with the connected figure. For example, in figure 3 the ‘F’ under ‘interact with filter’ is connected with the figure with the next steps when interacting with a [F]ilter. An important note is that the flowchart is not necessarily sequential. At all times a user can switch to a different action. The charts merely represent how a complete action would look like.








symbol	meaning
	decision
	process either by program or user
	start
	display on screen
	continue next figure
	database request
	indicates flow direction

Table 3: Legend with all possible flowchart symbols

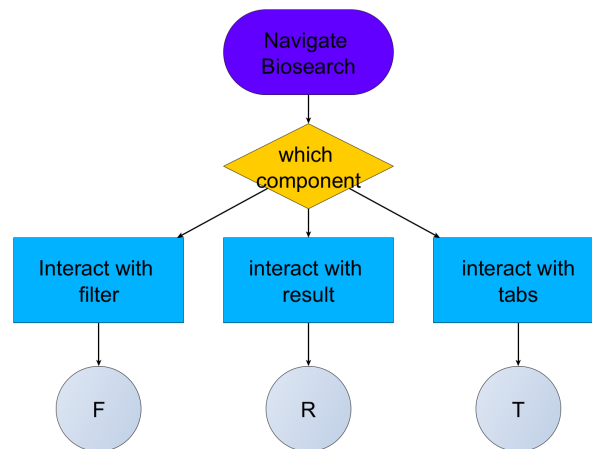


Figure 2: Main choices to navigate the MDDb biosearch when starting the page: **F**ilter [F], **R**esult [R] and **T**abs [T].

4.1 Filter

The filter is one of the most important components of this project. It allows a researcher to only extract the information they need. It has three main categories which can be seen in figure 3. The location, the taxonomy and the environment. All three of them are accordion elements to keep the page as clean as possible. When the page is loaded all three of them are closed. The main structure of the location and the taxonomy are the same. Both are hierarchical tree filters. For each of them there is the choice to either expand the tree to see a new layer or to add an option. The environment is a little different. This part provides an option to choose some biome terms or to decide the pH range.

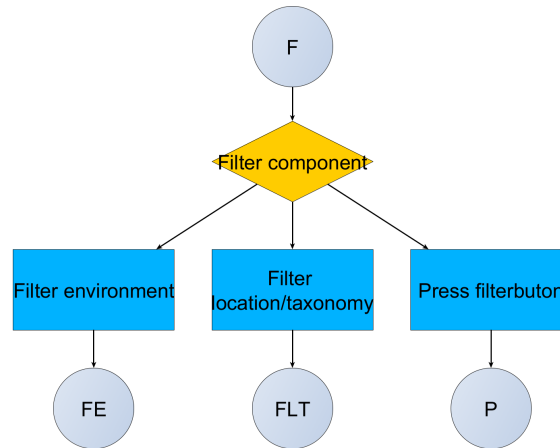


Figure 3: Flowchart for **F**ilter [F] choices: **F**ilter **E**nvironment [FE], **F**ilter **L**ocation/**T**axonomy [FLT] and **P**ress filter button [P]

4.1.1 Location/taxonomy

The flowcharts for the location and the taxonomy are very similar. The location and the taxonomy are combined in figure 4 because of similarities. The only difference being that the taxonomy has more layers. A layer is a subclass, for example the location has the layers: continent, subcontinent, country, sample. Taxonomy has: phylum, class, order, family, genus and species. There are multiple actions one can take. It first looks if the component is visible. If it is not someone has to open it first. If it is opened for the first time the first layer is made and displayed. If the first layer has been shown before it simply shows the content made before. Someone could expand on the interface which sends a signal to the controller which asks information from the database. The data is then processed and displayed on the screen. Each layer has two buttons. One which will expand the layer and one that allows a user to select the value. When expanding, these buttons are made with JavaScript. A selected value will appear at the top of the location or the taxonomy component as a checked value. Clicking on this checkbox will remove the selected value. Both the location and the taxonomy also include an autocomplete search. Navigating through all of the layers can be helpful and insightful because one can see the hierarchy but some users already know what value they want. In their cases it might be frustrating having to go through the layers over and over again. That is one of the reasons the autocomplete search was added. A user types in at least three

characters which will be send to the database and suggestions with these characters will be send back. Three characters are required because otherwise it would be too straining for the program causing it to load endlessly.

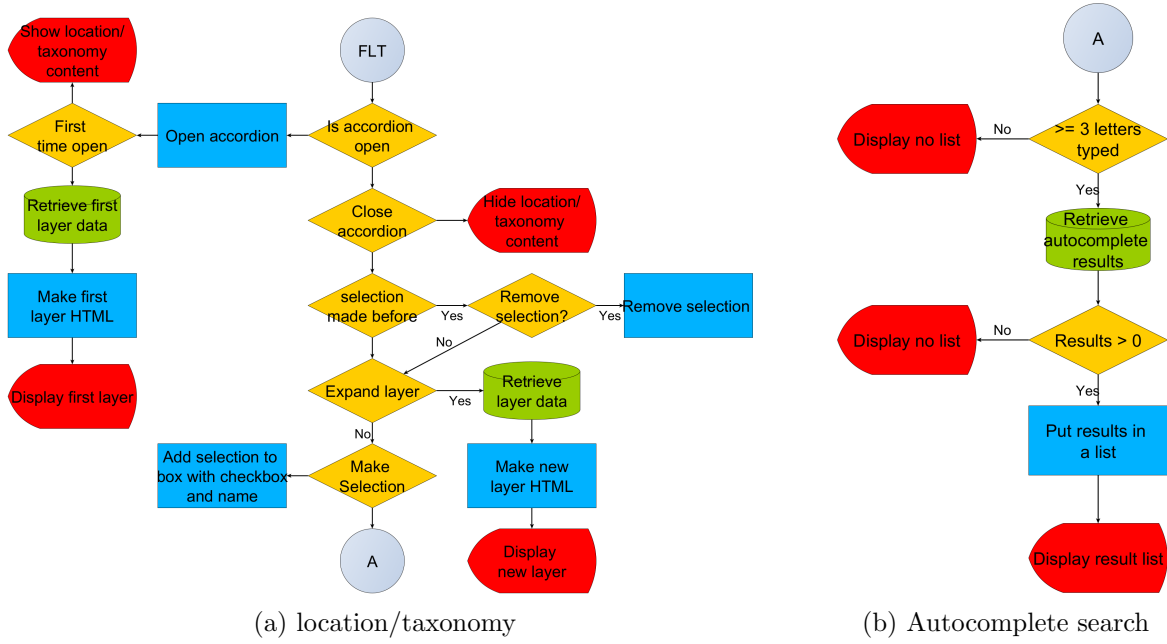


Figure 4: (a) Flowchart for **F**ilter **L**ocation and **T**axonomy [FLT]. (b) Shows how the **A**utocomplete search [A] dynamics works with a flowchart

4.1.2 Environment

The environment in figure 5 is a little different from the location and the taxonomy. Instead of navigating through layers, the environment provides both terms used to describe the contextual aspect, by means of biome terms and measurements, such as pH range that is used for navigating acidity values of the environmental sample. Like with the location and taxonomy, it first looks if the component is open in the first place. If it is not, the user can open it and like earlier it looks if it is opened for the first time. The biome terms are put in a list of terms with checkboxes. The pH is a numerical value which can be controlled through a range slider. A user can choose to slide the min and max value or to type the values in directly. If a user makes a mistake by having a minimum higher than the maximum value this will be corrected when someone presses the filter button.

