

**École polytechnique de Louvain**

# **Improvement of the management of the Walloon Entomological collections thanks to the use of Docker and a SQL database**

Authors: **Jean SCHOT, Simon GROGNARD**

Supervisors: **Axel LEGAY, Frederic FRANCIS**

Readers: **Christophe CROCHET, Grégoire NOËL**

Academic year 2022–2023

Master [120] in Computer Science & Master [120] in Computer Science  
and Engineering

# Abstract

This paper brings together the information needed to create a database used for an entomological museum that can be modified from a containerized web application using Docker.

It's becoming increasingly necessary to computerize everyday processes, retrieving and modifying museum data is one of them. In our case, our aim was to facilitate the encoding work of Gembloux entomologists, who until now had been working with paper forms and Excel spreadsheets.

To achieve this, we designed the database and the architecture of the various microservices that make up our application. We then implemented the database in SQL, as well as the scripts for transforming spreadsheets into SQL data and vice versa, using Python. The front-end and back-end microservices were implemented in JavaScript. Three different types of tests were then carried out: pgTAP tests for the database, SQL injection attempts with SQLMap, and finally user tests based on existing literature.

The aim of this work was to make it immediately usable by the Gembloux entomology team, although it can of course be improved in the future to meet new needs that may arise from the collection of new data. The use of microservices allows for much easier evolution and maintainability.

The whole implementation of our work is freely accessible on GitHub:  
<https://github.com/Simon13411/Memoire>

## Acknowledgement

We would like to express our gratitude to several people.

Thanks to Professor Axel Legay for giving us the opportunity to work on this project for our dissertation.

Thanks also to Grégoire Noël for his dedicated supervision throughout the year, and his encouragement which kept us motivated.

We would also like to thank his team and fellow entomologists for the attention they paid to our work and their participation in the test phase.

We would also like to thank Gembloux's IT department for taking the necessary steps with ULiège to host the web application.

Finally, we are extremely grateful to the people who have accompanied us throughout our studies, namely our friends and roommates from Garfield Kot, who gave us ideas when we needed them, and who have been a constant support during this last year.

Finally, we'd like to thank our family for always believing in us and allowing us to undertake these studies.

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Background and Context . . . . .	5
1.2	Purpose and Objectives . . . . .	6
1.3	Scope and Limitations . . . . .	6
1.4	Methodology . . . . .	7
<b>2</b>	<b>Literature Review</b>	<b>8</b>
2.1	Overview of Museum Databases and Websites . . . . .	8
2.2	Best Practices in Database and Website Design . . . . .	9
2.2.1	DaRWIN . . . . .	9
2.2.2	Natural history museum . . . . .	10
2.2.3	GBIF . . . . .	11
2.3	Important aspects for our Database and Websites . . . . .	12
<b>3</b>	<b>Analysis and Design</b>	<b>13</b>
3.1	Data Modeling and Database Design . . . . .	13
3.1.1	Goal of database modeling . . . . .	13
3.1.2	Analysis of needs . . . . .	14
3.1.3	Conceptual model . . . . .	14
3.1.4	Logical model . . . . .	16
3.1.5	Implementation choice . . . . .	19
3.2	User Interface Design . . . . .	19
3.2.1	Key principles . . . . .	19
3.2.2	Analysis of needs . . . . .	20
3.2.3	User Interface choices . . . . .	21
3.3	System Architecture . . . . .	23
3.3.1	Microservices architecture with Docker (containerization) . .	23
3.3.2	Efficient division of microservices . . . . .	24
3.3.3	Practical example . . . . .	24

<b>4</b>	<b>Implementation</b>	<b>27</b>
4.1	Databases Implementation . . . . .	27
4.1.1	Physical model . . . . .	27
4.1.2	Entomological database . . . . .	29
4.1.3	Login database . . . . .	40
4.2	Website Development . . . . .	40
4.2.1	Docker, Dockerfile and Docker-compose . . . . .	40
4.2.2	Front-end . . . . .	45
4.2.3	Back-end . . . . .	51
4.2.4	API Gateway - Forwarding requests from Front-end to Back-end	53
<b>5</b>	<b>Testing and Evaluation</b>	<b>57</b>
5.1	Test Plan . . . . .	57
5.2	Test Results and Analysis . . . . .	58
5.2.1	Testing Queries using pgTAP . . . . .	58
5.2.2	Security tests using SQLmap . . . . .	60
5.2.3	User testing . . . . .	64
<b>6</b>	<b>Conclusion</b>	<b>68</b>
6.1	Summary of Accomplishments . . . . .	68
6.2	Limitations and Future Work . . . . .	69
<b>A</b>	<b>Creation of the entomological table</b>	<b>73</b>
<b>B</b>	<b>Rule for filling the different excel</b>	<b>79</b>
<b>C</b>	<b>WebApp screenshots</b>	<b>84</b>

# Chapter 1

## Introduction

### 1.1 Background and Context

It is a frequent practice for businesses and researchers to store data in Excel files. However, searching for specific information in these files and making them easily accessible through a website can be challenging. One solution to improve storage and searching abilities and increase storage capacity is by utilizing an SQL database. Despite this option, many people still prefer using Excel due to its user-friendliness. To ensure that no data is lost and that the migration process is thorough, a careful approach is essential while migrating from Excel files to an SQL database.

To provide a specific instance, we will demonstrate the process of transferring the Excel database of the Walloon Entomological Conservatory to an SQL database and the steps to establish their online platform. The Walloon Entomological Conservatory is a compilation of insects from around the globe stowed away in containers, overseen by the Gembloux Agro Bio-Tech faculty of the University of Liège. Organizing data pertaining to these insects proved to be a challenging task for the team owing to the size of the collection. Making this information digital and easily searchable on their website was an essential need to share knowledge about these insects with people worldwide.

As a solution to these problems, we developed a Web-App with a microservices architecture utilizing Docker. The App incorporates a login system, communication with the database, front-end, and gathering data from a NAS through an NFS share. Our team collaborated closely with the entomologists in an Agile environment to ensure that their needs were met through the development of our system.

## 1.2 Purpose and Objectives

The objective of this paper is to explain the different steps to follow in order to create a SQL database that will be filled in with Excel files but also to make them accessible to public through a website.

This paper will show the theoretical information needed to do such a work but will also give a concrete example: the set up of a management system for the collections of the Walloon Entomological Conservatory.

For the set up of this management system, three main objectives will have to be achieved.

The first objective is to create a SQL database that contains all the insects and their boxes but also a large number of information collected by entomologists listed in Excel tables without any loss of data.

It is also necessary to create a web application that will serve as a website for visitors to the entomological museum but also as a GUI for entomologists.

They will be able to manage the database in a way that is much easier for people who are not comfortable with databases than with SQL queries.

By bringing theoretical information about setting up a database and a website displaying the information of a database, but also by giving a concrete and complete example of the development of the management system of the collections with the Walloon Entomological Conservatory, this paper aims to explain a complete, correct and scientific approach for this task and to show the result after following all the steps explained in this paper.

## 1.3 Scope and Limitations

The project aims to create a SQL database and to develop a containerized Web-App for the entomological team of the Walloon Entomological collections.

Gathering information about user needs and understanding this information is an integral part of our work. The Web-App will have to meet the needs of the entomologists with whom we work as well as be pleasant and easy to use. In addition, the old database system based on Excel files must be transferable to our SQL database. This Excel to SQL transfer system will have to be able to be used to add new data.

A lot of features are possibly interesting for this kind of application, we will only focus on the one that can be used immediately, as the collection of information

about insect boxes and individuals is still in progress at the Walloon Entomological Conservatory.

## 1.4 Methodology

For this work, we decided to adopt the agile software development method.

Agile methodology is an approach to software development that focuses on adaptability, collaboration, flexibility and speed. It is based on an iterative and incremental process, where requirements and solutions evolve through regular collaboration between development team members and stakeholders. Agile methodologies emphasize the active involvement of stakeholders in the development process so that the final products better meet the needs and expectations of users [1].

Since its inception, the Agile method has improved communication, product quality and customer satisfaction while reducing development time and costs [2].

To implement this agile methodology, it is essential to have regular and effective communication with the project stakeholders. This can be achieved through frequent meetings (that sometimes seem too frequent or too long [2]), software demonstrations during development, regular feedback and project reviews [1].

We, therefore, chose to collaborate closely with the entomologists, who acted as clients for our team. We held regular meetings with them to discuss the specific needs and requirements of the application in order to try to reach an optimal level of satisfaction. As a result, we were able to develop a final product that met the needs and expectations of their team.

The objective was also to obtain a maintainable, reusable and upgradable code. For this, we have done everything possible to make the code as understandable as possible by making clear choices of implementations, justification and comments, as well as writing several READMEs.

The different explanations in this paper will be discussed in this order: first a literature review of the different existing similar museums, the approach taken for the analysis and design, the details of our implementation choices and how these have been set up and finally our testing and evaluation plan in order to validate our solution. Both theoretical elements and more practical explanations will be given.



# Chapter 2

## Literature Review

### 2.1 Overview of Museum Databases and Websites

In this literature review, we will analyze 3 websites that maintain a database similar to ours. First of all, we will look at a Belgian website called DarWIN as the entomological museum of Gembloux has ideas for collaboration with it. Then we will look at the website of the National History Museum in London, which contains a large database not only on insects but also on plants, minerals, and fossils. Finally, we will look at a website that groups large data from all over the world, GBIF (Global Biodiversity Information Facility).

The Royal Belgian Institute of Natural Sciences has a large collection of zoology, anthropology, paleontology, mineralogy and geology. These collections have their roots in the time of Belgium's independence and continue to grow thanks to various donations from people and countries. DaRWIn was created for the purpose of online referencing and currently contains more than 2,000,000 specimens encoded in their database. This number continues to increase every day thanks to the heritage department, the various curators and their assistants. The website contains, among other things, data on entomology. [3]

For the National History Museum of London website, we will focus on the data portal which contains a large amount of data on different collections. This part aims to give free access to their data, allowing each user to explore, download and reuse the data for their own research. These different collections are some of the largest on earth, with almost every living thing represented. They have just over 5 million records in their database for all collections, but they aim to have as much as 20 million records in the next 5 years. [4]

For the last website, their main objective is to make a large database from all over the world freely available. So, they have created a website that gathers all this information. The holders of the collections give access to their datasets through Creative Commons licenses. [5]

## 2.2 Best Practices in Database and Website Design

### 2.2.1 DaRWIn

The screenshot displays the DaRWIn website's 'Specimen search criteria' form. The header features the DaRWIn logo and navigation links: ZOOLOGICAL SEARCH, GEOPALEO SEARCH, COLLECTIONS, TAKE A TOUR, and CONTACTS. Below the header, the form is organized into several sections:
 

- Taxonomy:** Includes fields for Scientific Name, Common Name, and a Level dropdown menu.
- Collections:** Features an Institution Identifier field and a list of collections with checkboxes, including 'Royal Belgian Institute of Natural Sciences', 'BE-RBINS Anthropology', 'BE-RBINS Biobank', 'BE-RBINS Entomology', and 'BE-RBINS Geology'.
- Countries:** Includes a Tag field with a placeholder 'Please use "" as tag separator'.
- Types:** A list of specimen types with checkboxes: holotype, lectotype, neotype, paralectotype, paratype, specimen, and type.
- Sexes:** A list of sex categories with checkboxes: female, female & male, male, mixed, non applicable, not stated, undefined, and sex unknown.
- Stages:** A list of developmental stages with checkboxes: 18 months, adult, adult but not large, adult + eggs, adult + embryo, adult + mae, and adult + mae.

 The URL 'https://darwin.naturalsciences.be/darwin/' is visible in the bottom left corner.

Figure 2.1: DaRWIn interface

The DaRWIn website uses PostgreSQL for its database and a customizable web interface thanks to the Symfony framework [6]. The use of this website contains two important parts:

- a read-only one, which allows any user to browse the website and look at the different data it contains.
- connected user, which allows these users depending on their level in the site (user, encoder, curator, administrator) to be able to modify or add new data on the site, they can also manage daily information and use sensitive data.

The different stored data have several interesting attributes which are separated into two main parts, first the information that is directly related to the specimen and other information that can be modified day-to-day.

sample data and related information	other features
place and date of collection	loans
missions and collectors	printing labels for storage
identifiers	statistics
technicians involved	reporting
taxonomy	
identification information	
bibliography	
related files	
storage	

It was already possible to export these data to excel, PDF or GGeoJSON files but recently this website allows the import of new data from files.

## 2.2.2 Natural history museum

The screenshot shows the 'Specimens' page of the Natural History Museum Data Portal. The page has a green header with the museum logo and navigation links. Below the header, there's a search bar and a table of specimen records. The table has columns for various attributes including gdfissue, associat, barcode, baseOf, bed, catalogNumber, catalog, chondri, chronos, class, clutchSize, collectio, collectio, and collectio. The table is filtered to show 2,169,993 records. The left sidebar shows a list of collection codes and their corresponding specimen counts.

Figure 2.2: Natural history museum interface

Like DaRWIn, the portal of the Natural History Museum of London allows to add new data. Each contributor is encouraged to use a specific form for their data but this is not mandatory [7]. However, some data must still be included as:

- Title of the dataset
- Title of the dataset

- Category of the dataset
- Author

Concerning the data, it is also possible to download them because their goal is to share them and to have a free access. To do this, you must enter your email address and select the desired data.

Another point that the portal allows is the permission for users to send bug reports if the data is badly encoded. This allows for quick changes to be made and ensures the quality of the data. This is quite important because the museum shares its information with several other entities including GBIF.

It is interesting to note that the website integrates a map showing the location where each specimen was discovered.

### 2.2.3 GBIF

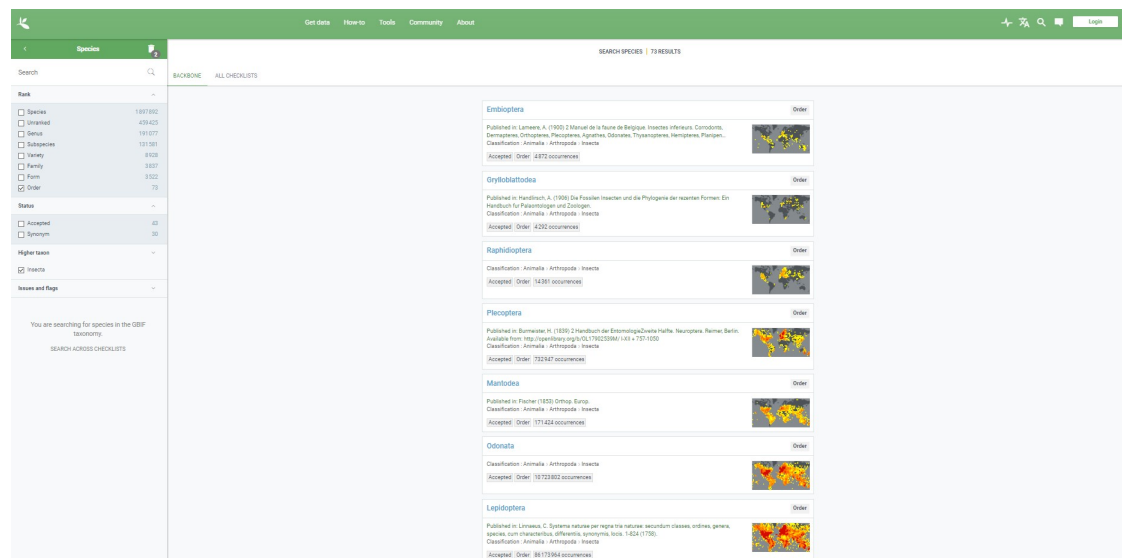


Figure 2.3: GBIF interface

Regarding GBIF, their main point is to share their data with the world. To do this and like other websites, it is possible to download the data you want[8]. As this website gathers large collections from all over the world, they have integrated directly the citations of the collections on the data pages. Thanks to this, users can easily use and quote the data they use. Here is an example for the Hymenoptera collection of the National Museum of Natural History

in Paris: MNHN, Chagnoux S (2023). The Hymenoptera collection (EY) of the Muséum national d'Histoire naturelle (MNHN - Paris). Version 77.314. MNHN - Museum national d'Histoire naturelle. Occurrence dataset <https://doi.org/10.15468/r3eez6> accessed via GBIF.org on 2023-05-23.

As said above, this web site gathers a lot of data from all over the world, so it is interesting to have a lot of statistics on the site about particular specimens. In addition to the different important information about each specimen, a map indicating where to find them is also present.

## **2.3 Important aspects for our Database and Websites**

It is now time to look at what can be interesting for our website and our database.

First of all, we notice that the three analyzed websites allow downloading and sharing of data. Since this is also a request from the entomologists of Gembloux, it is important to take this into account. The use of excel files will be preferred because entomologists already use this way of encoding.

Then, we notice that in many different ways, all websites also allow the insertion of new data, however it must be acceptable and well encoded. It is therefore necessary to have a certain encoding code and specific users who can add this data. It is therefore interesting to create several levels of users.

Finally, we notice that the last two web sites use maps to visualize the places where each insect lives. It is therefore interesting to take into account when developing the database and the website a way to display this information.

# Chapter 3

## Analysis and Design

### 3.1 Data Modeling and Database Design

#### 3.1.1 Goal of database modeling

Data modeling is a way to have a visual representation of a database. For this purpose, it creates link between entities thanks to relationship. The creation of a model is the first step when creating a database as all subsequent work will depend on the choices made when creating this model.

The creation of a model is quite important for several reasons:

- A small change in the model can have a big effect on the database
- A model reflects how the data is organized
- It will make the programming much easier
- It helps for the forms and the data encoding to avoid having a wrong type of data

A good data model is one that is complete, it means that are no missing data, this can cause majors problem. Another important point is to be careful not to have any redundancy. This means that it can cause a lack of storages in large database. During the elaboration of the model, it is important to have good look to the different rules the database must contain. For example, if an attribute cannot be empty, the model must mention it. It is also important not to be tied to a particular use of the data, otherwise it may only be useful in a certain application [9].

### 3.1.2 Analysis of needs

During the elaboration of the model, it is really important to analyze what the customer are looking for and what they already have. In this case, the Walloon Entomological Conservatory has several excel file with a lot of data, some of their data was not really important to store in the database, however some of them have special specificities. After some meeting with the person in charge of the museum, we were able to identify the important data that are needed to be stored. To be in total agreement, it is quite interesting to make intermediate reports with clients.

### 3.1.3 Conceptual model

The first step to design the database is to identify the different entities and to make the relationship between these entities. This model does not take into account the details. The entities are represented by boxes with their names. It will correspond to the different table in the database. For the relationship, there are several notations as explained below.