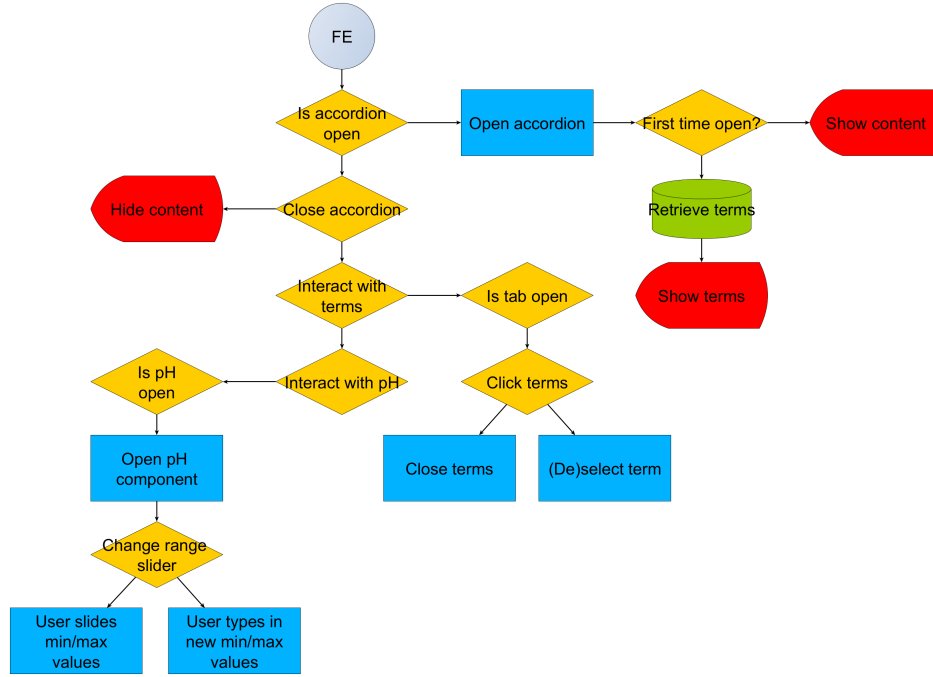


Figure 5: Flowchart for **F**ilter **E**nvironment [FE]

4.1.3 Filter button

When pressing the filter button all the filter information is extracted as explained in figure 6. A check is performed to make sure the values in the sliders are correct. Then the JavaScript code will look at which tab is active as it has to call a function based on that information. It will call either the function which makes a map or a function which makes a table. Both make a POST request to the server and provide the filter information. The request is received by their corresponding controllers. A query with all the filter information is made using the function in the filter class. This query is then handed to either the model for the map or for the table. Each demands different information from the database. The table is limited by the amount of records per page as that defines the table size. Visualising the data is different for the table and for the map. The table uses another PHP file in which the table is made. The controller sends the results from the model to this file and sends the result back to the JavaScript file. This result is HTML code and in JavaScript the result is appended in the right place on the page. The map controller sends the model result directly back to the JavaScript file in Json format. The map itself is made in Javascript using the Leaflet package. If there has been a map, before the map markers from the previous map have to be erased. This happens before the request is made to the server. Then the new map is initialised and the markers will be made using the information from the server. Sometimes the map provides many samples and many markers make the map look chaotic. That is why a clustering package is used to solve this problem. The clusters have a color based on the number of markers they contain. This color is determined in the package. To clarify what the colors mean a small legend is provided on the top of the map. The markers themselves differ in color as well. The DNA sequence variants represented as Zero-radius Operational Taxonomic Units (ZOTU) numbers, determine what the color of the marker is. A color legend is provided for this situation as well. The clusters have one of

three colors whereas the marker colors lie on a gradient. At the beginning there was no button and everything was live updated. A button however gives a user more sense of control and it asks less computation of the program.

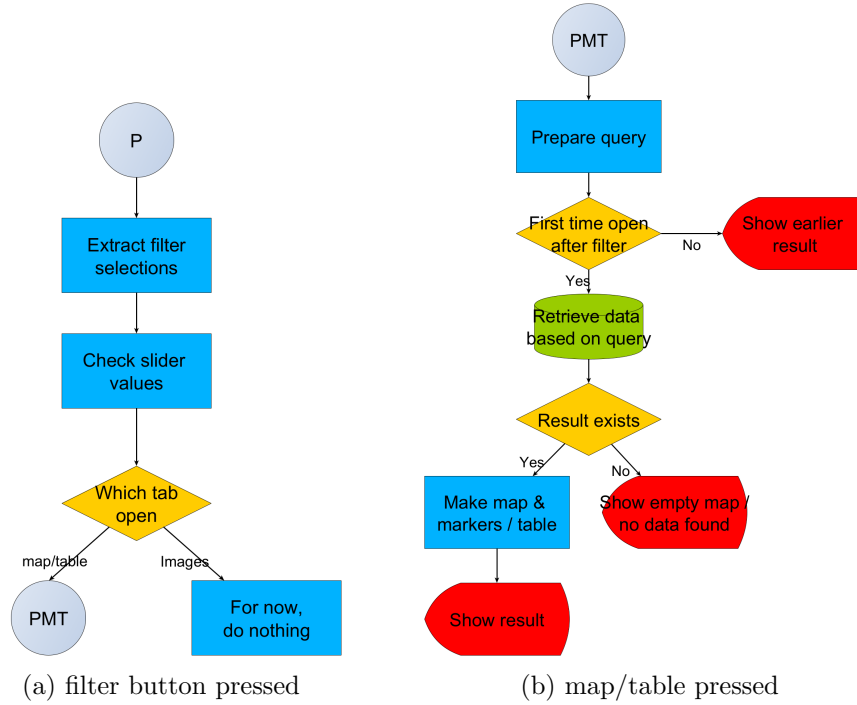


Figure 6: (a) Flowchart for **P**ressing a new tab [P]. (b) Flowchart for **P**ressing the **M**ap or **T**able tab [PMT]

4.2 Tabs

Figure 7 shows the flowchart for clicking on a tab. There are three tabs a user can choose from. When a user clicks on the tab they are already looking at, nothing will happen. That means no new request is send to the server. When they click on a different tab from the one they are on, one of two things can happen. To illustrate what happens an example will be provided below. A user is on the table tab. They then press the filter button and a new table will be made. Then they navigate to the map tab where a new tab will be made. The program will not make a new table when the user returns to the table tab, but simply shows the table made before. If the user had pressed the filter button while on the map tab and then navigated to the table tab a new table would be made. A JavaScript file will check which of the situations is occurring when the tab is changed. If needed, they will call the same function as if the filter button was pressed, which is explained in section 4.1.3. The filter information is extracted and sent to the server for either the map or the table tab. It depends on whether the table or the map tab is open. The process is the same so we will not go in depth into the controllers and models.