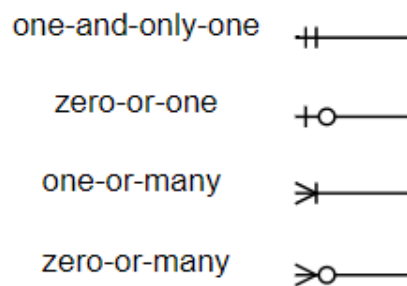


Figure 3.1: Crow's Foot Notation illustrated

Relationships are not necessarily symmetrical, and the nature of the relationship of A to B is specified at the connection with B.[10, 11]

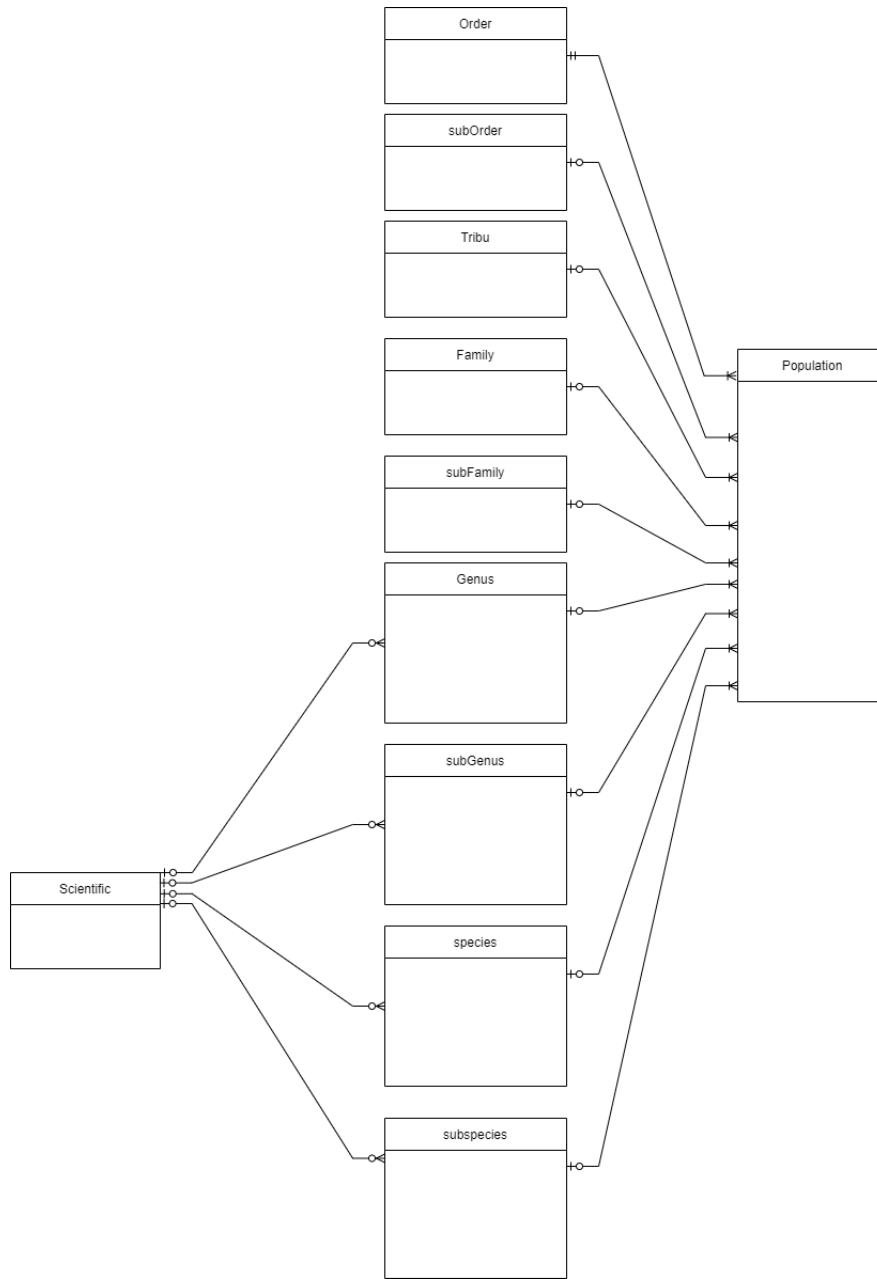


Figure 3.2: Left part of the conceptual model

The *population* table is one of the central tables of the database. It should have a relation to *Order*. However, it can have relations to *subOrder*, *Tribu*, *Family*, *subFamily*, *Genus*, *subGenus*, *species*, *subSpecies*.

The *Genus*, *subGenus*, *species*, *subSpecies* table can have a relation with the



*Scientific* table.

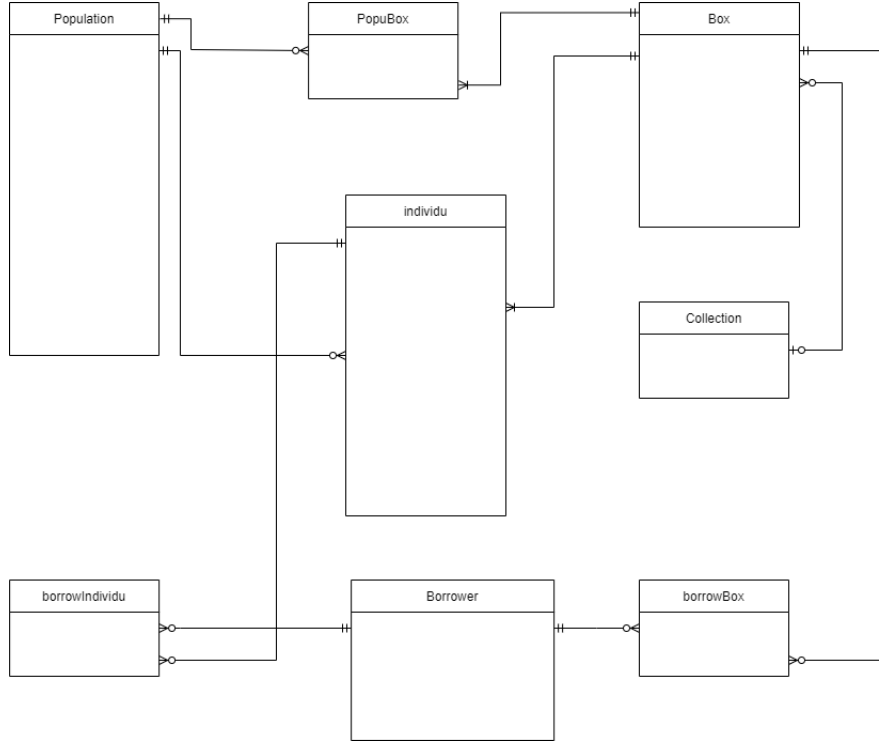


Figure 3.3: Right part of the conceptual model

The *Box* table is in relation with the *Population* table because a box can have one or many populations and a population can belongs to several boxes. This link is made with an additional table *PopuBox*. A box can also belong to a *Collection*, this is the link with *Collection* table. A box can also be loaned that's why it is connected with the *borrowBox* table. Finally, the *Box* table have a relationship with the *individu* table as a box can contain several individual.

The *individu* table have a relation with the *borrowIndividu* table in the case only specific individual are loaned. The tables *borrowIndividu* and *borrowBox* are in relation with the table *Borrower*.

### 3.1.4 Logical model

This kind of model represents the structure of the model, it takes the different tables and relationship from the conceptual model, but it adds the different keys to



*subOrder*), *tribu\_id* (*id\_tribu* in table *Tribu*), *family\_id* (*id\_family* in table *Family*), *subFamily\_id* (*id\_subfamily* in table *subFamily*), *genus\_id* (*id\_genus* in table *Genus*), *subgenus\_id* (*id\_subGenus* in table *subGenus*), *species\_id* (*id\_species* in table *species*), *subspecies\_id* (*id\_subspecies* in table *subspecies*).

The different tables *Order*, *subOrder*, *Tribu*, *Family*, *subFamily*, *Genus*, *subGenus*, *species*, *subspecies* have an attribute name.

*Genus*, *subGenus*, *species* and *subspecies* tables have a foreign key called *id\_sc* that refers to *id\_sc* of *Scientific* table in order to retrieve the descriptor. They also have an additional attribute *date* referring the date it was discovered.

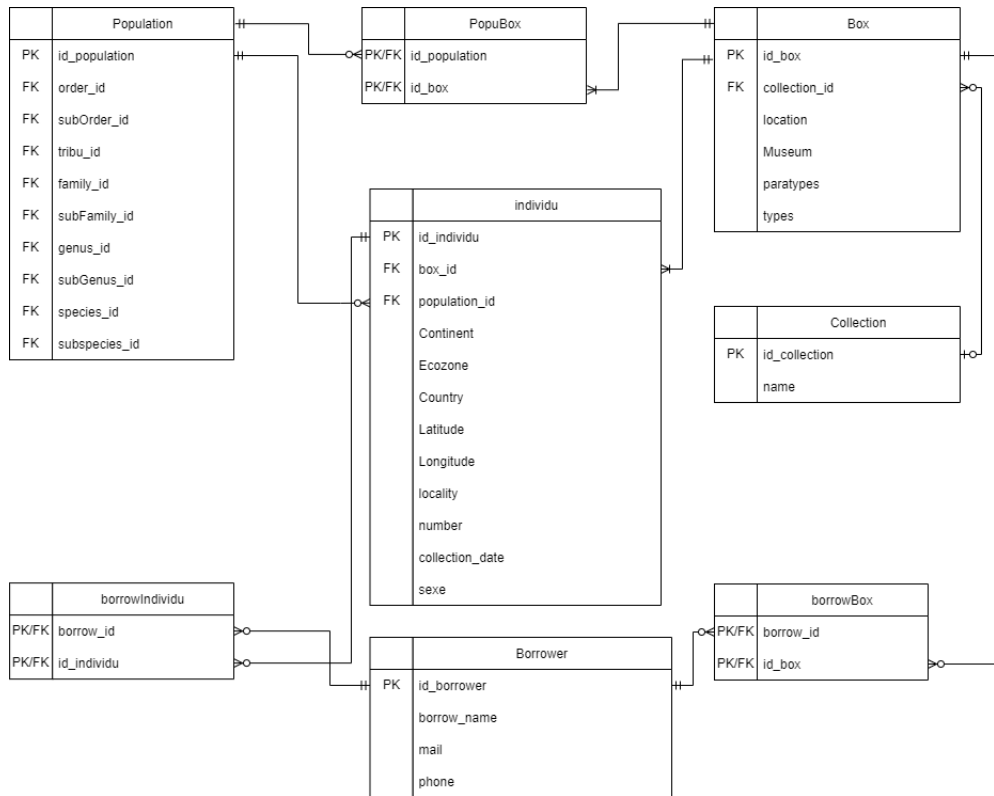


Figure 3.5: Right part of the logical model

*Box* has one foreign key, the *collection\_id* that refers to the *id\_collection* of the *Collection* table. Its primary key is *id\_box* and it has the following attributes which are not part of any keys: *location*, *Museum*, *paratypes*, *types*. It allows to know where the box is located, if it belongs to a museum and the number of paratypes and holotypes a box have.

The *Collection* table has a primary key `id_collection` and an attribute `name` to identify the owner of the collection.

*Individu* table has two foreign keys: `box_id` (referring to `id_box` from *Box* table) and `population_id` (referring to `id_population` from *Population* table). Its primary key is `id_individu` and it has the following attributes which are not part of any keys: `Continent`, `Country`, `Ecozone`, `Latitude`, `Longitude`, `locality`, `number`, `collection_date` and `sexe`.

We also have a table *borrowBox* and a table *borrowIndividu*. *borrowBox* has for attributes `borrow_id` (foreign key referring to `id_borrower` from *Borrower* table), `id_box` (foreign key referring to `id_box` from *Box* table). *borrowIndividu* has for attributes `borrow_id`, `id_individu` (foreign key referring to `id_individu` from *Individu* table). For both, combination of all attributes constitutes the primary key. The table *Borrower* has a primary key `id_borrower` and three more attributes: `borrow_name`, `mail` and `phone` which identify the information of the loaner.

### 3.1.5 Implementation choice

During the creation of this database several choices had to be made. At times, these were made because some data were not relevant, such as several unique ids that identified boxes. In our case one was enough and the choice was to take the number. Then, following a lack of data concerning the individuals, it was decided to attribute to a box one or more populations. In fact, this does not really make sense as a box can have several populations and therefore it will make the search more complex. It would have been preferable to assign populations to each individual as these are identified by a single population. This is why in the database there is a *PopuBox* table which allows a population to belong to one or more boxes and a box to have one or more populations. This point will be discussed in the possible future works part. Finally, even if some data were empty in the entomological museum files, it was decided to add them if they are added in the near future.

## 3.2 User Interface Design

### 3.2.1 Key principles

The user interface is one of the key elements that determine the success of a product. An effective user interface must be easy to use and understand, and must meet the needs of the users.

## **Understanding user needs**

Before creating a user interface, it is important to understand the needs and goals of the users. This can be accomplished by using research methods such as surveys, interviews, and user testing. This understanding of user needs will help design an effective user interface that meets their expectations. [13]

## **Simplify navigation**

Navigation is a key element of any user interface. It is important to ensure that navigation is simple and intuitive for the user. This can be achieved by using drop-down menus, clearly labeled buttons, and consistent organization of information. When designing the user interface, it is important to focus on usability rather than complexity. [14]

## **Use appropriate colors and use clear and readable typography**

Typography is another key element of the user interface. It is important to use clear and legible typography to ensure that the user can easily read and understand the information presented. It is also important to use appropriate colors that reflect the utility of the site but which do not interfere with the readability of the user interface. [15]

## **Testing the user interface**

It is important to test the user interface before deploying it to end users. User testing can help identify usability issues and ensure that the user interface is intuitive and easy to use. Testing can also help identify accessibility issues for users with different abilities. [15]

### **3.2.2 Analysis of needs**

For our user interface to be complete it is necessary that several elements are present.

It is necessary to be able to recover boxes and individuals, both without search criteria and with search criteria (Order, sub-order, etc...).

It is also necessary to be able to retrieve the details of a box (collections, borrower, attributes, ...) or of an individual (box to which it belongs, borrower, attributes, ...). The information must be modifiable by the users, the interface of the detail's pages must therefore be different whether you are logged out, logged in as entomologist or as administrator.

Information can also be added from a .xlsx file, so you need a page where authenticated users can upload these files to the back-end.

Administrators also need an administration page that will allow to create user accounts, modify the rights of these users, create and delete new attributes (orders, sub-orders, ...) but also manage the information of borrowers.

Finally, for users to authenticate, a login page is required. Once logged in, the user must also be able to change their password.

### 3.2.3 User Interface choices

Following figures are the sketches of what the various pages should look like at a minimum to meet the needs of the entomology team.

Some pages have the same type of needs and therefore share the same primitive design. Among them, three categories stand out:

1. Information retrieval pages (Search pages) (with selection controls, search button, table)

Figure 3.6: Design of information retrieval pages

2. Inputs pages (Admin panel, login page) (with containers, button to trigger

actions, input components)

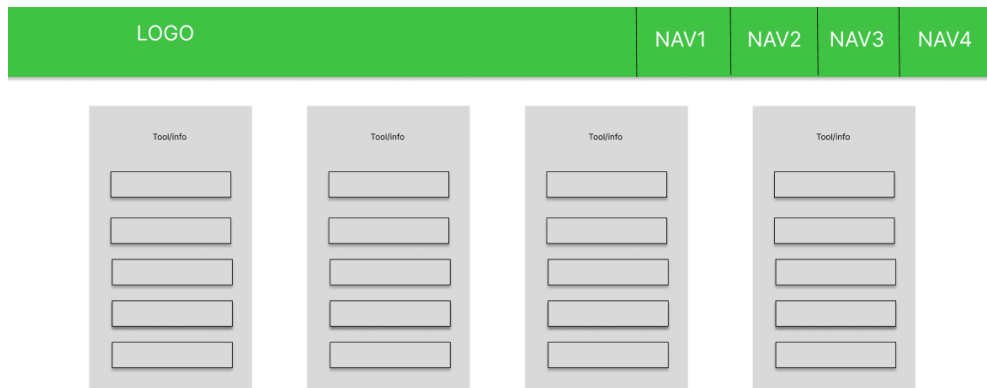


Figure 3.7: Design of inputs pages

3. Pages with information and an admin part (Details of a box/individual page)  
-with table, containers, button to trigger actions, input components)

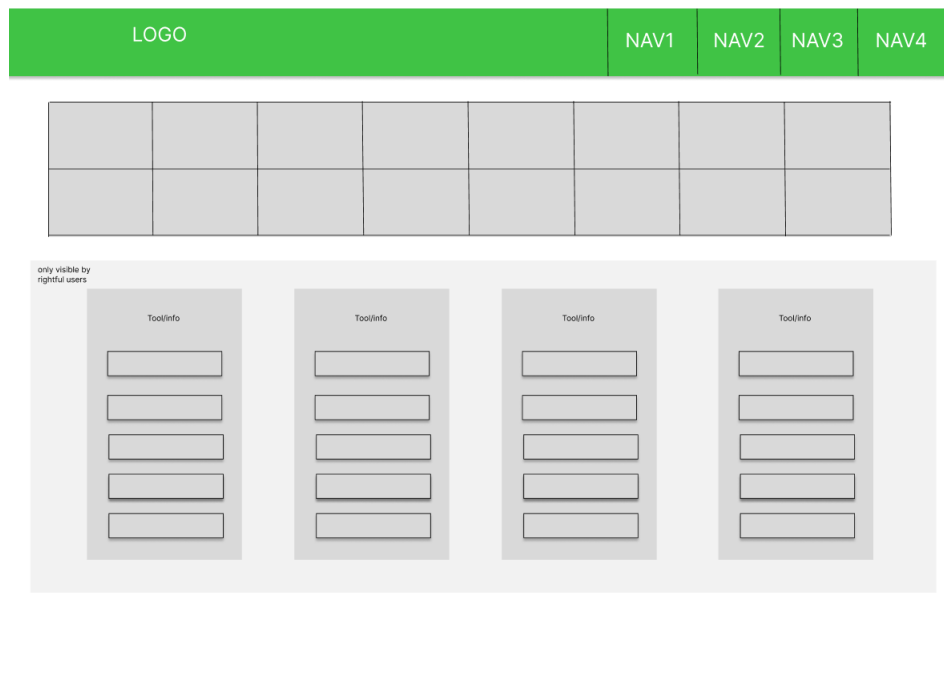


Figure 3.8: Design of details pages

## 3.3 System Architecture

### 3.3.1 Microservices architecture with Docker (containerization)

Containerization is a virtualization technique for creating lightweight, portable containers that are runtime units for applications. These containers are isolated from each other and have their own operating systems and libraries. They thus solve the problems related to the management of dependencies and compatibility between applications and their environment.[16] Docker is one of the most popular tools for creating and managing containers.

#### Docker

Docker is an open-source containerization platform that allows you to create, deploy and manage applications in containers. [17] It is based on Linux container technology, which allows an application's processes to be isolated in a container without using a virtual machine. Docker uses images to create containers, which are file system templates containing all the elements needed to run an application.

#### Advantages

Containerization has many advantages:

- Improved portability:  
containers can be run on any compatible system, making it easy to migrate an application from one environment to another.[17]
- Enhanced isolation:  
each container is isolated from the others and has its own resources, avoiding conflicts between applications and enhancing security.[17]
- Greater flexibility:  
Containers allow applications to be deployed quickly and in an automated manner, facilitating the adoption of DevOps practices. [18]
- Better resource management:  
containerization optimizes the use of hardware resources, sharing resources between containers and avoiding waste of unused resources. [18]



### 3.3.2 Efficient division of microservices

Efficiently dividing up microservices is crucial to ensuring the flexibility and scalability of an application's architecture. This allows developers to work on different services in parallel, which can speed up the delivery of new features. In this section, we'll look at best practices for effectively splitting microservices.

- Identify the distinct business domains:  
The first step in effectively dividing microservices is to identify the distinct business domains of the application. This will help create microservices that are highly consistent and address specific user needs. It is important not to create microservices that are too small, as this could lead to excessive complexity.[19]
- Promote service independence:  
Microservices should be designed to be independent of each other, with minimal communication between them. This allows teams to develop and deploy services independently, which can reduce dependencies between services. Application programming interfaces (APIs) should be standardized and communications should be limited to simple messages.[19]
- Reduce data duplication:  
It is important to reduce data duplication between microservices to ensure data consistency and integrity. It is therefore good to try, as much as possible, to have microservices using databases that are independent of each other. [20]

### 3.3.3 Practical example

#### Analysis of needs

For our application we needed:

- A login system, allowing the connection and registration of users, but also the different roles they can have (user or administrator). This login system must also allow authentication through tokens for security reasons.
- To be able to perform operations on the entomological database. This includes: adding data (from dataframes or files), updating data, deleting data, retrieving data (as a dataframe or as a file).
- To be able to upload files. We are talking about files that are not sensitive to changes in the entomological database, such as templates.

- To be able to retrieve pictures from a NAS (Network Attached Storage).
- To have a graphical interface accessible to users in order to launch operations.
- A way to communicate between this graphical interface and the different back-end services without the need to open the ports of these back-end services (for security reasons). This will also allow to have an overview of all possible requests to the back-end. In addition to all this, a Gateway also allows pre-configured redirection of requests in case of unavailability of the servers usually used.

## Setting up

Six microservices seem to be necessary for our application:

- **Front-end:**  
Manages the presentation aspects of the web application. It is responsible for the user interface, user event management and interaction with the gateway.
- **Gateway:**  
Receives the requests that the user sends through the interface (front-end) and transmits them to the various microservices. The gateway is also responsible for receiving the response from the microservices and retransmitting it to the user. The gateway acts as a reverse-proxy, in the sense that it keeps the digital identity of the various microservices hidden. The user can only communicate with the Gateway.
- **Login:**  
This service is used for the connection, the registration, the recovery of the user data but also the authentication by token. It has its own database (db-login).
- **DBOps (DB Operations):**  
This service is responsible for operations requiring queries to the entomological database. These operations being numerous it would be possible to split them into different categories and thus have more microservices. For the moment this single microservice seems to be sufficient for this type of operations. It has its own database (db-entomo).
- **FileDownloader:**  
Responsible for the transmission of files to the user. These files are established in advance, are stored in the microservice and are not sensitive to changes in the various databases.

- PicturesNFS:  
Retrieve pictures thanks to NFS (Network File System) protocol. The NAS is in the local network of the web server.

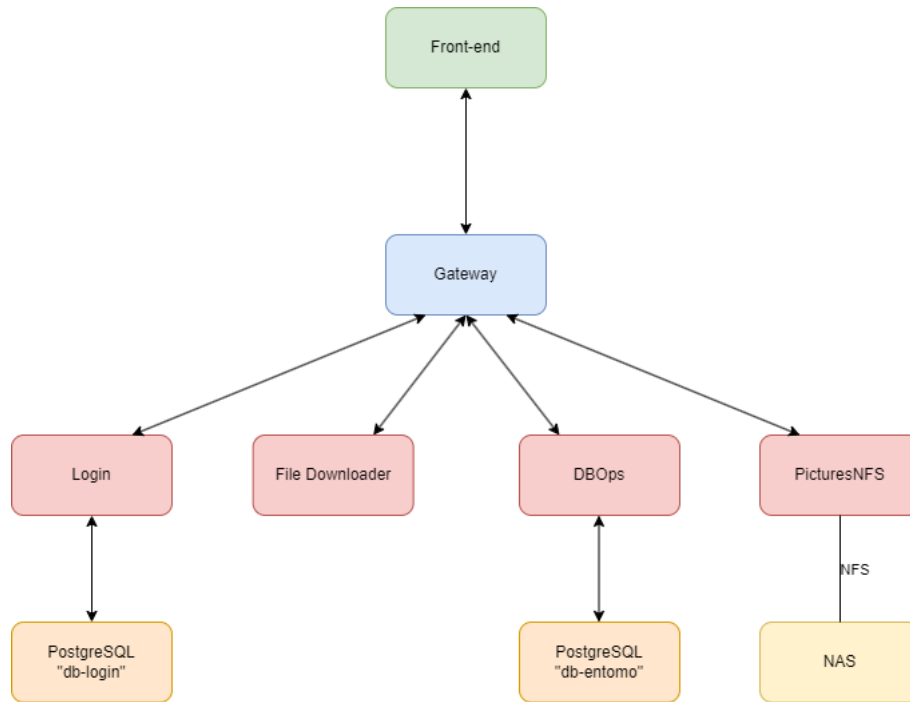


Figure 3.9: Architecture of our microservices system

The client can only retrieve the front-end interface and send requests to the API Gateway. This API Gateway then dispatches the requests to the different back-end microservices for which it will retrieve the response to the requests and send it back to the client.

Each microservice using a database has its own database. Besides security reasons, this is also useful for data persistence. [20]

# Chapter 4

## Implementation

### 4.1 Databases Implementation

#### 4.1.1 Physical model

This model represents the last model of the database design, in comparison with the previous one from the previous chapter, this one will mention the different types of data and if they can be null or not. At the moment this model has been validated, the database can be created using it as reference. It is not really interesting to go in details for this model in this section as it will be the implementation one. However, it should still be mentioned that the different date has been chosen to be encoded as varchar value. This has been done in order to give the user the choice to encode the dates in the way they wish.

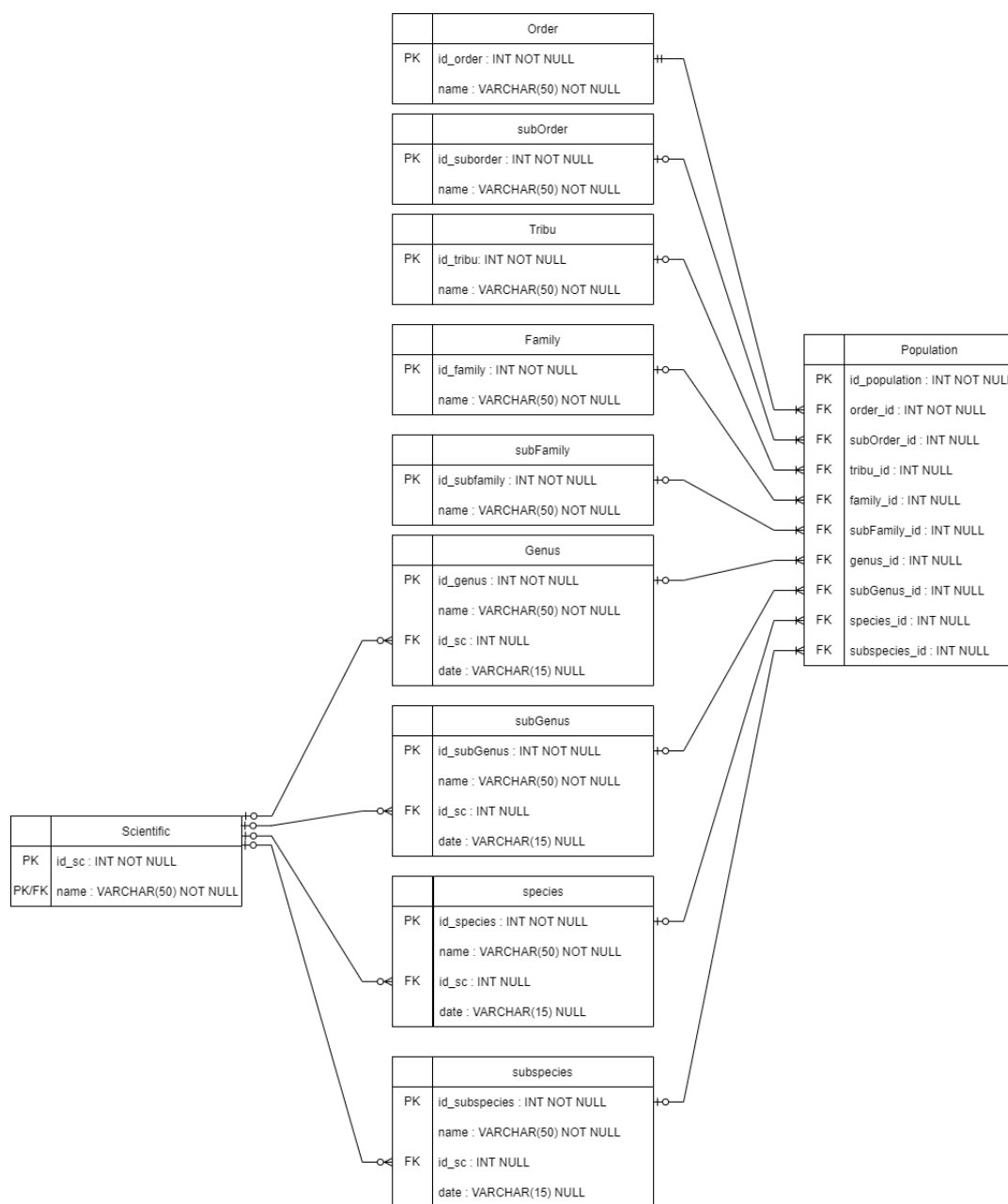


Figure 4.1: Left part of the physical model

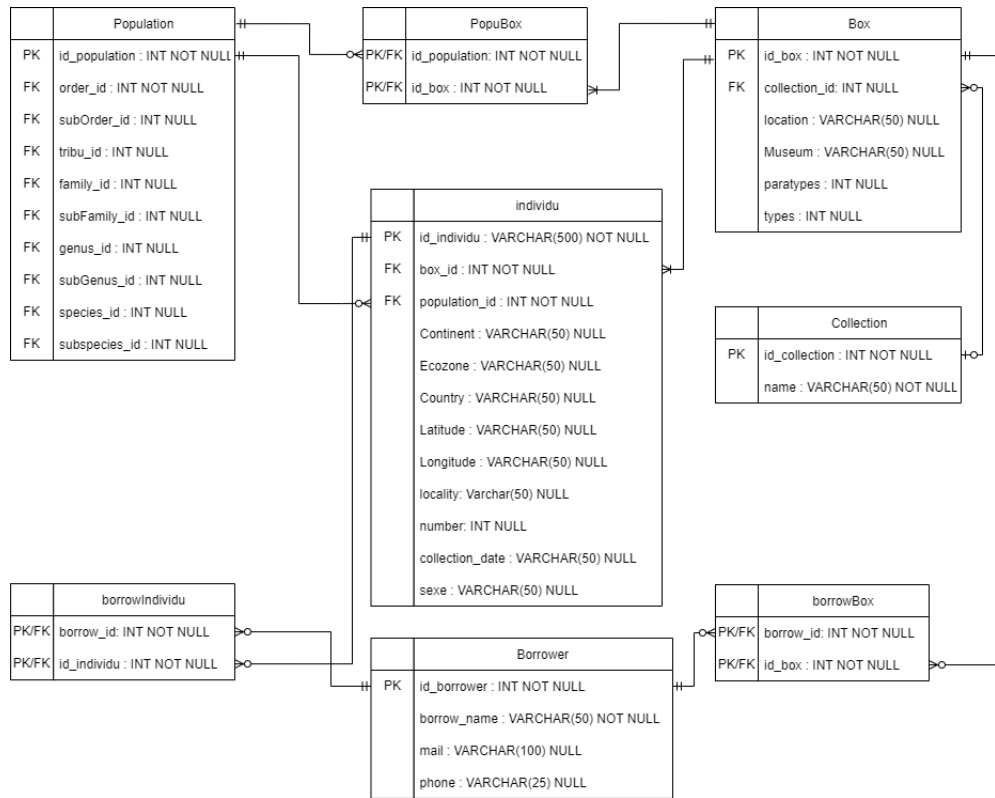


Figure 4.2: Right part of the physical model

## 4.1.2 Entomological database

### Creation of the tables

Thanks to the physical model above, the creation of tables can be done easily using this syntax. [21]

---

```

CREATE TABLE "table_name" (
  "attribute_pk" TYPE NOT NULL,
  "attribute1" TYPE NOT NULL,
  "attribute_fk" TYPE,
  "attribute2" TYPE,
  PRIMARY KEY ("attribute_pk"),
  UNIQUE ("attribute_pk"),
  UNIQUE ("attribute1"),
  FOREIGN KEY("attribute_fk") REFERENCES "table_fk"("attribute_fk_pk") ON
    DELETE CASCADE
);
  
```

---

The first line corresponds to the name of the table that should be created. The four following lines are for the different column of the table. In this example we can see that there are four attributes, one for the primary key, one for the foreign key and two to other for additional attribute of the table. Each attribute has a specific type and if they are mandatory they must be followed by `NOT NULL`. The last line of the creation is for the constraints on the different attributes. The primary key constraints can have one or several attributes and specify that these attributes are unique and not null. The unique constraints specify that this attribute is unique. The foreign key constraint specifies which attributes of the table reference an attributes from another table. The `ON DELETE CASCADE` can be added to this constraints in order to deleted the lines of this table when the same id is delete in the referenced table.

Now we can look of the different tables created in our database

---

```
CREATE TABLE "Genus" (  
  "id_genus" INT NOT NULL,  
  "name" VARCHAR(500) NOT NULL,  
  "id_sc" INT,  
  "date" VARCHAR(15),  
  PRIMARY KEY ("id_genus"),  
  UNIQUE ("id_genus"),  
  UNIQUE ("name"),  
  FOREIGN KEY("id_sc") REFERENCES "Scientific"("id_sc")  
);
```

---

In this SQL query, a table Genus is created, this contains a specific if that identify a unique genus with a unique name. This table has a foreign key attribute `id_sc` that reference the primary key of the table Scientific. This relationship can be seen in the different diagram. The last attribute of this table is the discovery date of the genus. The other of the classification are not explained here but are present in **appendix A**. These tables are compiled in the same way with the exception of certain attributes which are not present in certain tables as can be seen in the models.

---

```
CREATE TABLE "Population" (  
  "id_population" INT NOT NULL,  
  "order_id" INT NOT NULL,  
  "suborder_id" INT,  
  "tribu_id" INT NULL,  
  "family_id" INT,  
  "subFamily_id" INT,
```

```

"genus_id" INT,
"subGenus_id" INT,
"species_id" INT,
"subSpecies_id" INT,
PRIMARY KEY ("id_population"),
UNIQUE ("id_population"),
FOREIGN KEY("order_id") REFERENCES "Order"("id_order"),
FOREIGN KEY("suborder_id") REFERENCES "subOrder"("id_suborder"),
FOREIGN KEY("family_id") REFERENCES "Family"("id_family"),
FOREIGN KEY("subFamily_id") REFERENCES "subFamily"("id_subfamily"),
FOREIGN KEY("genus_id") REFERENCES "Genus"("id_genus"),
FOREIGN KEY("subGenus_id") REFERENCES "subGenus"("id_subgenus"),
FOREIGN KEY("species_id") REFERENCES "Species"("id_species"),
FOREIGN KEY("subSpecies_id") REFERENCES "subSpecies"("id_subspecies"),
FOREIGN KEY("tribu_id") REFERENCES "Tribu"("id_tribu")
);

```

```

CREATE TABLE "Box" (
"id_box" INT NOT NULL,
"collection_id" INT,
"location" VARCHAR(50),
"museum" VARCHAR(50),
"paratypes" INT,
"types" INT,
UNIQUE("id_box"),
PRIMARY KEY ("id_box")
);

```

```

CREATE TABLE "PopuBox" (
"population_id" INT NOT NULL,
"box_id" INT NOT NULL,
PRIMARY KEY("population_id", "box_id"),
FOREIGN KEY ("population_id") REFERENCES "Population"("id_population"),
FOREIGN KEY ("box_id") REFERENCES "Box"("id_box") ON DELETE CASCADE,
CONSTRAINT "UC_PopuBox" UNIQUE ("population_id", "box_id")
);

```

---

These queries create the population, box, and PopuBox tables. It's quite interesting to take a closer look at these tables. First, we can look at the population table and we notice that the attribute that identifies a row is the id. All other attributes are foreign keys that reference the different ids of the classification tables. It is important to note that a population must have at least one order. In the box table, there is also a unique id that identifies each box and several other



attributes that are linked directly to a box. This table also has a foreign key that links the box to a certain collection.

As explained above, a box can have several populations and a population can belong to several boxes, which is why the PopuBox table only contains two attributes that are foreign keys but the combination of these also gives the primary key of that box.

A final table that is also interesting in its complexity is the individual table (**appendix A**). When we look at the entity in the diagram, we notice that it has several foreign keys and several attributes specific to each individual. .

## Filtering the different existing excel files

Once all the table have been created, it is the time to fill the database with the data you already have. However, an important point to pay attention is to be rigorous between the data you have and how you will insert it in the database. In our case, we have a excel file with a large quantity of data. The first step for all of these is to verify that the excel file have the correct column and then to analyze each cell to be sure that the insertion would not create error. If these steps are not done, you either assume that the user never make bad encoding or you will got error during the inserting phase. The different rules for filling in the excel can be seen in the (**appendix B**).

---

```
def filterExcel(olddf):

    checkCol = ['Num_ID', 'Order', 'Suborder', 'Family', 'Subfamily',
                'Tribu', 'Genus', 'Genus_Descriptor', 'Genus_Date', 'Subgenus',
                'Subgenus_Descriptor', 'Subgenus_Date', 'species',
                'Species_Descriptor', 'Species_Date', 'Subspecies',
                'Subspecies_descriptor', 'Subspecies_Date', 'Types',
                'Paratypes', 'Museum', 'Box_Localization', 'Collection_Name']
    colname = []
    for col in olddf.columns:
        if col not in checkCol:
            return [],1,pd.DataFrame, ["pas les bonnes colonnes"]
        else:
            colname.append(col)
    for col in checkCol:
        if col not in colname:

            return [],1,pd.DataFrame, ["pas les bonnes colonnes"]

    df = olddf[colname].copy()
    gooddf = pd.DataFrame(columns = colname)
```

```

count = 0
reasons = []
bad = []
for i, row in df.iterrows() :
    ...