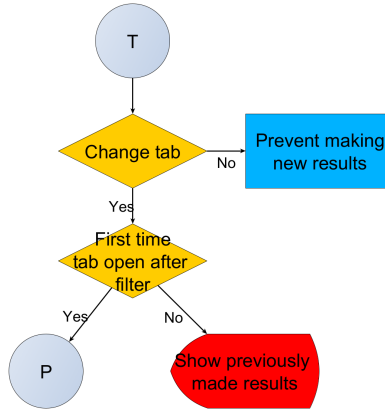


Figure 7: Flowchart for changing **T**abs [T]

4.3 Results

The results can be triggered to be made with different conditions. This can be seen in figure 8. The first is when loading the page. The default tab is the table which is why the table is made when loading the page. Interacting with the results means interacting with the visualisation that is made with the requested data. What result a user can interact with depends on which tab is open. In total there are three tabs: the table tab, the map and the chart tab. The latter tab is currently not active, but it is the next step in the process. The table tab is further explained in section 4.3.1 and the map is elaborated in section 4.3.2.

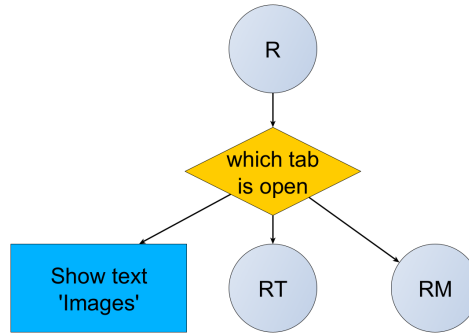


Figure 8: Flowchart for interacting with **R**esults [R]

4.3.1 Table

The table is a method to visualise the data. Figure 9 illustrates that it does that in a very orderly manner. When interacting with the table a user has multiple options. First they can change the number of rows in the table per page or move to the next page. Changing either one triggers an event which causes a new table to be made. The same steps are followed as if the filter button was pressed and the table tab was open explained in the section above. Different ways a user can interact with the table is clicking directly on the table itself. There are a few choices. A user could

click on one of the hyperlinks. These links lead to websites and explain either the term or send a user to the publication where the sample comes from. One could also change the order of a column in ascending or descending.

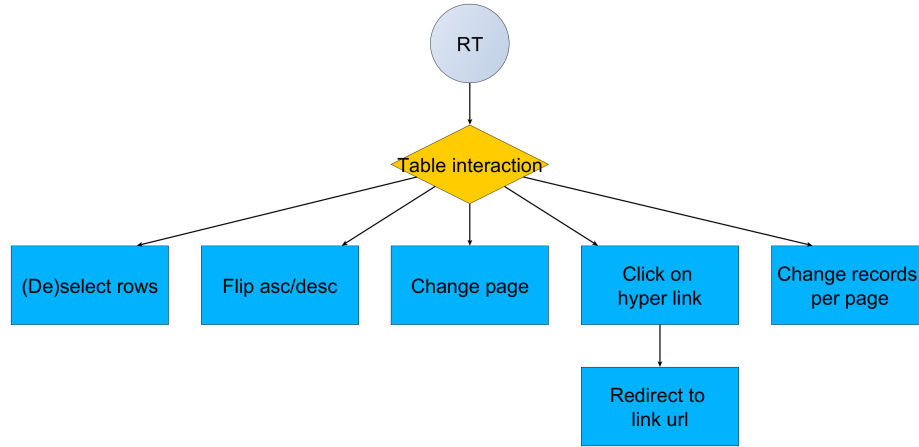


Figure 9: Flowchart for interacting with the **R**esult **T**able [RT]

4.3.2 Map

The map is the other implemented visualisation method. The flowchart can be found in figure 10. The map has two additional legends to explain what the colors in the map mean. There are two color groups. One for the cluster colors and one for the individual sample markers. People can zoom in by using the mouse or the mouse pad. This causes some of the clusters to split into smaller clusters and individual samples. The logic for the clusters comes from a Leaflet library.

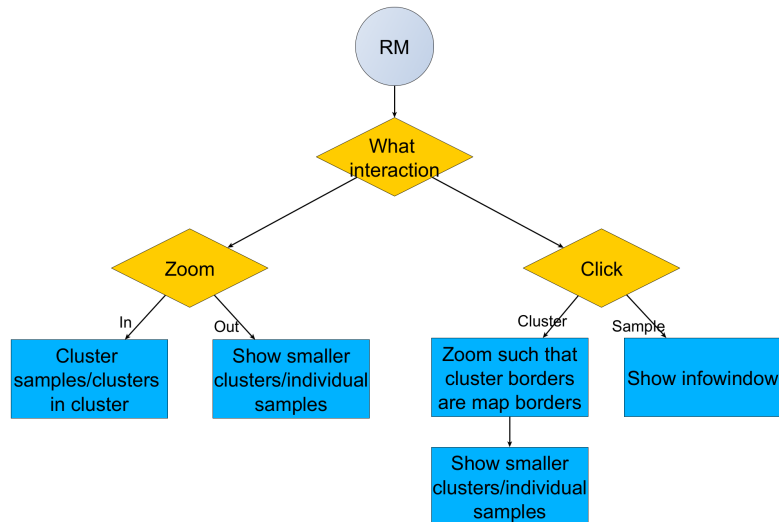


Figure 10: Flowchart for interacting with the **R**esult **M**ap [RM]

5 Results

The webtool can be accessed using <https://mycodiversity.liacs.nl/thesis/Mycodiversity/>. In order to verify whether the different filters combined work properly there are three use cases. The use cases are realistic scenarios of what researchers may look for. The use cases are also meant to see how the program deals with more complex queries. In addition to have some extra validation, separate components are tested as well. There is a performance test as well to see if the program works properly and to see if there is a difference between using localhost port 5000 (called local host for the remainder of the thesis), in the tables abbreviated with 'L' which is a development server and using the actual server, in the tables abbreviated with 'S', which it will eventually run on. Some of the outputs from these use components can be found in appendix B with figures 17-28. The appendix contains some visuals of the individual components of the page. To make a fair comparison the circumstances were as consistent as possible. All the tests were run on the same laptop around the same hour on the same network. In both cases there were no other programs running in the mean time. For each of the table the average time is given as well to make it easier to compare the localhost and the server in a more general sense.

5.1 Use cases

The first use case uses filters through all the options, location, taxonomy and environment. The second looks at an individual sample and one taxonomy group. The third and last use case looks for a specific region and pH range. In appendix A the steps to take to get the results are presented with visuals.

5.1.1 Use case 1

In the first use case all the filter components are used. In this example someone looks at the samples in Central Africa with Agaricomycetes. Central America is a subcontinent of the continent Africa. Agaricomycetes is a class of the phylum Basidiomycota. The selection is also limited to the tropical dry broadleaf forest biome term and the pH lies between 4.5 and 10. Table 4 shows how well the localhost and the server perform and Table 5 shows the validation. Figure 11 has the output map and Figure 12. Table 19 in Appendix A shows the selections made.

Time L table	Time S table	Time L map	Time S map
757 ms	206 ms	349 ms	50 ms

Table 4: use case 1 performance

Exp table	Actual table	Exp map	Actual map
221 rows	221 rows	2 samples	2 samples

Table 5: use case 1 validation

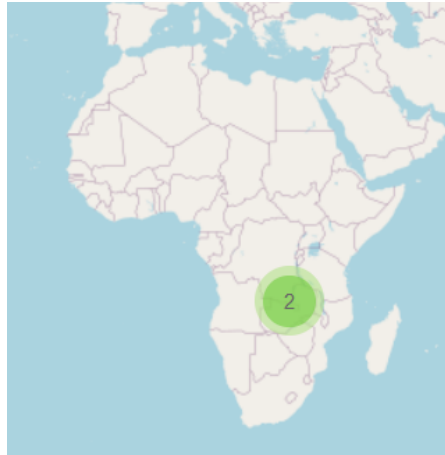


Figure 11: Use case 1 map result

Records per page 5

<input type="checkbox"/>	PHYLUM	ORGANISM	UNITE ID	ZOTU	CONTINENT	COUNTRY	SAMPLE	BIOSAMPLE	BIOME	ENVIRONMENT FEATURE
<input type="checkbox"/>	Basidiomycota	Psathyrellaceae_sp	SH220415.07FU	MddbOTU143671	Africa	Democratic Republic of Congo	SRS651466	SAMN02864650	tropical dry broadleaf forest biome	forest
<input type="checkbox"/>	Basidiomycota	Trechispora_sp	SH199563.07FU	MddbOTU143670	Africa	Democratic Republic of Congo	SRS651466	SAMN02864650	tropical dry broadleaf forest biome	forest
<input type="checkbox"/>	Basidiomycota	Favolaschia_cinnabarina	SH183921.07FU	MddbOTU143668	Africa	Democratic Republic of Congo	SRS651466	SAMN02864650	tropical dry broadleaf forest biome	forest
<input type="checkbox"/>	Basidiomycota	Peniophora_sp	SH204977.07FU	MddbOTU143659	Africa	Democratic Republic of Congo	SRS651466	SAMN02864650	tropical dry broadleaf forest biome	forest
<input type="checkbox"/>	Basidiomycota	Russula_sp	SH202449.07FU	MddbOTU143644	Africa	Democratic Republic of Congo	SRS651466	SAMN02864650	tropical dry broadleaf forest biome	forest

« 1 2 3 45 »

Figure 12: Use case 1 table result

5.1.2 Use case 2

The second use case looks at one sample and one phylum within that sample. This is the case for a user who wants to know if a fungal group is observed in a specific sample, in addition to know how many DNA sequence variants belonging to that specific group are detected. The sample is SRS651234 and the phylum is Ascomycota. Figure 13 shows the map result for the second use case and figure 14 contains the table result. Table 7 contains the validation results and Table 6 shows the performance between the server and the localhost version. Table 20 in Appendix A shows the selections made.

Time L table	Time S table	Time L map	Time S map
503 ms	104 ms	164 ms	74 ms

Table 6: use case 2 performance

Exp table	Actual table	Exp map	Actual map
80 rows	80 rows	1 samples	1 samples

Table 7: use case 2 validation



Figure 13: Use case 2 map result

Records per page 5

<input type="checkbox"/>	PHYLUM	ORGANISM	UNITE ID	ZOTU	CONTINENT	COUNTRY	SAMPLE	BIOSAMPLE	BIOME	ENVIRONMENT FEATURE
<input type="checkbox"/>	Ascomycota	Cenococcum_geophilum	SH199615.07FU	MDDBOTU059970	Europe	Estonia	SRS651234	SAMN02864418	temperate broadleaf forest biome	forest
<input type="checkbox"/>	Ascomycota	Leptodontidium_trabinellum	SH203235.07FU	MDDBOTU059963	Europe	Estonia	SRS651234	SAMN02864418	temperate broadleaf forest biome	forest
<input type="checkbox"/>	Ascomycota	Nectriaceae_sp	SH217585.07FU	MDDBOTU059957	Europe	Estonia	SRS651234	SAMN02864418	temperate broadleaf forest biome	forest
<input type="checkbox"/>	Ascomycota	Sordariomycetes_sp	SH199211.07FU	MDDBOTU059950	Europe	Estonia	SRS651234	SAMN02864418	temperate broadleaf forest biome	forest
<input type="checkbox"/>	Ascomycota	Hyaloscyphaceae_sp	SH181121.07FU	MDDBOTU059945	Europe	Estonia	SRS651234	SAMN02864418	temperate broadleaf forest biome	forest

« 1 2 3 16 »

Figure 14: Use case 2 table result

5.1.3 Use case 3

The third and last use case looks at a specific region and an environmental component. The region is the continent Europe and the pH values lie between 1 and 3. This is the case for a user who is interested to explore the fungal communities in Europe that live in extreme conditions, such as in high acidic soils. Figure 15 shows the map result for the second use case and figure 16 contains the table result. Table 9 contains the validation results and Table 8 shows the performance between the server and the localhost version. Table 21 in Appendix A shows the selections made.