```

---

The function filterExcel is the main function of the filtering. It will filter all the row of the excel. It takes in argument a pandas.DataFrame of the excel file and it returns a tuple of size 4:

- bad - list of the bad index in the excel file
- count - number of bad encoded lines
- goodf - pandas.DataFrame of the well encoded row
- reason - list of reason for each bad encoding

First of all, the function will check if the column of the excel are the good one and there are no missing or new column. It will then iterate through all the row of the Dataframe and each cell will be analyze as explained below.

---

```

"""
MandatoryValue is a function that will check whether or not the
    mandatory cell of the excel have certain value

:param: name = name of the specific cell (eg: row.Column)
:param: error = string error of the column name
:return: True if there is an error
"""
def MandatoryString(name, error):
    if not isinstance(name, str):
        if math.isnan(name):
            bad.append(i+2)
            reasons.append("There is no " + error)
            return True
        else:
            return False
    ...
for i, row in df.iterrows() :
    if not float(row.Num_ID).is_integer() or math.isnan(row.Num_ID):
        bad.append(i+2)
        reasons.append("Box id")
        continue
    if(MandatoryString(row.Order, "order")):

```

```

        continue
    if(MandatoryString(row.Collection_Name, "collection")):
        continue
    ...

```

---

At this point we can start by filtering each of the lines of the excel file. The first interesting check to make is to verify that the mandatory values are respected in the excel. In this example , there is three column that need to be mandatory. The Num\_ID must be an integer, the Order must be a string and the Collection\_name must be also a string. The math.isnan is there to force that these values should not be empty.

---

```

....
for i, row in df.iterrows() :
    ...
    if (isinstance(row.Order, str)): order = row.Order.split("_")
    else: order=[""]
    if (isinstance(row.Suborder, str)): suborder =
        row.Suborder.split("_")
    else: suborder=[""]
    ...
    if(len(order)>1 and (len(suborder)>1 or len(family)>1 or
        len(subfamily)>1 or len(tribu)>1 or len(genus)>1 or
        len(subgenus)>1 or len(species)>1 or len(subspecies)>1)):
        bad.append(i+2)
        reasons.append("several order and several sub
            classification")
        continue
    if(len(suborder)>1 and (len(family)>1 or len(subfamily)>1 or
        len(tribu)>1 or len(genus)>1 or len(subgenus)>1 or
        len(species)>1 or len(subspecies)>1)):
        bad.append(i+2)
        reasons.append("several suborder and several sub
            classification")
        continue
    ...
    ...

```

---

The entomologist of the Walloon Conservatory has a specific notation when a box have several classification, they separate them using a "\_". During the design of the database model and due to the lack of data concerning the individuals, the decision was made to link a population to a box and to an individual. This cause a problem to store efficiently the population, that's why in this filtering we check if

there is several classifications for the same scale, it cannot have sub-classification for this boxes. The example above shows only for several order and suborder but there is of course for the other classification.

---

```

"""
StringValue is a function that will check whether or not the cell of
the excel is a string or not

:param: name = name of the specific cell (eg: row.Columnn)
:param: error = string error of the column name
:return: True if there is an error
"""
def StringValue(name, error):
    if not isinstance(name, str) and not (name != name) :
        bad.append(i+2)
        reasons.append(error +" is a number")
        return True
    else:
        return False
...
for i, row in df.iterrows() :
    ...
    if(StringValue(row.Suborder, "SubOrder")):
        continue
    if(StringValue(row.Family, "Family")):
        continue
    if(StringValue(row.Subfamily, "SubFamily")):
        continue
    if(StringValue(row.Tribu, "Tribu")):
        continue
    ...

```

---

This part of the code is to assume that order, suborder, family, subfamily, tribu, genus, subgenus, species and subspecies are not a number. The same is done for the different descriptors, the museum, the localization of the box and the name of a collection.

---

```

"""
IntValue is a function that will check whether or not the cell of
the excel is a int or not

:param: name = name of the specific cell (eg: row.Columnn)
:param: error = string error of the column name

```

---

```

:~return: True if there is an error
"""
def IntValue(name, error):
    if isinstance(name, str) and not (name != name):
        if not any(char.isdigit() for char in name):
            bad.append(i+2)
            reasons.append(error + " is not a number")
            return True
        else:
            return False
    ...
for i, row in df.iterrows() :
    ...
    if(IntValue(row.Paratypes, "paratypes")):
        continue
    if(IntValue(row.Types, "types")):
        continue
    ...

```

---

Here we check that the types and paratypes are really encoded using integer and not with character value.

---

```

1  """
2  filter_values filter the value to convert a float to a integer if
   possible.
3  if not possible:  -return None (if it is a float)
4                    -return val (if it is a string)
5
6  :param: either a float, integer or a string
7  :return: the int(value), None, val
8  """
9  def filter_values(val):
10     if isinstance(val, float):
11         if val.is_integer():
12             return int(val)
13         elif math.isnan(val):
14             return None
15         else:
16             return "BadEncodingDate"
17     else:
18         return val
19     ...
20  for i, row in df.iterrows() :

```

```

21     ...
22     if(filter_values(row.Genus_Date)=="BadEncodingDate"):
23         bad.append(i+2)
24         reasons.append("date of the genus Descriptor is not a
                integer")
25         continue
26     else :
27         row.Genus_Date= filter_values(row.Genus_Date)

```

---

For the last check, the previous choice of implementation was that a date is encoded as string. In this function we try to convert possible float into integer but if it cannot it should return an error. Otherwise, it returns the value.

This script is used to filter the different boxes, another script should be used to filter the different individuals. The two scripts are quite the same except for some attributes that differs. Therefore, it is not useful to go in detail for this part of the script. One thing to mention is that an individual cannot have several populations and it should be mentioned in the filtering.

## Filling the database

At this point, all the data has been filtered and can be insert in the database. If it is an administrator that upload the data on the website some of them can therefore be update. Before looking at the different insertion script it is important to understand how the insertion works. [21]

---

```

INSERT INTO "table_name"
("attribute1", "attribute2")
VALUES
(Values1, Values2)

UPDATE "table_name"
SET attribute1=Values3
WHERE attribute2 = Values2

```

---

The insertion is made by designing in which table and in which column the new values need to be inserted. Concerning the update query, the table and the attribute are also mentioned but it can be updated using some condition whit the where clause.

---

```

1  """
2  insertName is a function that will insert in a database in the
    corresponding table the name and the id
3

```

```

4 :param data: a pandas dataframe
5 :param cursor: cursor to traverse the result of SQL query
6 :param conn: represent the connection to the database
7 :param columnName: the name of the column from the dataframe needed
8 :param tableDB: the name of the table where the data need to be encoded
9 :param id_DB: the name of the id of the corresponding table from the
    database
10 """
11 def insertName(data, cursor, conn, columnName, tableDB, id_DB) :
12     toinsert = data[columnName].values.tolist()
13     duplicationquery = sql.SQL("select MAX({field}) from
        {table}").format(
14         field=sql.Identifier(id_DB),
15         table=sql.Identifier(tableDB))
16     cursor.execute(duplicationquery)
17     result = cursor.fetchall()
18     Count = 1
19     if result != [(None,)] :
20         Count = result[0][0]+1
21     for i in range(0, len(toinsert)):
22         if isinstance(toinsert[i], str) : orderList =
            toinsert[i].split("_")
23         else : orderList = ["NULL"]
24         for index in orderList:
25             if index=="NULL": continue
26             duplicationquery = sql.SQL("""SELECT *
                FROM {table}
                WHERE "name" = (%s) """).format(
27                 table=sql.Identifier(tableDB))
28             dataexistquery = ( index, )
29             cursor.execute(duplicationquery, dataexistquery)
30             if cursor.fetchall() == [] :
31                 insertquery = sql.SQL(""" INSERT INTO {table} ({field},
                    "name")
32
33                                     VALUES
34                                     ((%s), (%s))
35                                     """).format(
36                                     field=sql.Identifier(id_DB),
37                                     table=sql.Identifier(tableDB))
38                 datainsertquery = (Count, index)
39                 cursor.execute(insertquery, datainsertquery)
40                 Count+=1
41     conn.commit()
42

```

---

Here is a basic example of a script that will insert specific classification that are not link with a scientific. This code will first take the maximum id from the table to know at which index it could insert the new value. After that it loop over a specific column name from the dataframe filled during the filtering. For each name, it will verify if they are already present in the database or not. If they are not, it will insert this new value with the new index calculated before. [22]

---

```

1 elif(admin):
2
3     deletePopuBox= """
4         DELETE
5         FROM "PopuBox"
6         WHERE "box_id"=(%s) """
7     datadeletePopuBox = (toinsertID[i],)
8     cursor.execute(deletePopuBox, datadeletePopuBox)
9     for pop in populationList:
10         duplicate = """SELECT *
11             FROM "PopuBox"
12             WHERE "box_id" = (%s) and "population_id" =
13                 (%s)"""
14         dataduplicate = (toinsertID[i], pop)
15         cursor.execute(duplicate, dataduplicate)
16         if(cursor.fetchall()==[]):
17             insertPopuBox = """INSERT INTO "PopuBox"
18                 ("population_id", "box_id")
19                 VALUES
20                 ((%s), (%s)) """
21             datainsertbox = (pop, toinsertID[i])
22             cursor.execute(insertPopuBox, datainsertbox)
23         collection = """ SELECT "id_collection"
24             FROM "Collection"
25             WHERE "name" = (%s) """
26         datacollection = (toinsertCollection[i],)
27         cursor.execute(collection, datacollection)
28         collectionList = cursor.fetchall()
29         insertquery = """UPDATE "Box"
30             SET "collection_id" = (%s) ,"location" = (%s),
31                 "museum"= (%s), "paratypes" = (%s),
32                 "types"= (%s)
33             WHERE "id_box" = (%s) """
34         datainsertquery = ( collectionList[0][0],
35             toinsertLocation[i], toinsertMuseum[i],
36             toinsertParaType[i], toinsertType[i],toinsertID[i])

```



When an admin runs the scripts, a certain flag will be mentioned to allow overwrite. This will then allow specific information about a box or an individual to be updated. In the script above, we can see that there will first be a deletion in the PopuBox table, indeed since an update will take place, some populations might not be part of a specific box anymore. Then, as in the other scripts, new rows are added if they do not already exist. To finish by updating the information of the box, without removing it. These steps are slightly different for modifying the information of an individual but very similar.

During these updates by an admin, if a new classification is discovered, it is added as before. The big difference here is the update of a box by these new attributes.

### 4.1.3 Login database

Our application having a login system, it was necessary for this system to be functional to have a Login database containing the information of the user (name, password -hashed obviously-, and its role) as well as the information of the various tokens of session (for questions of authentication during certain operations).

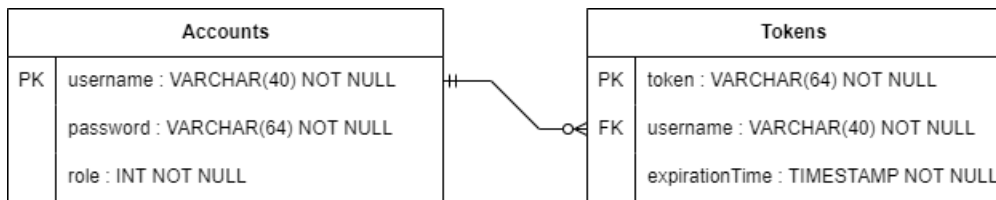


Figure 4.3: Physical diagram of Login database

## 4.2 Website Development

### 4.2.1 Docker, Dockerfile and Docker-compose

To reiterate, Docker is a freely available software for developing, launching, and maintaining apps in enclosed runtime environments known as containers. Containers are self-contained environments that consist of application runtimes, dependencies, and configuration files, among other things.[16]

#### Dockerfile

To create containers, Docker uses a series of instructions that are defined in a file called Dockerfile. The Dockerfile is a text file that contains all the instruc-

tions needed to assemble a Docker image. Images are the "source code" of our container.[17]

The format of Dockerfile is pretty simple and is composed of several instructions that allow you to build and customize the image.

Some of the most common instructions are the following[23] :

- FROM: this instruction specifies the base image to use to build the current image.
- RUN: this instruction allows to execute a command in the image being built.
- COPY: this instruction allows you to copy files from the local file system to the image being built.
- WORKDIR: this instruction allows to define the working directory for the following instructions.
- CMD: this instruction is used to define the default command to be executed when the container is started.

Once the Dockerfile has been created, it can be used to build a Docker image by running the docker build command. This command reads the Dockerfile and creates a Docker image based on the instructions it contains. The resulting image can then be used to create Docker containers thanks to the docker run command. But in our case, we use another solution than the docker run command to deploy containers: docker-compose.

Our front microservice Dockerfile is really interesting to analyze because of its use of a base image to build another image.

---

```
1 FROM node:18.14.1-alpine3.16 as build
2
3 WORKDIR src/front-end
4
5 COPY package*.json .
6 RUN npm install -g npm@latest
7 RUN npm install browserslist
8
9 COPY public public
10 COPY src src
11 COPY nginx.conf .
12
13 #production
14 RUN npm run build
```

```
15
16 FROM nginx:stable-alpine
17 COPY --from=build /src/front-end/build /usr/share/nginx/html
18 COPY --from=build /src/front-end/nginx.conf
    /etc/nginx/conf.d/default.conf
19 EXPOSE 80
20 CMD ["nginx", "-g", "daemon off;"]
```

---

The first line indicates that we use as a starting point for the creation of the image, the official image `Node.js 18.14.1-alpine3.16` .

With the line `WORKDIR src/front-end` we define the working directory of the Docker container where the front-end application will be installed.

In addition to copying the files necessary for the container to function properly, we must also update npm to the latest version with `"RUN npm install -g npm@latest"`. Finally for this image called "build" we run `"npm run build"` to build the front-end application in production mode.

Now we have to use the official `nginx:stable-alpine` image as a base for the second build step. We need to copy the front-end application files generated in the previous step from the build image into the `nginx` image.

Port 80 is then exposed to allow access to the application via HTTP.

The default command for the Docker container is then set: `CMD ["nginx", "-g", "daemon off;"]`. This launches the nginx web server to run the front-end application.

## Docker-compose

This technology allows to define the services, the volumes (storage) necessary for the execution of microservices but also the dependencies between these microservices in a YAML file. This simplifies the management of complex applications by automating deployment, scaling and dependency management.[24]

The docker-compose file includes several sections, the most important of which are the following:

- **version:** this section specifies the version of the docker-compose syntax to use.
- **services:** this section specifies the different services to create for the application. Each service is described in a separate section, which contains information about the image to use, the ports, etc.
- **volumes:** this section specifies the volumes to use to store persistent data.

The YML file used for this project is written like this:

---

```
1 version: '3'
```

---

The docker-compose.yml file starts with the YML file version we want to use. The version 1 is very limited and does not allow to define some parameters such as external volumes or custom networks is not supported anymore. Version 2 introduced the notions of networks and external volumes. Version 3 is the last one and allows to define the resources of the machine hosting the containers.

It is this last version that will be used in order to allow an easier upgrade of our application.

---

```
3 services:
4   gate :
5     build: back-end/APIGateway/.
6     image: apigate
7     container_name: gatec
8     ports:
9       - "4000:80"
10    links:
11      - dbops
12      - login
13      - dlder
14      - picturenfs
15
16    db-entomo :
17      build: back-end/Database/.
18      image: postgres
19      container_name: db-entomoc
20      ports:
21        - ***** (censored)
22      environment:
23        - ***** (censored)
24      volumes:
25        - DataEntomo:/var/lib/postgresql/data
26      healthcheck:
27        test: pg_isready -U postgres
28
29    dbops :
30      build: back-end/DBOps/.
31      image: dbops
32      container_name: dbopsc
33      ports:
```

```
34     - ***** (censored)
35     links:
36     - db-entomo
37     depends_on:
38         db-entomo:
39         condition: service_healthy
```

---

Above the definition of the services gate (API Gateway), DBOps (Operation on entomological database) and db-entomo (entomological database).

The definition of each service includes information such as the directory where the image is located and the name of the image produced by the Docker-file in this folder (or directly the image if it is an official image, lines 24-25 could have been replaced by `image: postgres:latest`) in our case.

The running containers will have a name, this name is the one defined in `container_name`.

The ports section in just a port mapping between host port and container port (`HOST:CONTAINER`).

Now things get a bit more complex for the `links`, `healthcheck` and `depends_on` sections.

As illustrated in **Figure 3.6: Architecture of our microservices system**, the different microservices can only send requests to certain other microservices chosen in advance.

Let's take the example of the microservice gate. It must be able to communicate with all the back-end microservices in order to be able to send them the requests sent initially by the client.

By linking the containers, we form a shared network that allows the containers to communicate with each other, transmit requests and receive responses through a bidirectional channel.

Let's move on to the `healthcheck` and `depends_on` sections which are, in this project, related to each other.

The command provided in `healthcheck` sets up a test that is run to determine if the containers in this service are "healthy" or not [24]. Thanks to this `healthcheck`, we can create a `depends_on` condition for some containers that have to wait for a database to start. A non-started database at the time of the start of these containers depending on a database would lead to a crash of these containers due to the impossibility of connecting to an unavailable database.

You may have noticed the volume section of db-entomo. Specifying this information

will keep the data persistent in this database. The volumes used in the database must first be defined in a different field than the services: the volume field. The way volumes are used in this project is not very complex. More complete examples are available in the docker-compose documentation [24].

---

```
90 volumes:
91   DataEntomo:
92   DataLogin:
93   (...)
```

---

## 4.2.2 Front-end

### React

To implement the front-end of our application we used React which is a JavaScript library to create user interfaces [25].

When implementing a React application, components are used. A component can be seen as a part of another component or a page. React components can be created using classes or functions, here Class components have been used.

In the classes of this project, there are:

- State variables specific to the class that describe the object and allow to have variables in the display of a component
- props to transmit data between components (these props are passed from parent component to child)
- Methods, including:
  - `render()` which returns the visual elements to be displayed [25]
  - `componentDidMount()` which is called immediately after the component has been mounted [25]
  - Functions allowing to manage the various user actions and the actions that these generate, including:
    - \* Asynchronous functions sending HTTP requests thanks to axios (JavaScript library that allows to make HTTP requests)

In the following example, the FileUploader class is a component with a constructor. This component has two state variables and have props which been transmitted by the parent class thanks to the code that could for example be

---

```
<FileUploader type='Box' path='/csvtosql' />
```

---

in the render function of parent object.

---

```
1 import React, { Component } from 'react';
2 import axios from 'axios'
... (...)
9 class FileUploader extends Component {
10   constructor(props) {
11     super(props);
12     this.state = {
13       selectedFile: null,
14       uploadstate: ''
15     };
16   }
... }
13 (...)
```

---

The `handleFileChange()` and `CsvToSQL()` functions are asynchronous functions launched at the time of specific user actions, namely the drag&drop of a file by the user for `handleFileChange()` and a click on a button for `CsvToSQL()`. While `handleFileChange()` changes the value of the `selectedFile` state variable, `CsvToSQL()` is in charge of sending an HTTP request as well as managing the received response.

U can also see that props variables are used in axios request and thus depends on the values of these props variables.

---

```
16   handleFileChange = (event) => {
...     (...)
19     this.setState({selectedFile: file})
...     (...)
27   };
28
29   CsvToSQL = (event) => {
...     (...)
42     axios.put(`${url}${this.props.path}$?type=${this.props.type},
        formData)
43     .then(response => {
...       (...)
56     })
57     .catch(err => {
...       (...)
74     });
75   }
```

---

Finally, the render function returns the visual elements to be displayed. The render function is coded in jsx which is a language very similar to HTML with some JavaScript.

We see the selectedFile state variable being used as well as the 2 functions explained above

---

```
81     render() {
82         return (
83             <div>
84                 (<label>
85                     {this.state.selectFile}
86                     <input type="file" onChange={this.handleFileChange}
87                         accept=".xlsx" />
88                 </label>
89                 (...))
90             </div>
91         );
92     }
```

---

It is also interesting to talk about how Routing to different pages works in React in order to define the paths (or URLs) of the different pages in the application. In our project it is done in `render()` function in App class of App.js file.

A route takes in props its path and its element which is the page/component it must display.

---

```
97 render() {
98     return (
99         <>
100             {this.state.isLoading ?
101                 (
102                     <></>
103                 ):(
104                     <Router>
105                     <Routes>
106                         <Route path="/" element={<BoxesHome
107                             isAuthenticated={this.isAuthenticated}
108                             isAdmin={this.isAdmin} Logout={this.Logout}/>} />
109                         <Route path="/box-search" element={<BoxesHome
110                             isAuthenticated={this.isAuthenticated}
111                             isAdmin={this.isAdmin} Logout={this.Logout}/>} />
112                         (...)
113                         <Route path="/usersettings" element={<SettingUser
114                             getUser={this.getUser}
115                             ...
116                         />} />
117                     </Routes>
118                 </Router>
119             )
120         </>
121     );
122 }
```

---



```

110             isAuthenticated={this.isAuthenticated}
111             isAdmin={this.isAdmin} Logout={this.Logout}/> } />
112         </Routes>
113     </Router>
114 )
115 }
116 }

```

---

Thanks to this routing the BoxesHome component will be accessible from AppUrl/ and AppUrl/box-search, the SettingUser component will only be accessible from AppUrl/usersettings.

## Style

It was also necessary to style this front-end.

To build and style our front-end we used HTML element (tags) like `div`, `a`, `image` but also MUI. MUI is a library of React components created by Facebook and used to create user interfaces. This library provides ready-to-use components that can be customized to meet different needs.

In this project it is mainly :

- Tables which allow to create dynamic data tables.
- Selects which are drop-down lists that allow the user to select an option from a list.
- Checkbox which can be used to turn an option on or off. In this project it is mainly used in order to change the user interface following user current needs (modification modes, choice of attribute in Tables, etc.).

In addition to MUI, we also used CSS language (in the Navbar.css and App.css files) which allows us to define the location of the components on a page and the form they take.

Thanks to CSS language you can create "style classes" which are applicated to certain components:

---

```

.navbar-container {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 80px;
}

```

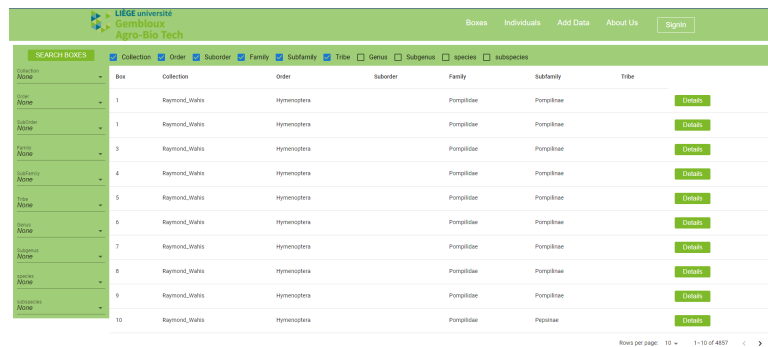
```
width: 100%;
}
```

or apply a style to a certain type of component (here img)

```
img {
  max-height: 80px;
}
```

## Results

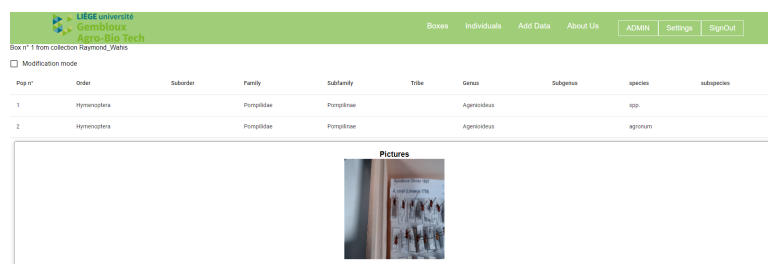
The following pages are the results of an implementation using the technologies described above and following the requirements and designs described in Chapter 3. Some of these pages are also the result of changes made after the user testing phase described in Chapter 5.3.



The screenshot shows a web application interface for 'LIEGE université Gembloux Agro-Bio Tech'. The top navigation bar includes links for 'Boxes', 'Individuals', 'Add Data', 'About Us', and a 'Signin' button. Below the navigation bar is a search filter section with tabs for 'Collection', 'Order', 'Suborder', 'Family', 'Subfamily', 'Tribe', 'Genus', 'Subgenus', 'Species', and 'Subspecies'. The main content area displays a table of search results for 'Box n° 1 from collection Raymond\_Whites'. The table has columns for 'Box', 'Collection', 'Order', 'Suborder', 'Family', 'Subfamily', and 'Tribe'. Each row includes a 'Details' button. The table shows 10 results, all from the 'Raymond\_Whites' collection, belonging to the 'Hymenoptera' order and 'Pompilidae' family. The 'Subfamily' column lists 'Pompilinae' for the first 9 results and 'Peposini' for the 10th result. The 'Tribe' column is empty for all results.

Box	Collection	Order	Suborder	Family	Subfamily	Tribe
1	Raymond_Whites	Hymenoptera		Pompilidae	Pompilinae	
1	Raymond_Whites	Hymenoptera		Pompilidae	Pompilinae	
3	Raymond_Whites	Hymenoptera		Pompilidae	Pompilinae	
4	Raymond_Whites	Hymenoptera		Pompilidae	Pompilinae	
5	Raymond_Whites	Hymenoptera		Pompilidae	Pompilinae	
6	Raymond_Whites	Hymenoptera		Pompilidae	Pompilinae	
7	Raymond_Whites	Hymenoptera		Pompilidae	Pompilinae	
8	Raymond_Whites	Hymenoptera		Pompilidae	Pompilinae	
9	Raymond_Whites	Hymenoptera		Pompilidae	Pompilinae	
10	Raymond_Whites	Hymenoptera		Pompilidae	Peposini	

Figure 4.4: Search Page (C.1)



The screenshot shows a web application interface for 'LIEGE université Gembloux Agro-Bio Tech'. The top navigation bar includes links for 'Boxes', 'Individuals', 'Add Data', 'About Us', 'ADMIN', 'Settings', and 'SignOut'. Below the navigation bar is a search filter section with tabs for 'Collection', 'Order', 'Suborder', 'Family', 'Subfamily', 'Tribe', 'Genus', 'Subgenus', 'Species', and 'Subspecies'. The main content area displays a table of search results for 'Box n° 1 from collection Raymond\_Whites'. The table has columns for 'Box n°', 'Order', 'Suborder', 'Family', 'Subfamily', 'Tribe', 'Genus', 'Subgenus', 'Species', and 'Subspecies'. Each row includes a 'Details' button. The table shows 2 results, both from the 'Raymond\_Whites' collection, belonging to the 'Hymenoptera' order and 'Pompilidae' family. The 'Subfamily' column lists 'Pompilinae' for both results. The 'Tribe' column is empty for both results. The 'Genus' column lists 'Agonotus' for the first result and 'Agonotus' for the second result. The 'Subgenus' column is empty for both results. The 'Species' column lists 'sp.' for the first result and 'sp.' for the second result. The 'Subspecies' column is empty for both results. Below the table is a section titled 'Pictures' which contains a single image of a collection of insects.

Box n°	Order	Suborder	Family	Subfamily	Tribe	Genus	Subgenus	Species	Subspecies
1	Hymenoptera		Pompilidae	Pompilinae		Agonotus		sp.	
2	Hymenoptera		Pompilidae	Pompilinae		Agonotus		sp.	

Figure 4.5: Details Page (C.2)

The screenshot shows the 'Details Page (modification mode)' for 'Box n° 1 from collection Raymond\_Thalm'. The top navigation bar includes 'Boxes', 'Individuals', 'Add Data', 'About Us', 'ADMIN', 'Settings', and 'SignOut'. Below the navigation bar, there's a table with columns: 'Box n°', 'Order', 'Suborder', 'Family', 'Subfamily', 'Tribe', 'Genus', 'Subgenus', 'species', and 'subspecies'. The table contains two rows of data. Below the table, there are several panels for editing:

- Collection:** A dropdown menu for 'Raymond\_Thalm' with 'Edit' and 'Modify' buttons.
- Delete population n°1:** A panel with a 'Delete' button and a 'Modify population n°1' section with dropdowns for Order, Suborder, Family, Subfamily, Tribe, and Genus.
- Delete population n°2:** A panel with a 'Delete' button and a 'Modify population n°2' section with dropdowns for Order, Suborder, Family, Subfamily, Tribe, and Genus.
- Add Population:** A panel with a 'New' button and dropdowns for Suborder, Family, Subfamily, Tribe, Genus, and Subgenus.

Figure 4.6: Details Page (modification mode) (C.3)

The screenshot shows the 'About Us' page. The top navigation bar is the same as in Figure 4.6. The main content area has the title 'About Us' and the logo 'LIEGE université Gembloux Agro-Bio Tech'. Below the title, there's a paragraph of text about the platform's origin. At the bottom, there's a section titled 'Project Managers' with a list of names and a section titled 'The rest of the Team' with a list of names.

Figure 4.7: About Us Page (C.7)

The screenshot shows the 'Login' page. The top navigation bar is the same as in Figure 4.6. The main content area has the title 'Login' and a login form with fields for 'Username' and 'Password', and a 'Log In' button.

Figure 4.8: Login Page (C.5)

The screenshot shows the 'Admin' page. The top navigation bar is the same as in Figure 4.6. The main content area has several panels for managing the database:

- Add classifications:** A panel with dropdowns for Order, SubOrder, Family, SubFamily, Tribe, Genus, and SubGenus, and an 'Add' button.
- Delete classification:** A panel with dropdowns for Order, SubOrder, Family, SubFamily, Tribe, Genus, and SubGenus, and a 'Delete' button.
- Modify Boxes data - Overwrite mode:** A panel with a 'Choose an item' dropdown and a 'Modify' button.
- Modify individuals data - Overwrite mode:** A panel with a 'Choose an item' dropdown and a 'Modify' button.
- Add user:** A panel with fields for Username, Password, Confirm password, and a 'Add' button.
- Modify user:** A panel with fields for Username, New password, Confirm password, and a 'Modify' button.
- Add collection:** A panel with a 'New Collection' dropdown and an 'Add' button.
- Modify collection:** A panel with a 'Collection to modify' dropdown and a 'Modify' button.
- Add borrower:** A panel with fields for New borrower's name, New borrower's mail, New borrower's phone, and an 'Add' button.
- Modify borrower:** A panel with a 'Borrower to modify' dropdown and a 'Modify' button.

Figure 4.9: Admin Page (C.6)

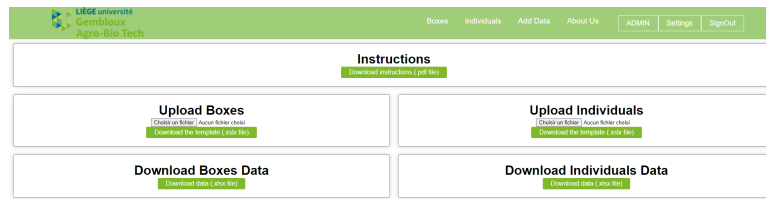


Figure 4.10: Add Data (Scripts) Page (C.7)

### 4.2.3 Back-end

As previously explained, our back-end is divided into microservices, putting aside the front-end and the API gateway, these are four in number: Login, DBOps (Operation on entomological database), File Downloader and PictureNFS (Picture retrieval from a NAS).

Here is a reminder of their functions:

- **Login:**  
This service is used for the connection, the registration, the recovery of the user data but also the authentication by token. It has its own database (db-login).
- **DBOps (DB Operations):**  
This service is responsible for operations requiring queries to the entomological database. These operations being numerous it would be possible to split them into different categories and thus have more microservices. For the moment this single microservice seems to be sufficient for this type of operations. It has its own database (db-entomo).
- **FileDownloader:**  
Responsible for the transmission of files to the user. These files are established in advance, are stored in the microservice and are not sensitive to changes in the various databases.
- **PicturesNFS:**  
Retrieve pictures thanks to NFS (Network File System) protocol. The NAS is in the local network of the web server.

Each of the folders of these microservices has, in addition to a Dockerfile allowing to create its image, several files that are copied in this image:

- A `gulpfile.js` file, which is a configuration file for gulp, which is a build tool for JavaScript projects. Here, it is used to launch a daemon, which is a program

that runs in the background. This file is identical in each microservice (even the gateway) because it is used for the same thing.

- A `daemon.js` which is a file containing the code that the daemon runs (for reminder: the program that runs in the background). In this case, the daemon is an Express server that listens to HTTP requests while redirecting them to `app.js` which will manage the requests. This file is identical in each microservice (even the gateway) because it is used for the same thing.
- a `package.json` file which is a file commonly used in Node.js projects to describe the project's dependencies and their versions. It contains a list of all npm packages used in the project, along with their exact versions or a range of allowed versions. This file is very different from one microservice to another, simply because each microservice has different module requirements.
- an `app.js` file which contains the code managing the incoming requests and the answers to be sent back.
- an `utils` file where are implemented the functions used to operate on a database (`db_ops.js` for DBOps and `login_ops.js` for Login). The functions of these files are called in `app.js` when receiving some requests to write, read or even sometimes both in the database. In addition to the pg client which is used to operate on the database [26], some spawn functions also have child processes in order to launch scripts thanks to the `child_process` module or the `fs` module which enables interacting with the file system.

As mentioned above, child processes are used. It is interesting to explain why and how:

When a python script run is needed (when converting the database to an `xlsx` file, or adding data from an `xlsx` file) a `childprocess` is created:

---

```
1  /*
2  Launch script sql->csv (scripts/SQLToCsvBox.py or
   scripts/SQLToCsvIndividu.py)
3  input: -type: Box/Individual
4  output: csv file corresponding to sql database
5  */
6  function SqlToCsv(type){
7  return new Promise(function(resolve, reject) {
8      var scriptfilename = ''
9      if (type === 'Box') {
10         scriptfilename = 'SQLToCsvBox.py'
11     }
12     else if (type === 'Individual') {
```

```

13     scriptfilename = 'SQLToCsvIndividu.py'
14 }
15 else {
16     return reject("Wrong script's type")
17 }
18 console.log('Subprocess ${scriptfilename} spawning')
19 const script = spawn('python3', [scriptfilename]);
20 console.log("Subprocess spawned")

```

The work is then done on the child process side (python scripts). The only work left to do on the parent process side is to retrieve the result (or the logs in case of an error).

This is done with the commands:

- ```

script.stdout.on('data', (data) => {
    (...)
});

```

---

where data is a readable stream that represents the stdout of the child process.

- ```

script.stderr.on('err', (err) => {
    (...)
});

```

---

where err is a readable stream that represents the stdout of the child process.

- ```

script.on('close', (code) => {
    (...)
});

```

---

launched when the child process run is finished.

#### 4.2.4 API Gateway - Forwarding requests from Front-end to Back-end

Our API Gateway receives requests from the client and then dispatches them to the different microservices depending on which message has been received.

To put it simply, it is based on conditions checking whether the path of the received request belongs to such or such category.

We separate the queries into 10 categories in this project, each client request is treated differently depending on the category to which it belongs :

- **getDBOPS**: HTTP GET request that will be processed by the **DBOps** microservice. The response will not be in the form of a stream.
- **getLOGIN**: HTTP GET request that will be processed by the **Login** microservice. The response will not be in the form of a stream.
- **getDBOPSwStreamReponse**: HTTP GET request that will be processed by the **DBOps** microservice. Response type is a stream because response contain a file.
- **getDLDERwStreamResponse**: HTTP GET request that will be processed by the **File Downloader** microservice. Reponse type is a stream because response contain a file. In fact, every answers returned by **File Downloader** microservice are of type stream.
- **getPICTUwStreamResponse**: HTTP GET request that will be processed by the **PictureNFS** microservice. Reponse type is a stream because response contain a file. In fact, every answers returned by **PictureNFS** microservice are of type stream.
- **postLOGIN**: HTTP POST request that will be processed by the **Login** microservice.
- **postDBOPSwAdminVerif**: HTTP POST request that will be processed by the **DBOps** microservice. These requests also require a call to the login microservice to verify the administrator token.
- **postDBOPSwUserVerif**: HTTP POST request that will be processed by the **DBOps** microservice. These requests also require a call to the login microservice to verify the user token.
- **DBOPSupload**: HTTP PUT request that will be processed by the **DBOps** microservice. Here the request contains a file.
- **DBOPSuploadAdmin**: HTTP PUT request that will be processed by the **DBOps** microservice. Here the request contains a file. These requests also require a call to the login microservice to verify the user token.

A category simply takes the form of a list as shown here:

---

```
const postLOGIN = ['/login', '/signup', '/adminright',
  '/validate-token', '/modifypw', '/modifyright']

const getLOGIN = ['/get-users']
```

```
const postDBOPSwAdminVerif = ['/add-attribute/:name',  
  '/delete-attribute/:name', '/addcollection', '/modifycollection',  
  '/addborrower', '/modifyborrower']
```

---

And are then recognized and treated appropriately through our conditions -Here match is a function implemented in path-to-regexp npm package [27] and app is the express router [28]- :

---

```
app.use((req, res) => {  
  if (getDBOPS.some(route => match(route)(req.path)) && req.method ===  
    'GET') {  
    axios.get('http://${IP_DBOPS}${req.url}')  
      .then((response) => {  
        res.status(200).json(response.data);  
      })  
      .catch((err) => {  
        errorHandler(err, res)  
      });  
  }  
  else if (getDBOPSwStreamReponse.some(route =>  
    match(route)(req.path)) && req.method === 'GET') {  
    axios.get('http://${IP_DBOPS}${req.url}', { responseType:  
      'stream' })  
      .then((response) => {  
        response.data.pipe(res)  
      })  
      .catch((err) => {  
        errorHandler(err, res)  
      });  
  }  
  (...)  
  else if (req.method === 'OPTIONS' && AllRequests.some(route =>  
    match(route)(req.path))) {  
    res.set('Access-Control-Allow-Origin', '*');  
    res.set('Access-Control-Allow-Methods', 'GET, POST, PUT');  
    res.set('Access-Control-Allow-Headers', 'Content-Type');  
    res.set('Access-Control-Max-Age', '3600');  
    res.sendStatus(200);  
  }  
  else {  
    res.status(404);  
  }  
}
```

---



U can see that the OPTIONS requests are also managed in order to let the client know whether or not his request is managed by the server and thus the requests that are not supposed to be sent to our back-end return a 404 error.