Time L table	Time S table	Time L map	Time S map
1275 ms	468 ms	420 ms	103 ms

Table 8: use case 3 performance

Exp table	Actual table	Exp map	Actual map
11754 rows	11754 rows	62 samples	62 samples

Table 9: use case 3 validation



Figure 15: Use case 3 map result

Records per page 5

<input type="checkbox"/>	PHYLUM	ORGANISM	UNITE ID	ZOTU	CONTINENT	COUNTRY	SAMPLE	BIOSAMPLE	BIOME	ENVIRONMENT FEATURE
<input type="checkbox"/>	Ascomycota	Pseudoplectania_nigrella	SH187686.07FU	MddbOTU172432	Europe	Estonia	SRS651573	SAMN02864756	temperate coniferous forest biome	forest
<input type="checkbox"/>	Ascomycota	Hyaloscyphaceae_sp	SH181121.07FU	MddbOTU172431	Europe	Estonia	SRS651573	SAMN02864756	temperate coniferous forest biome	forest
<input type="checkbox"/>	Ascomycota	Meliniomyces_variabilis	SH181078.07FU	MddbOTU172429	Europe	Estonia	SRS651573	SAMN02864756	temperate coniferous forest biome	forest
<input type="checkbox"/>	Ascomycota	Ascomycota_sp	SH196794.07FU	MddbOTU172428	Europe	Estonia	SRS651573	SAMN02864756	temperate coniferous forest biome	forest
<input type="checkbox"/>	Ascomycota	Meliniomyces_sp	SH640522.07FU	MddbOTU172427	Europe	Estonia	SRS651573	SAMN02864756	temperate coniferous forest biome	forest

« 1 2 3 3251 »

Figure 16: Use case 3 table result

5.2 Validation

This section will cover the outcome of the validation tests.

For the validation we take a look at the table and the map. The most important aspect for the table to look at is whether the correct rows are presented. This is achieved by counting the rows and compare them with the expected number. For the map we both take a look at the number of samples and the number of DNA sequences found. This number should be the same as the number of rows in the table.

Table 10 shows the 5 different test that were performed for both the taxonomy and the location. Making the layers is a recursive function. For this reason, the assumption is made that if the first, the last and a middle layer work correctly, all of the layers work as they should. There are tests to validate if the program deals with multiple values as well, number 4 dealing with middle layers. As the database deals with AND operations two values in the same layer but in different branches should output the union of the two. Two values in different layers in the same branch the output is equivalent to the intersection. And lastly when the values are in different layers and in different branches there has to be a disjunction, meaning no results.

1	Value in First layer
2	Value in Last layer
3	Two values in same layer in different hierarchy branches
4	Two values in different layer in same hierarchy branch
5	Two values in different layer in different hierarchy branch

Table 10: 5 different tests for location and taxonomy

Table 11 shows the validation results for the location. It shows the expected number of rows the table should have and the actual value. It also shows how many samples are expected on the map and how many are generated.

loc selection	Exp table	Actual table	Exp map	Actual map
Africa	9836 rows	9836 rows	35 samples	35 samples
Srs651354	320 rows	320 rows	1 samples	1 samples
Cameroon, Germany	921 rows	921 rows	3 samples	3 samples
Africa, SRS651354	320 rows	320 rows	1 samples	1 samples
Germany, SRS651354	no data found	no data found	empty map	empty map

Table 11: validation location

Table 12 shows the validation results for the location. It shows the expected number of rows the table should have and the actual value. It also shows how many samples are expected on the map and how many are generated.

Table 13 shows the validation for the environment test. For the test one term was checked and for the other test a pH value was given between 4 and 10.

Table 14 shows the validation for the pH slider. When someone fills in a value the program checks whether the values are withing the borders. It also checks what happens when someone fills in correct values as a control. And it checks if someone fills in a minimum that is higher than the

tax selection	Exp table	Actual table	Exp map	Actual map
ascomycota	9836 rows	9836 rows	35 samples	35 samples
Archaeorhizomyces_borealis	320 rows	320 rows	1 sample	1 sample
Archaeorhizomycetes, insecta	921 rows	921 rows	3 samples	3 samples
ascomycota, Archaeorhizomyces_borealis	320 rows	320 rows	1 sample	1 sample
Insecta, Archaeorhizomyces_borealis	no data found	no data found	empty map	empty map

Table 12: validation taxonomy

environment	Exp table	Actual table	Exp map	Actual map
Terms Mixed forest	14186 rows	14186 rows	41 samples	41 samples
pH 4-10	74229 rows	74229 rows	233 samples	233 samples

Table 13: validation environment

maximum. The program ensures that the minimum is always the smaller than the maximum by swapping the input if needed.

pH	input	expected output	output
Min < 1	Min = -5	Min = 1	Min = 1
Max < 1	Max = -5	Max = 1	Max = 1
Min > 14	Min = 15	Min = 14	Min = 14
Max > 14	Max = 15	Max = 14	Max = 14
Min < max	Min = 5, max = 10	Min = 5, max = 10	Min = 5, max = 10
Min > max	Min = 10, max = 5	Min = 5, max = 10	Min = 5, max = 10

Table 14: Check numerical values

5.3 Performance

This section will show the comparison between a localhost host and the actual web server. This gives a general idea of the importance of developing locally as opposed to on the server. It also gives a good idea on how well the program performs. Tables 15 Table 16 Table 17 contain the performances of the location, the taxonomy and environment respectively. The time is noted for each test separately and the average time is given. The average time gives an indication of how long a researcher needs for a set of different tests.

Table 18 shows the performance of some general test. These are main components of the page. The first interesting test is to see how long it takes to load the complete page. It is also interesting to see how long it takes to load the table and map with all the samples. An additional check is to see how long opening the location and taxonomy components take in which the continents and phylums are made. The last part of the filter, the environment is also evaluated by looking at the biome terms. Finally, we have taken a look at the auto searches from the location and the taxonomy.

loc selection	Time L table	Time S table	Time L map	Time S map
Africa	1117 ms	429 ms	356 ms	104 ms
Srs651354	405 ms	90 ms	200 ms	321 ms
Cameroon, Germany	733 ms	107 ms	348 ms	297 ms
Africa, SRS651354	424 ms	154 ms	176 ms	279 ms
Germany, SRS651354	439 ms	104 ms	186 ms	70 ms
Average time	623.6 ms	176.8 ms	253.2 ms	214.2 ms

Table 15: Performance location **Local** [L] vs **Server** [S]

tax selection	Time L table	Time S table	Time L map	Time S map
ascomycota	1678 ms	551 ms	534 ms	110 ms
Archaeorhizomyces_borealis	611 ms	124 ms	449 ms	95 ms
Archaeorhizomycetes, insecta	832 ms	310 ms	504 ms	32 ms
ascomycota, Archaeorhizomyces_borealis	464 ms	258 ms	164 ms	92 ms
Insecta, Archaeorhizomyces_borealis	436 ms	88 ms	248 ms	100 ms
Average time	804.2 ms	266.2 ms	379.8 ms	85.8 ms

Table 16: Performance taxonomy **Local** [L] vs **Server** [S]

environment	Time L table	Time S table	Time L map	Time S map
Terms Mixed forest	944 ms	379 ms	244 ms	88 ms
pH 4-10	1928 ms	559 ms	500 ms	348 ms
Average time	1436 ms	469 ms	372 ms	218 ms

Table 17: Performance environment **Local** [L] vs **Server** [S]

general tests	Time L	Time S
Whole page	4250 ms	1630 ms
Table all samples	2228 ms	676 ms
Map all samples	508 ms	257 ms
Continents	136 ms	169 ms
phylum	204 ms	74 ms
biome terms	213 ms	60 ms
Auto search 'sou'	528 ms	79 ms
Auto search 'dio'	1039 ms	500 ms
Average time (without whole page and table)	438 ms	190 ms
Total time (whole page and table)	6478 ms	2306 ms

Table 18: Performace general components **L**ocal [L] vs **S**erver [S]

6 Conclusions

In this thesis we have investigated different algorithmic approaches to retrieve data over different scales with analysis. In addition there are visual representations made for the results. The data can be retrieved through three main filter components. These were investigated and made more efficient and there are methods added. They were analysed through a validation check and performance evaluations.