# Chapter 5

## Testing and Evaluation

### 5.1 Test Plan

Three types of tests will be performed on our application:

- One to test the correct functioning of the SQL queries used during the different operations on the entomological database. This information being very important, we must ensure that they are stored without any risk for their integrity. In order to test these queries, the use of pgTAP is interesting.
- A second one to test the security of the different fields of the application where the user can enter data, and therefore the different data that can be entered as parameters of HTTP requests to our application. For this we will test that no request is sensitive to SQL injections thanks to SQLMap.
- A last one to know the user satisfaction thanks to pre-selected test users. This user satisfaction will give a good idea of the completeness of the application, of its proper functioning, as well as the quality of the interface in the eyes of the users. We will use a well-known and documented method for years, the user testing.

Each step of these tests as well as the changes that have been made to the application as a result of them will be described in this section.

These changes will have been explained in the previous chapters as it is the final product that is documented in this document, although it will be indicated that these elements are due to the changes found necessary to be made as a result of the different tests.

## 5.2 Test Results and Analysis

### 5.2.1 Testing Queries using pgTAP

#### pgTAP

pgTAP is a unit testing framework for a PostgreSQL database management[29]. It provides several functions to either test the design of the database but also but also to check the correct encoding of the data. Testing a database is really important, because when creating the database or running certain scripts, data can be incorrectly encoded. These are often very hard to notice because it is impossible to look at all the data one by one. pgTAP is preferred to PGUnit as it is not been updated since Postgres 8.3+ (2008) [30]

When creating tables, many elements are taken into account such as primary keys, foreign keys or the form of the attributes, whether they can be empty or not. If these elements are badly encoded, this can create problems in the future and are often complicated to correct. It can also cause problems when inserting data if you try to assign values to a primary key when it is already present. This is why it is also useful to test insertion queries.

pgTAP must be installed on the same host as your PostgreSQL database, if will not worked remotely. The installation is quite easy and they have a good tutorial on their website( [link](#)). After the downloading of the repository, you must launch the command `make`, `make install` and `make installcheck`. These commands will install the framework, however there may be errors during the installation and again the lines to be modified accordingly are written on their website. Once it is installed, you can connect your database by creating an extension to it `CREATE EXTENSION pgtap;`. Finally, you can test your database [30].

#### Tests and Results

The first interesting test to do is to check if your database contains all the expected tables. To do this, there is a function called `tables_are` which will take as argument a list of all expected tables. This is preceded by the function `plan()` which indicates the number of tests expected in this file (here: 1). At the end it is important to indicate that the tests are finished using the query `SELECT * FROM finish();` To execute the tests, you must run the `pg_prove` command of the desired test on your database.

```

BEGIN;
CREATE EXTENSION pgtap;
SELECT plan(1);
SELECT tables_are('public', ARRAY['Order', 'subOrder', 'Family',
    'subFamily', 'Tribu', 'Scientific', 'Genus', 'subGenus', 'Species',
    'subSpecies', 'Population', 'Collection', 'Box', 'PopuBox',
    'Individu', 'Borrower', 'borrowerBox', 'borrowerIndividu']);
SELECT * FROM finish();
ROLLBACK;

```

---

As mentioned above, pgTAP has many features that allow you to test the shape of a table. In the example below, we can see that the different queries allow to test the shape of the order table. This test verifies that all columns have been added by indicating which ones are primary keys and if they can have a null value. In addition, it also checks whether the elements of a column must be unique or not.

---

```

BEGIN;
CREATE EXTENSION pgtap;
SELECT plan(9);
SELECT columns_are('Order', ARRAY['id_order', 'name']);
SELECT has_column('Order', 'id_order');
SELECT col_is_pk('Order', 'id_order');
SELECT col_not_null('Order', 'id_order');
SELECT col_type_is('Order', 'id_order', 'integer');
SELECT has_column('Order', 'name');
SELECT col_type_is('Order', 'name', 'character varying(100)');
SELECT col_not_null('Order', 'name');
SELECT col_is_unique('Order', 'name');
SELECT * FROM finish();
ROLLBACK;

```

---

```

1..9
ok 1 - Table "Order" should have the correct columns
ok 2 - Column "Order".id_order should exist
ok 3 - Column "Order"(id_order) should be a primary key
ok 4 - Column "Order".id_order should be NOT NULL
ok 5 - Column "Order".id_order should be type integer
ok 6 - Column "Order".name should exist
ok 7 - Column "Order".name should be type character varying(100)
ok 8 - Column "Order".name should be NOT NULL
ok 9 - Column "Order"(name) should have a unique constraint
ok
All tests successful.
Files=1, Tests=9, 2 wallclock secs ( 0.02 usr  0.00 sys +  0.04 cusr  0.01 csy
s =  0.07 CPU)
Result: PASS

```

Figure 5.1: Execution of a successful pgTAP command

Finally, we can check that the inserts of our different scripts do not contain errors by using two functions:

- `lives_ok( :sql )` that take in argument a query and ensure that it does not return an error.
- `throws_ok( :sql, :errmsg )` that take in argument a query and the error message it should return.

---

```

SELECT lives_ok(
  'INSERT INTO
    "Order" ("id_order", "name")
  VALUES
    (5000, $$OrderPGTAP$$);'
);
SELECT throws_ok(
  'INSERT INTO
    "Order" ("id_order", "name")
  VALUES
    (5000, $$OrderPGTAP2$$);',
  'duplicate key value violates unique constraint "Order_pkey"'
);

```

---

## 5.2.2 Security tests using SQLmap

### SQLmap

SQLmap is an open-source tool used to detect and exploit SQL injection vulnerabilities in web applications. This technology is used to test the security of SQL

databases used by web applications. SQLmap can be used for MySQL, Oracle, PostgreSQL, Microsoft SQL Server and other databases [31].

SQL injection is a technique for exploiting security holes in applications that use SQL databases. It consists of inserting malicious code into the SQL queries sent to the database. The goal is to corrupt the database and obtain confidential information stored in it.

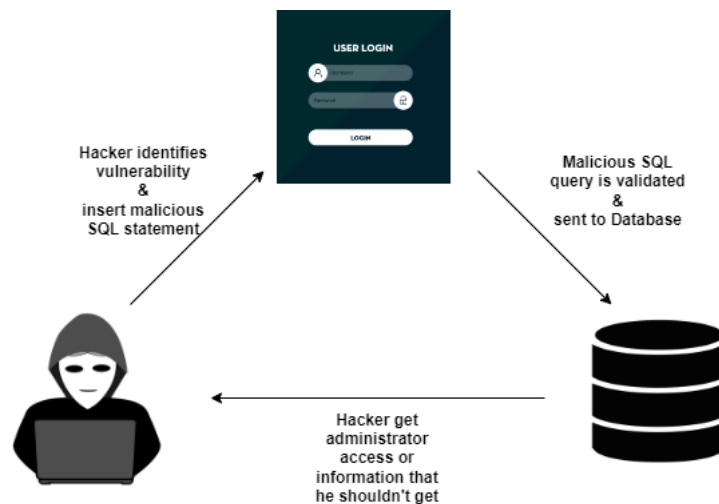


Figure 5.2: SQL injection illustrated

SQLmap can detect security holes in web applications by sending modified SQL queries and examining the received responses. It can also exploit these vulnerabilities by modifying SQL queries sent to the database and executing malicious commands. [31].

Academic research has been conducted to evaluate the effectiveness of SQLmap in detecting and exploiting SQL injection vulnerabilities. In *SQL Injection Attack: Detection and Prevention* by Swayam Charania and Vidhi Vyas, the authors used several SQL injection detection tools to test the security of various web applications and showed that the SQLmap was effective in detecting SQL injection vulnerabilities in web applications [32].

It is worth noting that SQLmap should only be used on web applications for which you have permission to test security. Using SQLmap on applications without authorization may be a violation of the law and may result in legal action. In addition, SQLmap can potentially cause database damage if used incorrectly [31].

## Tests and Results

As mentioned before, to know whether or not a query is sensitive to SQL injection we use SQLmap. To be more precise, these are the commands that are used depending on the type of query to test, here api gateway ip is localhost:4000

Here's an example for GET request (This request retrieves the information of a box from its id) :

---

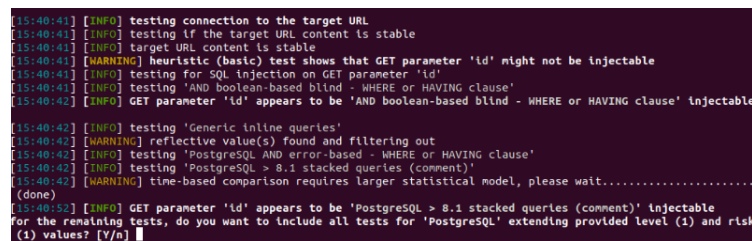
```
python3 sqlmap.py -u "http://localhost:4000/get_boxdetails?id=1" -p
--method get --dbms postgres --ignore-code 404
```

---

Four options are used in this command:

- The "-u" option specifies the URL of the web page to test.
- The "-p" option is used to tell SQLMap to test all parameters of the specified URL.
- The "--method" option is used to specify the HTTP method to use for the attack (in this case, "get").
- The "--dbms" option is used to specify the database management system (DBMS) to target (in this case, "postgres").
- The "--ignore-code" option is used to specify the HTTP response codes to ignore during the attack (in this case, "404").

During the test of injection operated thanks to the command given just above, the id parameter was found to be injectable, and this as soon as the first injection types were tested.



```
[15:40:41] [INFO] testing connection to the target URL
[15:40:41] [INFO] testing if the target URL content is stable
[15:40:41] [INFO] target URL content is stable
[15:40:41] [WARNING] heuristic (basic) test shows that GET parameter 'id' might not be injectable
[15:40:41] [INFO] testing for SQL injection on GET parameter 'id'
[15:40:41] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[15:40:42] [INFO] GET parameter 'id' appears to be 'AND boolean-based blind - WHERE or HAVING clause' injectable
[15:40:42] [INFO] testing 'Generic inline queries'
[15:40:42] [WARNING] reflective value(s) found and filtering out
[15:40:42] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[15:40:42] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[15:40:42] [WARNING] time-based comparison requires larger statistical model, please wait.....
[done]
[15:40:52] [INFO] GET parameter 'id' appears to be 'PostgreSQL > 8.1 stacked queries (comment)' injectable
for the remaining tests, do you want to include all tests for 'PostgreSQL' extending provided level (1) and risk
(1) values? [Y/n]
```

Figure 5.3: Injectable parameter detected by SQLMap

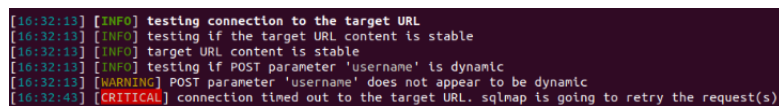
Let's discuss the solution (the fix) after talking about a second type of problem that can be detected indirectly by SQLMap: unanswered request because of unhandled error and thus also indirectly a crash of the back-end.

To do this, let's introduce the command used to test the parameters of a POST request (This one verifies login information):

```
1 python3 sqlmap.py -u "http://localhost:4000/login" --data
  "username=1,password=2" --method post --dbms postgres
  --ignore-code 401
```

Notice that we use here the "-data" option and not the "-p" option because we are dealing with body parameters and not query parameters.

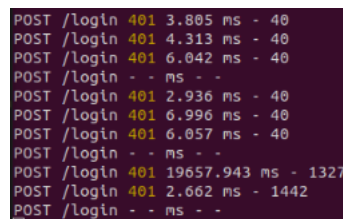
This kind of problem can be detected thanks to the critical errors of SQLMap, raised here when the target url time out.



```
[16:32:13] [INFO] testing connection to the target URL
[16:32:13] [INFO] testing if the target URL content is stable
[16:32:13] [INFO] target URL content is stable
[16:32:13] [INFO] testing if POST parameter 'username' is dynamic
[16:32:13] [WARNING] POST parameter 'username' does not appear to be dynamic
[16:32:43] [CRITICAL] connection timed out to the target URL. sqlmap is going to retry the request(s)
```

Figure 5.4: SQLMap critical error (target URL timed out)

After observing this kind of error, you just have to check the logs of the microservice in question to know what is wrong.



```
POST /login 401 3.805 ms - 40
POST /login 401 4.313 ms - 40
POST /login 401 6.042 ms - 40
POST /login - - ms - -
POST /login 401 2.936 ms - 40
POST /login 401 6.996 ms - 40
POST /login 401 6.057 ms - 40
POST /login - - ms - -
POST /login 401 19657.943 ms - 1327
POST /login 401 2.662 ms - 1442
POST /login - - ms - -
```

Figure 5.5: Login microservice not responding to certain requests

To deal with unhandled errors, a custom try/catch system is sufficient, but how to deal with the first problem raised, SQL injection.

Solution is pretty simple, in place of using formatted string queries you can use parameterized query feature provided by pg client in JavaScript [26] (but also by psycopg3 [33] in python scripts of this project).

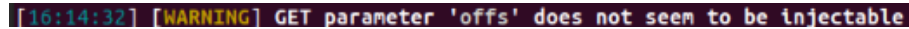
The problem is solved this way because the client uses the query preparation functionality of PostgreSQL.

This separates the SQL query from its input data. The SQL query is sent to the database server as a prepared statement, with placeholders for the input data. The input data is sent separately from the query, securely and serialized, which prevents attackers from injecting malicious code into the SQL query.



This prevents attackers from modifying the SQL query by inserting special characters or malicious code into the input data.

After making this change to each of the queries, each of the parameters of the request operating on a database in this project return the same thing when tested:



```
[16:14:32] [WARNING] GET parameter 'offs' does not seem to be injectable
```

Figure 5.6: Uninjectable parameter tested with SQLMap

### 5.2.3 User testing

#### Approach

To evaluate our application, we also decided to call on users to test it. They are the ones who will use the application, so they are totally legitimate to give their opinion on it and to know if it is complete according to them.

As described in *Universal principles of design*, testing user interface and identify usability issues is important in order to have a product that is pleasant to use and complete for the user [15]. So, in a nutshell, user testing is an essential step in the testing process, as it allows you to understand how users interact with your product and to identify potential problems or improvements.

According to the book *Software for use: a practical guide to the models and methods of usage-centered design*, user testing is much more effective with a specific objective in mind [34]. Our objective is to evaluate the usability of our application but also its completeness for the end users.

At the level of usability, five characteristics of applications seem to be important: learnability, rememberability, efficiency in use, reliability in use, user satisfaction [34].

From the point of view of completeness, and according to Nielsen, it is necessary to take into account some of the above-mentioned elements but also the way in which user errors are managed [14].

Taking all this information into account, we can therefore draw up a user questionnaire which would consist of these questions:

1. How intuitive do you find the site to be to use? Were there times when you had trouble finding what you were looking for on the app?
2. Is the interface pleasing to the eye?

3. Did you encounter any bugs?
4. If you had to add features, what would they be?
5. When you encounter an error, is the message describing it clear?

Now that the questions are written, it remains to be seen how many testers will be needed. Nielsen recommends having between 5 and 8 testers, because after the fifth user, you're wasting your time watching the same results over and over again without learning much new [35] (What proved to be true during the user tests of this project).

The tests were done with the help of 8 people: 4 entomologists, 2 people at ease with computers and 2 people a little less used to the internet.

## Results

1. How intuitive do you find the site to be to use? Were there times when you had trouble finding what you were looking for on the app?
  - 100% found the application intuitive to use
  - "I find it quite intuitive and easy to use", "I find it quite easy to understand where we need to go."
2. Is the interface pleasing to the eye?
  - There is room for improvement
  - "In terms of design, it is still rough but it can be improved easily I think", "The app design is a bit cold", "Some pages are too busy and there is too much information especially in user/admin mode"
3. Did you encounter any bugs?
  - Thanks to this question we were able to detect translation errors, typos, even if they are not bugs.
4. If you had to add features, what would they be?
  - Non-entomologists' testers and half of entomologists testers did not find any features to add.
  - But these features were mentioned by two testers:
    - A map for each box/individual indicating its geographical origin: unfortunately, we don't have enough data for this yet, but this information can be retrieved from the database, it just needs to be added

- A search field for collections, orders, boxes, individuals: we have chosen to implement drop-down menus, these drop-down menus can display a value included in the choices when it is written by the user. Unfortunately, this was not noticed at first sight by some testers.
- As the author of the application, we also have ideas for features to add. These will be discussed in the conclusion.

5. When you encounter an error, is the message describing it clear?

- 100% found the error messages easily understandable and confirm that these error messages were only displayed when a server or user error occurred.

It was therefore necessary to change the graphic style of the application, besides the need to unload it, it was also good to make it less dull and more visually pleasant. No real bug having been detected and the features already implemented being already sufficient for the current state of the database, there was only that to improve by following the comments of the testers.

## Changes made

In addition to a change of color so that the logo is fully visible, the interface has been made a little more colorful and the amount of information displayed has been made customizable both on the search page (through the selection of attributes to be displayed) and on the details page (where the containers to change the information of a page or an individual are displayed only if the user activates the modification mode).

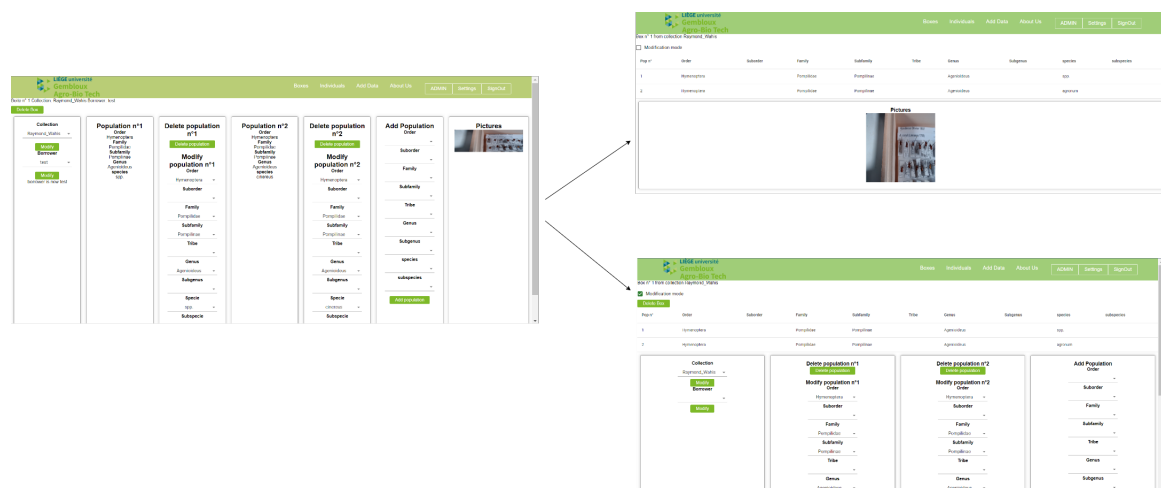


Figure 5.7: Changes made to Details page (From C.8 to C.2 and C.3)



# Chapter 6

## Conclusion

### 6.1 Summary of Accomplishments

During the course of this work, the first step was to develop the design of the database and the application.