At the start of the project there were four checkbox lists. The continents and the countries were combined in one tree list. These only had one 'layer'. All of the layers were added for this project meaning people can search more specifically and therefore retrieve the desired information faster. These layers could be made with separate checkbox lists but that would be chaotic for the users. That is why a hierarchical treeview was used. Another addition that was made are the autocomplete search bars. The pH is added for the environment. This is another specification for people to filter on. Users can therefore search faster and more directed than before. We can conclude that the multiscale access of the MycoDiversity DataBase was successfully sped up.

Another part of the research goal was to investigate visualization methods for fungal biodiversity data. The table is a visualization method that already existed. The information displayed in the table has not been altered. The newly investigated visualization method was a map. This was first attempted with the google maps API. However, this was not the fastest method. That is why the map was eventually produced with the Leaflet API which was faster and presented the results more clearly. The map was validated and the performance is always fairly consistent no matter what the query is. The tables are more susceptible for change in the number of samples.

Tested the three main filter components. These all passed the validation tests. The autocomplete searches worked as well. The pH sliders were validated as they have free input. Users should not be able to fill in wrong input. That is why there is a control and a validation to check if it works properly. The sliders passed the test and are therefore adjusted against invalid input from users. To make an additional analysis, three use cases were performed to see if the combination of the different filters work as well. The use cases passed the validation. The performance was good as well due to the small amount of results. The filter combinations are the cause for only getting a few results. The conclusion we could draw is that the database can handle large amounts of data but processing the results can become slow. The reason the table is really slow is because of the rowcount. This is a built-in feature which cannot easily be accelerated. Alternatives like `count(*)` are slower. The expectation is that users will look at quite specific samples which would mean that it would not be as big of a problem.

The code was originally fairly cluttered and in one place. The existing code was separated in the MVC components and the new code was built following the same design pattern. This means that the different components are divided over different files. A README is added to clarify how to add and to keep everything consistent. This makes everything more sustainable. As this is part of a long term project, consistency is an important factor.

7 Further Research

7.1 Other pages

The biodiversity search is only one of the pages. The entire project should provide more functionalities, a literature search for example.

7.2 More samples

The MDDB used in this project contained around 500 samples. This is a fraction of the available samples. The sample number also grows in time

7.3 Download

Visualizing the data in tables and maps is already interesting but another important thing would be to represent the DNA data. A researcher should be able to download this information. The DNA data will be presented in the FASTA format.

7.4 Efficiency

This project was mainly focused on creating new features and not necessarily on the efficiency of those methods. The program could also be made more efficient by for example using a cache.

7.5 Other db

The decision to use MonetDB might not have been completely fair. The previous research was meant to compare R-DB and C-DB, therefore two R-DB's were compared along with one C-DB. The C-DB performed the best which makes sense. The thing is though, for the bigger project it might also be interesting to compare different C-DB's as well. MonetDB is very fast as far as we can tell but unfortunately has not a very supportive community. A database like MariaDB might be interesting. On top of a bigger community, MariaDB can be integrated in most frameworks like Django or Laravel as well.

7.6 More tabs

The results can be displayed in two ways, in a table and in a map. It would be interesting to add different methods to display the results. The most important way is through the use of charts and graphs. These could include charts where the relations of the different fungi are made clear. In combination with the map this could yield some interesting information.

7.7 More filter options

The location and the taxonomy options are quite complete. Only the layout can have some more visual feedback. The environment however is very limited. There are only two aspects on which you can filter. There are a lot more features in the environment one could filter on. Think for example about the elevation or nutrition level in terms of nitrogen or carbon.

References

- [1] Kabir G Peay, Peter G Kennedy, and Thomas D Bruns. Fungal community ecology: a hybrid beast with a molecular master. *Bioscience*, 58(9):799–810, 2008. <https://doi.org/10.1641/B580907>.
- [2] Rose Marie Woodsmall and Dennis A Benson. Information resources at the national center for biotechnology information. *Bulletin of the Medical Library Association*, 81(3):282, 1993. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC225790/>.
- [3] Irene Martorelli, Leon S Helwerda, Jesse Kerkvliet, Sofia IF Gomes, Jorinde Nuytinck, Chivany RA van der Werff, Guus J Ramackers, Alexander P Gulyaev, Vincent SFT Merckx, and Fons J Verbeek. Fungal metabarcoding data integration framework for the mycodiversity database (mddb). *Journal of integrative bioinformatics*, 17(1), 2020. <https://doi.org/10.1515/jib-2019-0046>.
- [4] Heath E O’Brien, Jeri Lynn Parrent, Jason A Jackson, Jean-Marc Moncalvo, and Rytas Vilgalys. Fungal community analysis by large-scale sequencing of environmental samples. *Applied and environmental microbiology*, 71(9):5544–5550, 2005. <https://doi.org/10.1128/AEM.71.9.5544-5550.2005>.
- [5] SRA Toolkit Development Team. Sra toolkit, 2011. <https://github.com/ncbi/sra-tools>.
- [6] Blake Barnhill and Matt David. Row vs column oriented databases, Jun 2021. <https://dataschool.com/data-modeling-101/row-vs-column-oriented-databases/>.
- [7] MonetDB. Open source column-oriented database management system. <https://www.monetdb.org>.
- [8] Odbc driver. <https://docs.microsoft.com/en-us/sql/connect/odbc/download-odbc-driver-for-sql-server?view=sql-server-ver15>.
- [9] History of php - manual. <https://www.php.net/manual/en/history.php.php>.
- [10] Usage statistics of php for websites. <https://w3techs.com/technologies/details/pl-php>.
- [11] Matt Silverman. The history of html5, Jul 2012. <https://mashable.com/2012/07/17/history-html5/?europa=true>.
- [12] Ankush Sharma and Aakanksha Sharma. Introduction to html (hyper text markup language)-a review paper. *Int. J. Sci. Res*, 7(5):2017–2019, 2018.
- [13] Bert Bos, Dec 2016. <https://www.w3.org/Style/CSS20/>.
- [14] Uikit. <https://getuikit.com/>.
- [15] Allen Wirfs-Brock and Brendan Eich. Javascript: the first 20 years. *Proceedings of the ACM on Programming Languages*, 4(HOPL):1–189, 2020. <https://doi.org/10.1145/3386327>.
- [16] John Resig, Russ Ferguson, and John Paxton. page 77–110. A Press, 2015.