For the design of the database, ER diagrams with crow's foot notation were used, passing through the conceptual, logical and physical stages.

For the design of the application, we went through a step of taking information about the needs of the users and then moving on to the step of obtaining a general rendering of the application. We also had to design the architecture of the application. Using microservices, it was necessary to analyze which were the various containers necessary to obtain a good division of these in order to have the best possible portability and isolation as well as a good flexibility.

At the implementation level, we had to create the SQL tables as well as the python scripts allowing to fill them on the basis of excel files. It was also necessary, as previously mentioned, to create a user interface, for which the React framework was used for the front-end and JavaScript for the back-end. The operational development was managed under Docker.

Several different tests were carried out to ensure the completeness and efficiency of the solution provided. Firstly, the pgTAP unit framework was used to check the relevance of the data before and after additions, deletions and data updates.

The security was tested at the level of SQL injections thanks to SQLmap, an automated tool allowing to test the parameters of http requests.

Finally, a user test was set up to ensure that the needs of entomologists were managed by the application.

## 6.2 Limitations and Future Work

This first version of the application, having been designed to meet the needs currently achievable with the data already in the hands of entomologists, is improvable following future needs. Throughout the work, it has been a point of honor to make the code as maintainable and improvable as possible. In this way, it would be interesting in the future to add new attributes according to the needs of the users (the collection of an individual, population dictionary, etc.) or even to show new information for which a space has already been dedicated in the database but which is not yet available (ecozone, geographical coordinates, etc.).

For example, in the long term, once more data has been collected, it would be necessary to remove the direct link between a population and a box and keep only the link between an individual and a population, so that the different populations in a box are just the populations of the insects in that box.

It would also be interesting to improve the logs system so that they are stored in a NoSQL database, which would allow, in addition to a better retrieval of the logs (already possible in part with the `docker logs` command), a verification of the users' actions if we link the logs to their actions.

If an ambition to have a mobile interface for the application emerges, it would be interesting to add a microservice for the mobile front-end which could be coded in Flutter and put online in the same way as the current front-end.

# Bibliography

- [1] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, “Agile software development methods: Review and analysis,” *arXiv preprint arXiv:1709.08439*, 2017.
- [2] C. Mann and F. Maurer, “A case study on the impact of scrum on overtime and customer satisfaction,” in *Agile Development Conference (ADC’05)*, IEEE, 2005.
- [3] “Darwin - data research warehouse information network.”
- [4] “Natural history museum - data portal.”
- [5] “Gbif | global biodiversity information facility.”
- [6] M. Adam, F. Theeten, J.-M. Herpers, T. Vandenberghe, P. Semal, D. Van den Spiegel, and P.-A. Duchesne, “Darwin: An open source natural history collections data management system,” *Biodiversity Information Science and Standards*, 2019.
- [7] B. Scott, E. Baker, M. Woodburn, S. Vincent, H. Hardy, and V. S. Smith, “The natural history museum data portal,” *Database*, vol. 2019, 2019.
- [8] A.-S. Archambeau, E. Chenin, M.-E. Lecoq, P. R. Ristol, and R. Vignes Lebbe, “Le gbif: ouvrir l’accès aux données primaires sur la biodiversité,” *Netcom. Réseaux, communication et territoires*, no. 27-1/2, pp. 154–164, 2013.
- [9] G. C. Simsion and G. C. Witt, *Data Modeling Essential*. Morgan Kaufmann Publisher, 3 ed., 2005.
- [10] M. G. Morales, B. D. Denno, D. R. Miller, G. L. Miller, Y. Ben-Dov, and N. B. Hardy, “Scalenet: a literature-based model of scale insect biology and systematics,” *Database J. Biol. Databases Curation*, vol. 2016, 2016.

- [11] S. Hitchman, "The details of conceptual modelling notations are important - A comparison of relationship normative language," *Commun. Assoc. Inf. Syst.*, vol. 9, p. 10, 2002.
- [12] N. Elmasri, *Fundamentals of Database System*. Pearson, 7 ed., 2016.
- [13] J. Nielsen, "Usability inspection methods," in *Conference on Human Factors in Computing Systems, CHI 1994, Boston, Massachusetts, USA, April 24-28, 1994, Conference Companion* (C. Plaisant, ed.), ACM, 1994.
- [14] J. Nielsen, *Usability engineering*. Academic Press, 1993.
- [15] W. Lidwell, K. Holden, and J. Butler, *Universal principles of design*. Rockport Publishers, 2nd ed., 2010.
- [16] I. Miell and A. Sayers, *Docker in Practice*. Manning Publications, 2019.
- [17] J. Turnbull, *The Docker Book: Containerization is the new virtualization*. James Turnbull, 2014.
- [18] P. Saha, A. Beltre, P. Uminski, and M. Govindaraju, "Evaluation of docker containers for scientific workloads in the cloud," *CoRR*, vol. abs/1905.08415, 2019.
- [19] S. Newman, *Building microservices*. " O'Reilly Media, Inc.", 2021.
- [20] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: yesterday, today, and tomorrow," *Present and ulterior software engineering*, 2017.
- [21] "Postgresql - documentation."
- [22] "Mysql - documentation."
- [23] Docker Documentation, "Dockerfile reference." Online, Accessed in 2023.
- [24] Docker Documentation, "Docker compose." Online, Accessed in 2023.
- [25] Facebook, "React - a javascript library for building user interfaces." <https://reactjs.org/docs/getting-started.html>, 2021. Accessed: April 24, 2023.
- [26] B. Carlson, "Node-postgres documentation." <https://node-postgres.com/features/queries>. Accessed in 2023.
- [27] P. Lee, "path-to-regex." <https://www.npmjs.com/package/path-to-regex>, 2021. Accessed in 2023.



- [28] T. Holowaychuk and contributors, “Express.” <https://www.npmjs.com/package/express>, 2010–2022. Accessed in 2023.
- [29] “pgtap.”
- [30] S. McClive, “Unit testing postgres with pgtap.”
- [31] sqlmap Documentation, “sqlmap: automatic sql injection and database takeover tool.” Online. Accessed in 2023.
- [32] S. Charania and V. Vyas, “Sql injection attack: Detection and prevention,” *Int. Res. J. Eng. Technol*, pp. 2395–56, 2016.
- [33] P. Team, *Psycopg3: PostgreSQL database adapter for the Python*. Accessed in 2023.
- [34] L. L. Constantine and L. A. Lockwood, *Software for use: a practical guide to the models and methods of usage-centered design*. Pearson Education, 1999.
- [35] J. Nielsen, “Why you only need to test with 5 users.” Nielsen Norman Group, 2000.

# Appendix A

## Creation of the entomological table

---

```
CREATE TABLE "Order" (  
  "id_order" INT NOT NULL,  
  "name" VARCHAR(100) NOT NULL,  
  PRIMARY KEY ("id_order"),  
  UNIQUE ("id_order"),  
  UNIQUE ("name")  
);
```

```
CREATE TABLE "subOrder" (  
  "id_suborder" INT NOT NULL,  
  "name" VARCHAR(100) NOT NULL,  
  PRIMARY KEY ("id_suborder"),  
  UNIQUE ("id_suborder"),  
  UNIQUE ("name")  
);
```

```
CREATE TABLE "Tribu" (  
  "id_tribu" INT NOT NULL,  
  "name" VARCHAR(100) NOT NULL,  
  PRIMARY KEY ("id_tribu"),  
  UNIQUE ("id_tribu"),  
  UNIQUE ("name")  
);
```

```
CREATE TABLE "Family" (  
  "id_family" INT NOT NULL,
```

```

"name" VARCHAR(100) NOT NULL,
PRIMARY KEY ("id_family"),
UNIQUE ("id_family"),
UNIQUE ("name")
);

CREATE TABLE "subFamily" (
"id_subfamily" INT NOT NULL,
"name" VARCHAR(100) NOT NULL,
PRIMARY KEY ("id_subfamily"),
UNIQUE ("id_subfamily"),
UNIQUE ("name")
);

CREATE TABLE "Scientific" (
"id_sc" INT NOT NULL,
"name" VARCHAR(100) NOT NULL,
PRIMARY KEY ("id_sc"),
UNIQUE ("id_sc"),
UNIQUE("name")
);

CREATE TABLE "Genus" (
"id_genus" INT NOT NULL,
"name" VARCHAR(100) NOT NULL,
"id_sc" INT,
"date" VARCHAR(15),
PRIMARY KEY ("id_genus"),
UNIQUE ("id_genus"),
UNIQUE ("name"),
FOREIGN KEY("id_sc") REFERENCES "Scientific"("id_sc")
);

CREATE TABLE "subGenus" (
"id_subgenus" INT NOT NULL,
"name" VARCHAR(100) NOT NULL,
"id_sc" INT,
"date" VARCHAR(15),
PRIMARY KEY ("id_subgenus"),
UNIQUE ("id_subgenus"),
UNIQUE ("name"),
FOREIGN KEY("id_sc") REFERENCES "Scientific"("id_sc")
);

```

```
);
```

```
CREATE TABLE "Species" (  
  "id_species" INT NOT NULL,  
  "name" VARCHAR(100) NOT NULL,  
  "id_sc" INT,  
  "date" VARCHAR(15),  
  PRIMARY KEY ("id_species"),  
  UNIQUE ("id_species"),  
  UNIQUE ("name"),  
  FOREIGN KEY("id_sc") REFERENCES "Scientific"("id_sc")  
);
```

```
CREATE TABLE "subSpecies" (  
  "id_subspecies" INT NOT NULL,  
  "name" VARCHAR(100) NOT NULL,  
  "id_sc" INT,  
  "date" VARCHAR(15),  
  PRIMARY KEY ("id_subspecies"),  
  UNIQUE ("id_subspecies"),  
  UNIQUE ("name"),  
  FOREIGN KEY("id_sc") REFERENCES "Scientific"("id_sc")  
);
```

```
CREATE TABLE "Population" (  
  "id_population" INT NOT NULL,  
  "order_id" INT NOT NULL,  
  "suborder_id" INT,  
  "tribu_id" INT NULL,  
  "family_id" INT,  
  "subFamily_id" INT,  
  "genus_id" INT,  
  "subGenus_id" INT,  
  "species_id" INT,  
  "subSpecies_id" INT,  
  PRIMARY KEY ("id_population"),  
  UNIQUE ("id_population"),  
  FOREIGN KEY("order_id") REFERENCES "Order"("id_order"),  
  FOREIGN KEY("suborder_id") REFERENCES "subOrder"("id_suborder"),  
  FOREIGN KEY("family_id") REFERENCES "Family"("id_family"),
```

```

FOREIGN KEY("subFamily_id") REFERENCES "subFamily"("id_subfamily"),
FOREIGN KEY("genus_id") REFERENCES "Genus"("id_genus"),
FOREIGN KEY("subGenus_id") REFERENCES "subGenus"("id_subgenus"),
FOREIGN KEY("species_id") REFERENCES "Species"("id_species"),
FOREIGN KEY("subSpecies_id") REFERENCES "subSpecies"("id_subspecies"),
FOREIGN KEY("tribu_id") REFERENCES "Tribu"("id_tribu")

```

```

);

```

```

CREATE TABLE "Collection" (
  "id_collection" INT NOT NULL,
  "name" VARCHAR(100) NOT NULL,
  UNIQUE("id_collection"),
  UNIQUE ("name"),
  PRIMARY KEY ("id_collection")
);

```

```

CREATE TABLE "Box" (
  "id_box" INT NOT NULL,
  "collection_id" INT,
  "location" VARCHAR(50),
  "museum" VARCHAR(50),
  "paratypes" INT,
  "types" INT,
  UNIQUE("id_box"),
  PRIMARY KEY ("id_box"),
  FOREIGN KEY ("collection_id") REFERENCES "Collection"("id_collection")
);

```

```

CREATE TABLE "PopuBox" (
  "population_id" INT NOT NULL,
  "box_id" INT NOT NULL,
  PRIMARY KEY("population_id", "box_id"),
  FOREIGN KEY ("population_id") REFERENCES "Population"("id_population"),
  FOREIGN KEY ("box_id") REFERENCES "Box"("id_box") ON DELETE CASCADE,
  CONSTRAINT "UC_PopuBox" UNIQUE ("population_id", "box_id")
);

```

```

CREATE TABLE "Individu" (
  "id_individu" VARCHAR(500) NOT NULL,

```

```

"box_id" INT NOT NULL,
"population_id" INT NOT NULL,
"continent" VARCHAR(50),
"country" VARCHAR(50),
"ecozone" VARCHAR(50),
"latitude" VARCHAR(50),
"longitude" VARCHAR(50),
"locality" VARCHAR(50),
"number" INT,
"collection_date" VARCHAR(50),
"sexe" VARCHAR(50),
UNIQUE ("id_individu"),
PRIMARY KEY ("id_individu"),
FOREIGN KEY ("box_id") REFERENCES "Box"("id_box"),
FOREIGN KEY ("population_id") REFERENCES "Population"("id_population")
);

```

```

CREATE TABLE "Borrower" (
"id_borrower" INT NOT NULL,
"name" VARCHAR(50) NOT NULL,
"mail" VARCHAR(100),
"phone" VARCHAR(25),
UNIQUE ("id_borrower"),
UNIQUE("name"),
UNIQUE ("mail"),
UNIQUE ("phone"),
PRIMARY KEY ("id_borrower")
);

```

```

CREATE TABLE "borrowerBox" (
"borrower_id" INT NOT NULL,
"box_id" INT NOT NULL,
UNIQUE("borrower_id"),
PRIMARY KEY("borrower_id","box_id"),
FOREIGN KEY ("borrower_id") REFERENCES "Borrower"("id_borrower"),
FOREIGN KEY ("box_id") REFERENCES "Box"("id_box") ON DELETE CASCADE,
CONSTRAINT "UC_borrowerBox" UNIQUE ("borrower_id","box_id")
);

```

```

CREATE TABLE "borrowerIndividu" (
"borrower_id" INT NOT NULL,
"individu_id" VARCHAR(500) NOT NULL,

```

```
UNIQUE("borrower_id"),
PRIMARY KEY("borrower_id","individu_id"),
FOREIGN KEY ("borrower_id") REFERENCES "Borrower"("id_borrower"),
FOREIGN KEY ("individu_id") REFERENCES "Individu"("id_individu") ON
    DELETE CASCADE,
CONSTRAINT "UC_borrowerIndividu" UNIQUE ("borrower_id","individu_id")
);
```

---

## Appendix B

### Rule for filling the different excel



## How to fill the excel for box

| Attributes          | Description                                          | Constraints                                                                                                                                                                                                                                                                                     |
|---------------------|------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Num_ID              | The number of the box                                | <ul style="list-style-type: none"> <li>This field is mandatory.</li> <li>It should have the value 0.</li> </ul>                                                                                                                                                                                 |
| Order               | The name of the order                                | <ul style="list-style-type: none"> <li>This field is mandatory.</li> <li>If there are several orders, they must be separated by “_” (e.g.: Lepidoterres_Hymenoptera). In this case, it cannot have a sub-order, family, sub-family, tribe, genus, sub-genus, species and sub-species</li> </ul> |
| Suborder            | The name of the sub-order                            | <ul style="list-style-type: none"> <li>if there are several sub-orders, they must be separated by “_”. In this case, it cannot have a family, sub-family, tribe, genus, sub-genus, species and sub-species</li> </ul>                                                                           |
| Family              | The name of the family                               | <ul style="list-style-type: none"> <li>if there are several families, they must be separated by “_”. In this case, it cannot have a sub-family, tribe, genus, sub-genus, species and sub-species</li> </ul>                                                                                     |
| Subfamily           | The name of the sub-family                           | <ul style="list-style-type: none"> <li>if there are several sub-families, they must be separated by “_”. In this case, it cannot have a tribe, genus, sub-genus, species and sub-species</li> </ul>                                                                                             |
| Tribu               | The name of the tribe                                | <ul style="list-style-type: none"> <li>if there are several tribe, they must be separated by . In this case, it cannot have a genus, sub-genus, species and sub-species</li> </ul>                                                                                                              |
| Genus               | The name of the genus                                | <ul style="list-style-type: none"> <li>if there are several genera, they must be separated by “_”. In this case, it cannot have a sub-genus, species and sub-species</li> </ul>                                                                                                                 |
| Genus_Descriptor    | The name of the scientist who discover the genus     |                                                                                                                                                                                                                                                                                                 |
| Genus_Date          | The date the scientist discovered the genus          |                                                                                                                                                                                                                                                                                                 |
| Subgenus            | The name of the sub-genus                            | <ul style="list-style-type: none"> <li>if there are several sub-genera, they must be separated by .In this case, it cannot have a species and sub-species</li> </ul>                                                                                                                            |
| Subgenus_Descriptor | The name of the scientist who discover the sub-genus |                                                                                                                                                                                                                                                                                                 |

|                       |                                                        |                                                                                                                                                           |
|-----------------------|--------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| Subgenus_Date         | The date the scientist discovered the sub-genus        |                                                                                                                                                           |
| Species               | The name of the species                                | <ul style="list-style-type: none"> <li>if there are several species, they must be separated by “_”. In this case, it cannot have a sub-species</li> </ul> |
| Species_Descriptor    | The name of the scientist who discover the species     |                                                                                                                                                           |
| Species_Date          | The date the scientist discovered the species          |                                                                                                                                                           |
| Subspecies            | The name of the sub-species                            | <ul style="list-style-type: none"> <li>if there are several sub-species, they must be separated by “_”.</li> </ul>                                        |
| Subspecies_descriptor | The name of the scientist who discover the sub-species |                                                                                                                                                           |
| Subspecies_Date       | The date the scientist discovered the sub-species      |                                                                                                                                                           |
| Types                 | The number of types in the box                         |                                                                                                                                                           |
| Paratypes             | The number of paratypes in the box                     |                                                                                                                                                           |
| Museum                | In which museum the box belong                         |                                                                                                                                                           |
| Box_Localization      | The box localization                                   |                                                                                                                                                           |
| Collection_Name       | The name of the collection                             |                                                                                                                                                           |