- [17] Vladimir Agafonkin. Leaflet: an open-source javascript library for mobile-friendly interactive maps. 2011. <https://leafletjs.com/index.html>.
- [18] Juan Pablo VentosoMarch. End point, Mar 2019. <https://www.endpoint.com/blog/2019/03/23/switching-google-maps-leaflet>.
- [19] British Columbia Institute of Technology. Php style guide, Sep 2019. <https://codeigniter.com/userguide3/general/styleguide.html>.

A Appendix use cases


Step	visual
1	Location <input checked="" type="checkbox"/> Central Africa
2	Taxonomy <input checked="" type="checkbox"/> Agaricomycetes
3	<input checked="" type="checkbox"/> tropical dry broadleaf forest biome
4	<div>PH</div> <div>4.5</div> <div>10</div> 

Table 19: Use case 1 steps. 1 selecting the location. 2 selecting the taxonomy. 3 selecting the biome term. 4 selecting the pH range

Step	visual
1	Location <input checked="" type="checkbox"/> SRS651234
2	Taxonomy <input checked="" type="checkbox"/> Ascomycota

Table 20: Use case 2 steps. 1 selecting the location. 2 selecting the taxonomy.


Step	visual
1	Location <input checked="" type="checkbox"/> Europe
2	<div>PH</div> <div>1</div> <div>3</div> 

Table 21: Use case 3 steps. 1 selecting the location. 2 selecting the pH range.

B Appendix page components

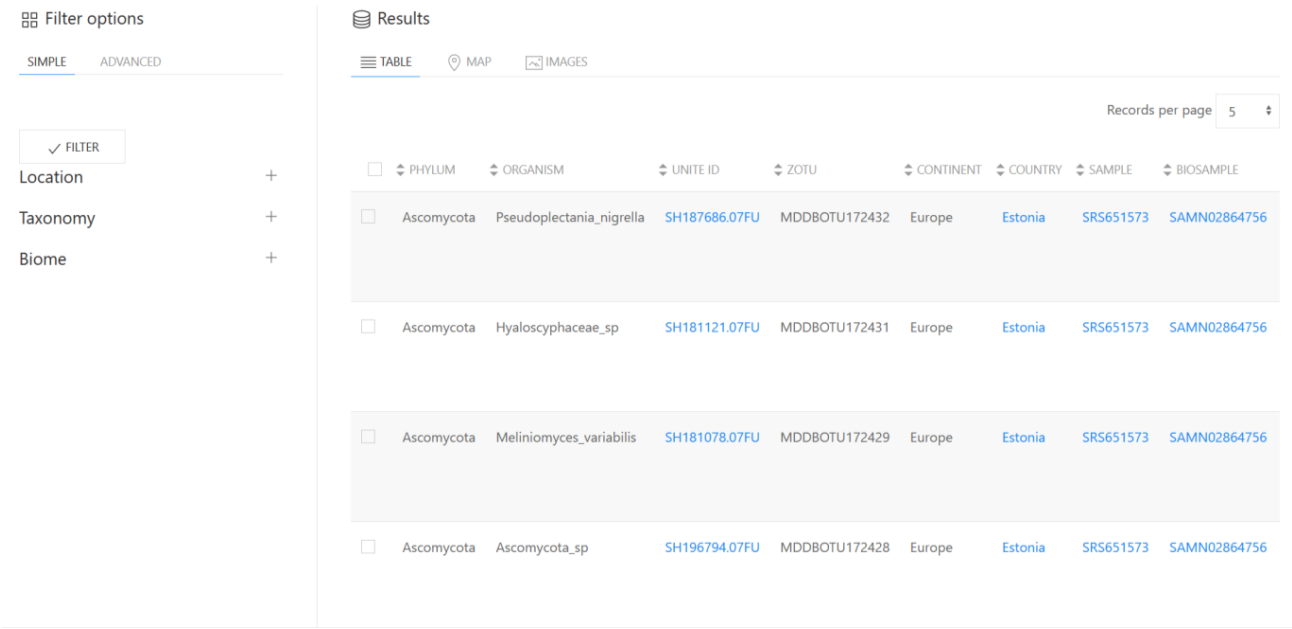


Figure 17: Complete page

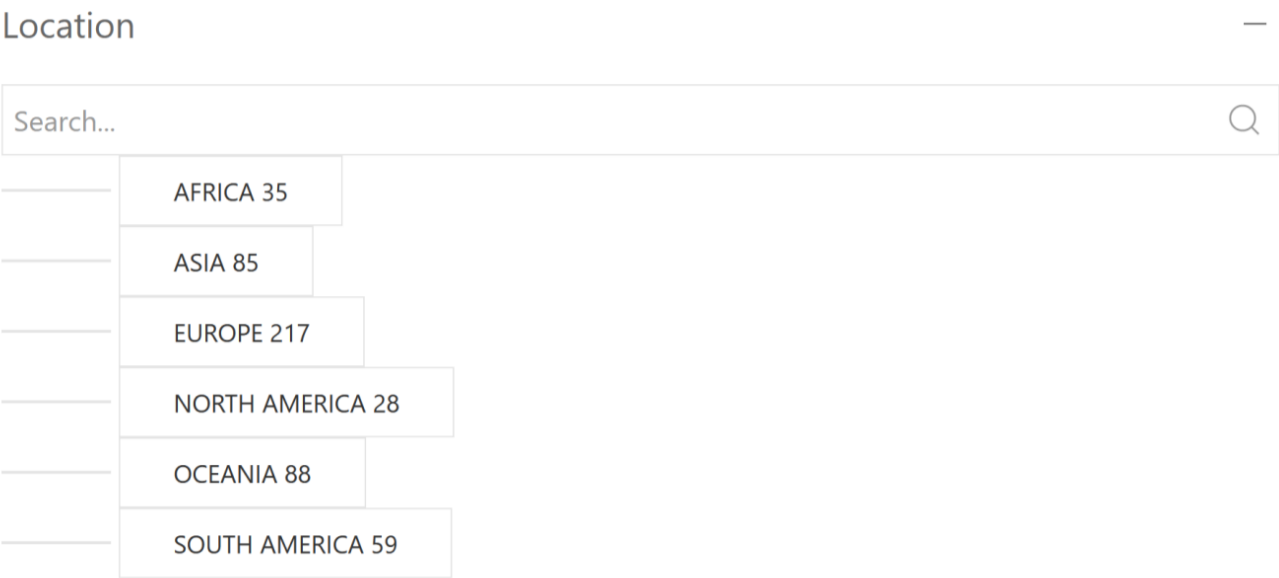


Figure 18: location

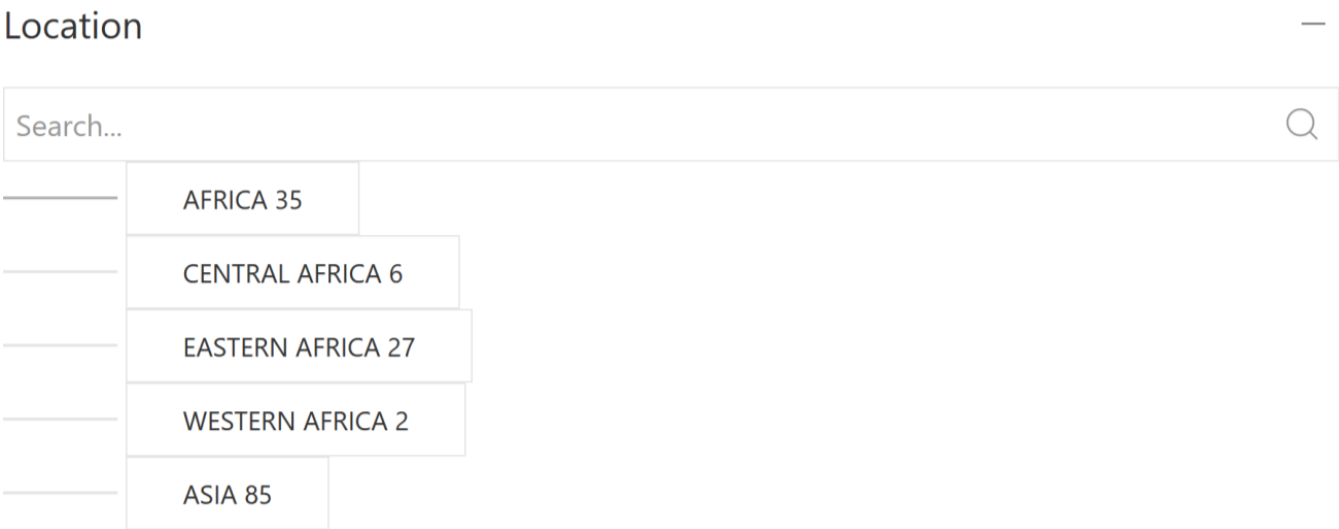


Figure 19: Location, Africa expanded

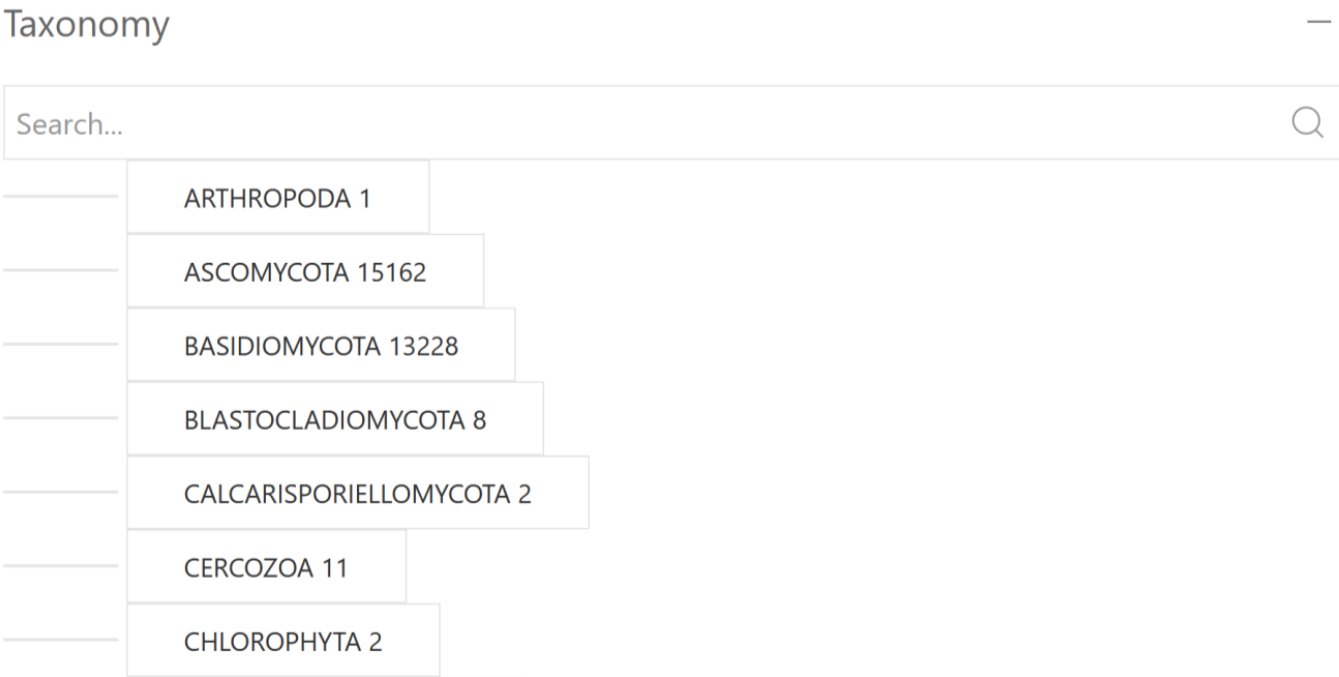


Figure 20: Taxonomy

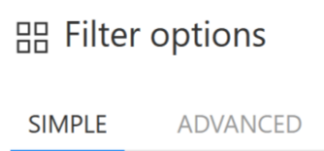


Figure 21: Filter options

sou

Q

South America continent

South America subcontinent

Southern Asia subcontinent

Southern Europe subcontinent

Southeast Asia subcontinent

Figure 22: Autocomplete search

ENV BIOME TERM

☐ flooded grassland biome

☐ mediterranean forest biome

☐ mixed forest biome

☐ montane grassland biome

☐ montane shrubland biome

☐ subpolar coniferous forest biome

☐ temperate broadleaf forest biome

☐ temperate coniferous forest biome

☐ temperate grassland biome

☐ tropical dry broadleaf forest biome

☐ tropical moist broadleaf forest biome

☐ tundra biome

Figure 23: Biome terms

PH

1

14

Figure 24: pH slider



Figure 25: Filter button

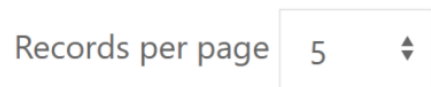


Figure 26: Records per page current number

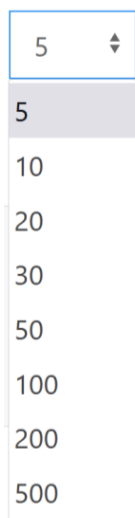


Figure 27: Records per page options



Figure 28: Pagination