## How to fill the excel for individuals

| Attributes            | Description                                          | Constraints                                                                                                                 |
|-----------------------|------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| SpecimenCode          | The code of the specimen                             | <ul style="list-style-type: none"> <li>This field is mandatory.</li> <li>Two specimens cannot have the same code</li> </ul> |
| Order                 | The name of the order                                | <ul style="list-style-type: none"> <li>This field is mandatory.</li> <li>Only one order</li> </ul>                          |
| Suborder              | The name of the sub-order                            | <ul style="list-style-type: none"> <li>Only one sub-order</li> </ul>                                                        |
| Family                | The name of the family                               | <ul style="list-style-type: none"> <li>Only one family</li> </ul>                                                           |
| Subfamily             | The name of the sub-family                           | <ul style="list-style-type: none"> <li>Only one sub-family</li> </ul>                                                       |
| Tribu                 | The name of the tribe                                | <ul style="list-style-type: none"> <li>Only one tribe</li> </ul>                                                            |
| Genus                 | The name of the genus                                | <ul style="list-style-type: none"> <li>Only one genus</li> </ul>                                                            |
| Genus_Descriptor      | The name of the scientist who discover the genus     |                                                                                                                             |
| Genus_Date            | The date the scientist discovered the genus          |                                                                                                                             |
| Subgenus              | The name of the sub-genus                            | <ul style="list-style-type: none"> <li>Only one sub-genus</li> </ul>                                                        |
| Subgenus_Descriptor   | The name of the scientist who discover the sub-genus |                                                                                                                             |
| Subgenus_Date         | The date the scientist discovered the sub-genus      |                                                                                                                             |
| Species               | The name of the species                              | <ul style="list-style-type: none"> <li>Only one species</li> </ul>                                                          |
| Species_Descriptor    | The name of the scientist who discover the species   |                                                                                                                             |
| Species_Date          | The date the scientist discovered the species        |                                                                                                                             |
| Subspecies            | The name of the sub-species                          | <ul style="list-style-type: none"> <li>Only one sub-species</li> </ul>                                                      |
| Subspecies_descriptor | The name of the scientist who                        |                                                                                                                             |

|                 |                                                   |                                                                                                                          |
|-----------------|---------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|
|                 | discover the sub-species                          |                                                                                                                          |
| Subspecies_Date | The date the scientist discovered the sub-species |                                                                                                                          |
| Num_ID          | The individual's box number                       | <ul style="list-style-type: none"> <li>• This field is mandatory.</li> <li>• Should be 0 if not part of a box</li> </ul> |
| Continent       | The continent where the individual comes from     |                                                                                                                          |
| Country         | The country where the individual comes from       |                                                                                                                          |
| Ecozone         | The ecozone where the individual lives            |                                                                                                                          |
| Locality        | The locality where the individual comes from      |                                                                                                                          |
| Number          | Number of individuals                             | <ul style="list-style-type: none"> <li>• Should be equal to at least 1</li> </ul>                                        |
| Collection_Date | Date of when the individual has been discovered   |                                                                                                                          |
| Sexe            | Sexe of the individual                            |                                                                                                                          |

# Appendix C

## WebApp screenshots

| SEARCH BOXES       | <input checked="" type="checkbox"/> Collection | <input checked="" type="checkbox"/> Order | <input checked="" type="checkbox"/> Suborder | <input checked="" type="checkbox"/> Family | <input checked="" type="checkbox"/> Subfamily | <input checked="" type="checkbox"/> Tribe | <input type="checkbox"/> Genus | <input type="checkbox"/> Subgenus | <input type="checkbox"/> species | <input type="checkbox"/> subspecies |
|--------------------|------------------------------------------------|-------------------------------------------|----------------------------------------------|--------------------------------------------|-----------------------------------------------|-------------------------------------------|--------------------------------|-----------------------------------|----------------------------------|-------------------------------------|
| Collection<br>None | Box                                            | Collection                                | Order                                        | Suborder                                   | Family                                        | Subfamily                                 | Tribe                          |                                   |                                  |                                     |
| Order<br>None      | 1                                              | Raymond_Wahis                             | Hymenoptera                                  |                                            | Pompilidae                                    | Pompilinae                                |                                |                                   |                                  | <a href="#">Details</a>             |
| SubOrder<br>None   | 1                                              | Raymond_Wahis                             | Hymenoptera                                  |                                            | Pompilidae                                    | Pompilinae                                |                                |                                   |                                  | <a href="#">Details</a>             |
| Family<br>None     | 3                                              | Raymond_Wahis                             | Hymenoptera                                  |                                            | Pompilidae                                    | Pompilinae                                |                                |                                   |                                  | <a href="#">Details</a>             |
| SubFamily<br>None  | 4                                              | Raymond_Wahis                             | Hymenoptera                                  |                                            | Pompilidae                                    | Pompilinae                                |                                |                                   |                                  | <a href="#">Details</a>             |
| Tribe<br>None      | 5                                              | Raymond_Wahis                             | Hymenoptera                                  |                                            | Pompilidae                                    | Pompilinae                                |                                |                                   |                                  | <a href="#">Details</a>             |
| Genus<br>None      | 6                                              | Raymond_Wahis                             | Hymenoptera                                  |                                            | Pompilidae                                    | Pompilinae                                |                                |                                   |                                  | <a href="#">Details</a>             |
| Subgenus<br>None   | 7                                              | Raymond_Wahis                             | Hymenoptera                                  |                                            | Pompilidae                                    | Pompilinae                                |                                |                                   |                                  | <a href="#">Details</a>             |
| species<br>None    | 8                                              | Raymond_Wahis                             | Hymenoptera                                  |                                            | Pompilidae                                    | Pompilinae                                |                                |                                   |                                  | <a href="#">Details</a>             |
| subspecies<br>None | 9                                              | Raymond_Wahis                             | Hymenoptera                                  |                                            | Pompilidae                                    | Pompilinae                                |                                |                                   |                                  | <a href="#">Details</a>             |
|                    | 10                                             | Raymond_Wahis                             | Hymenoptera                                  |                                            | Pompilidae                                    | Pepsinae                                  |                                |                                   |                                  | <a href="#">Details</a>             |

Rows per page: 10 1 - 10 of 4857

Figure C.1: Search Page

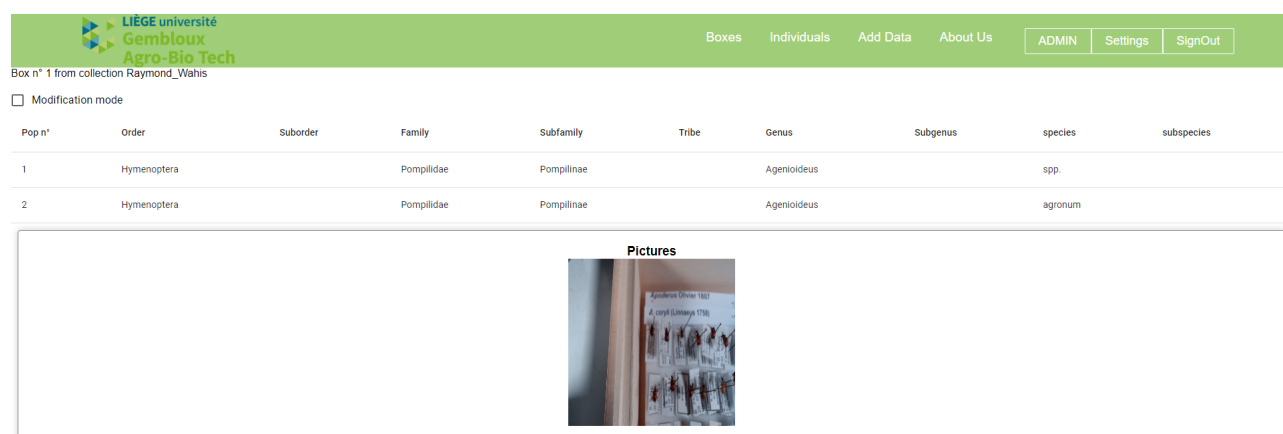


Figure C.2: Details Page

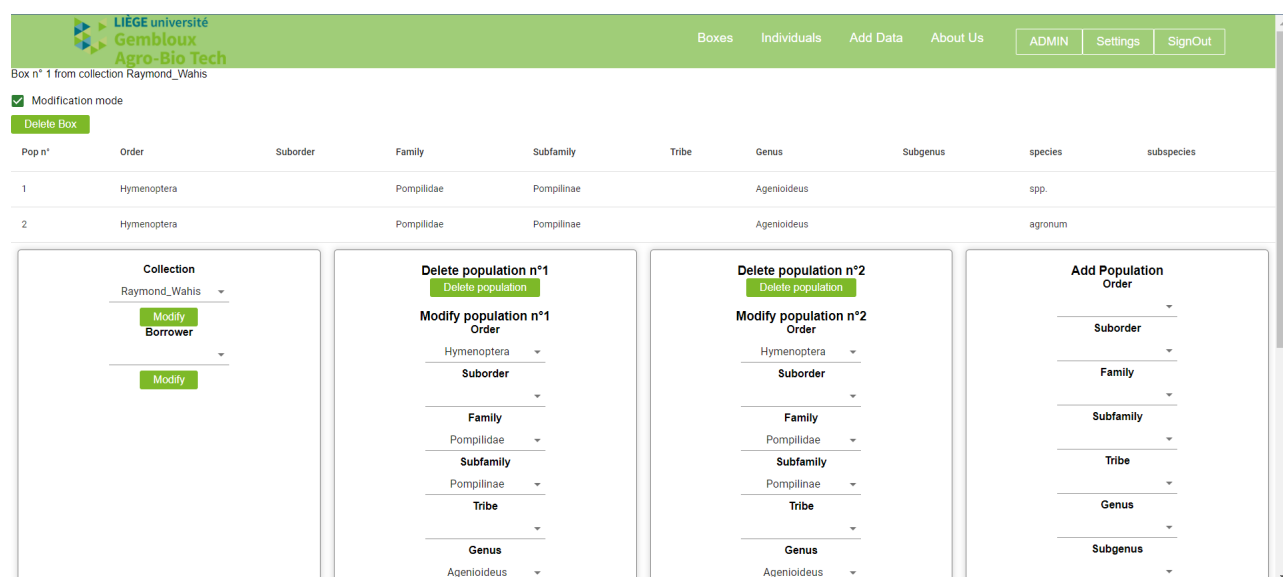


Figure C.3: Details Page (modification mode)

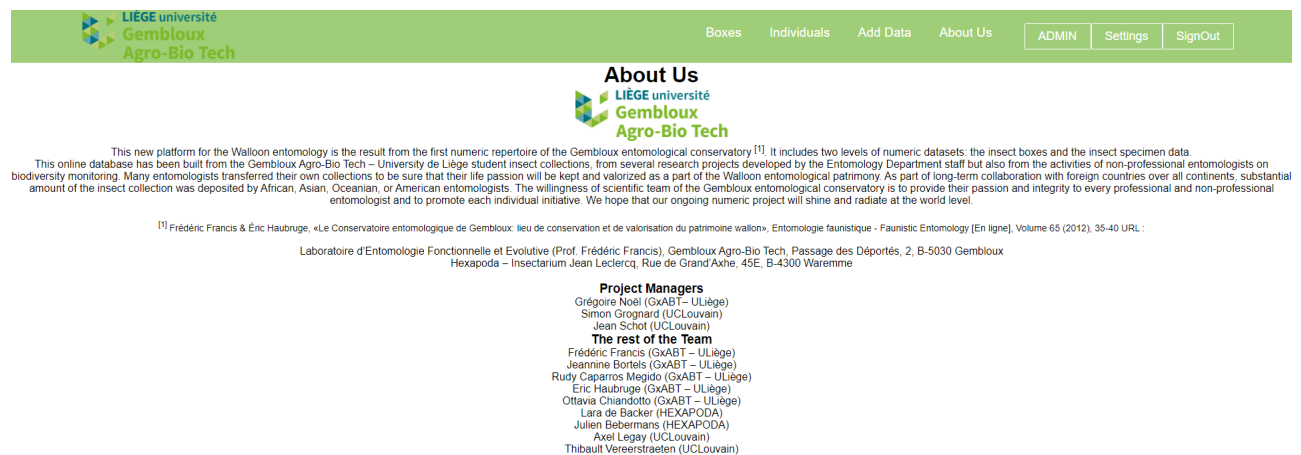


Figure C.4: About Us Page

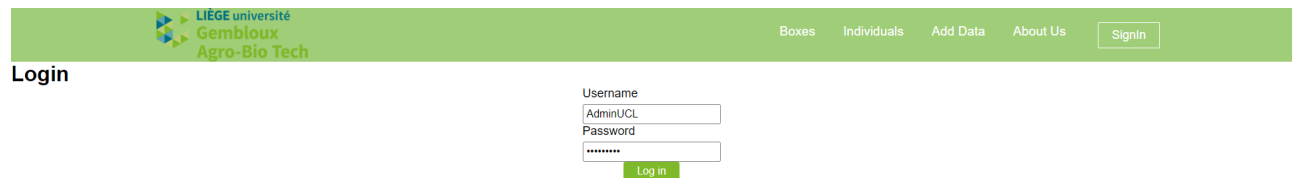


Figure C.5: Login Page

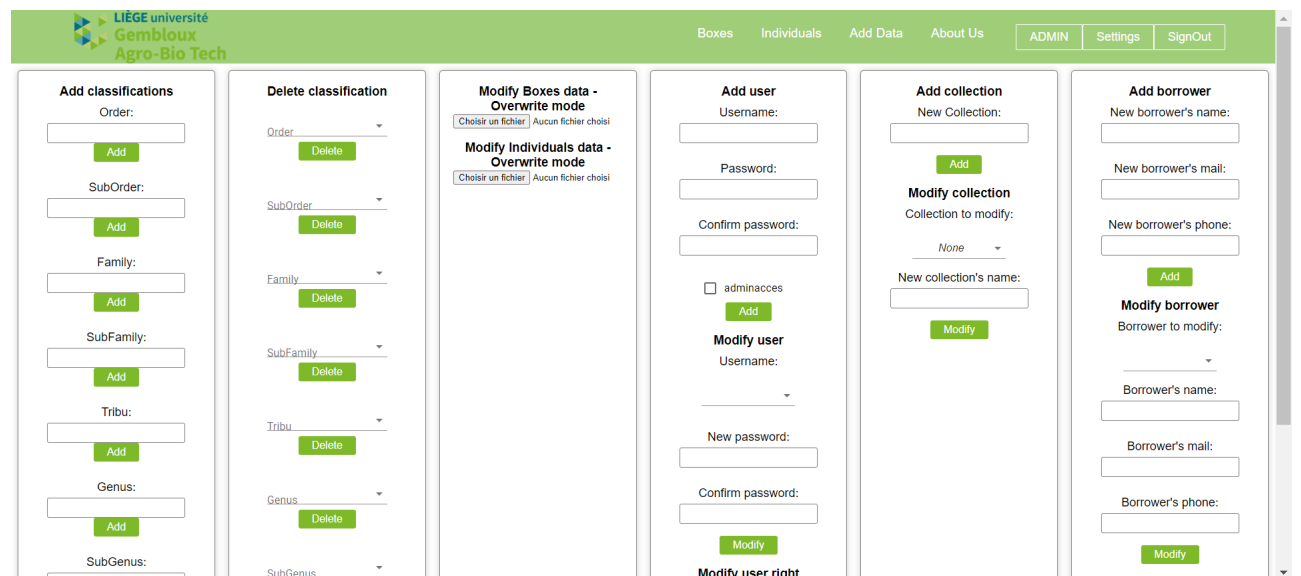


Figure C.6: Admin Page

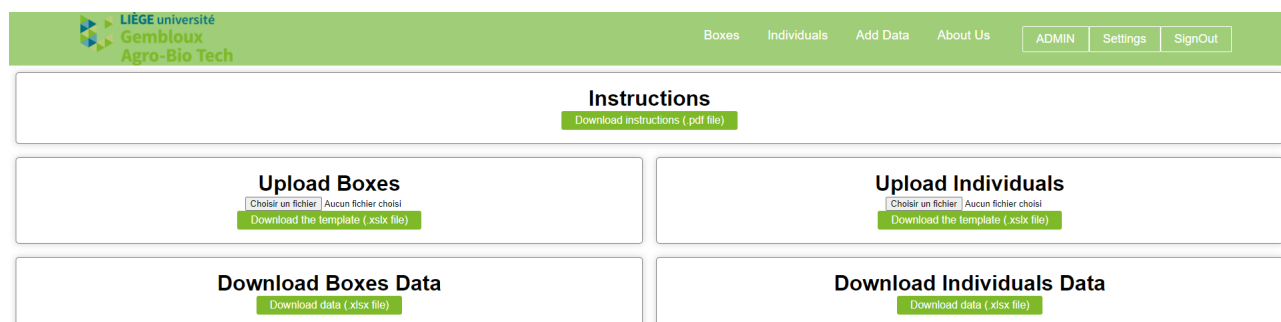


Figure C.7: Add Data (Scripts) Page

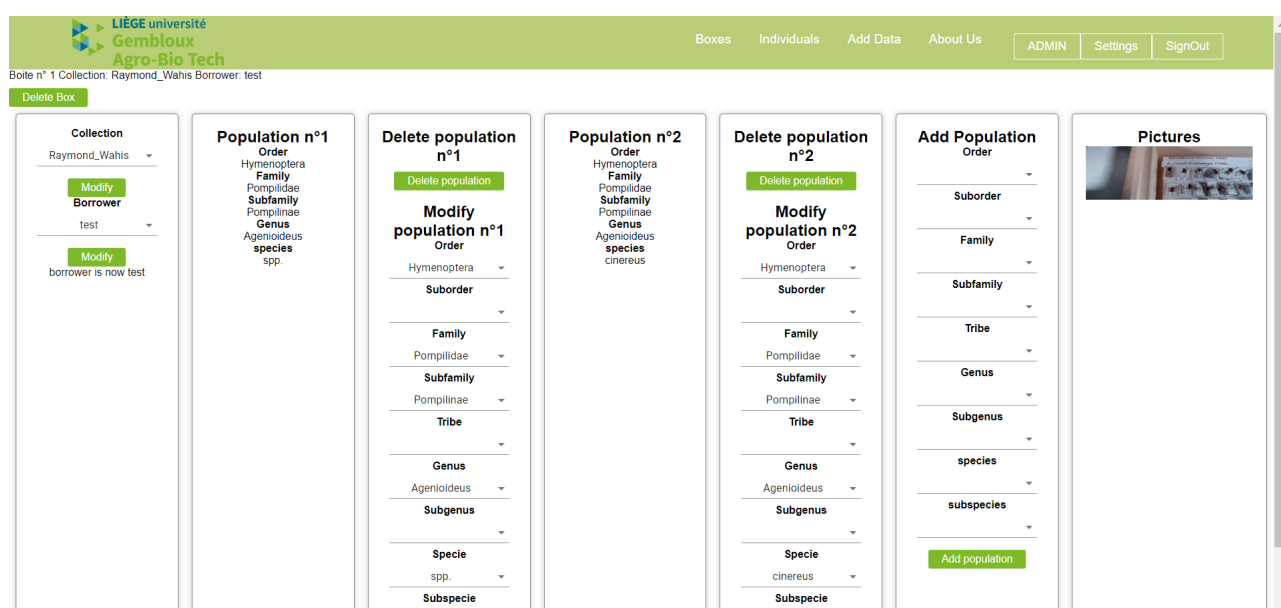



Figure C.8: Details page before changes done after user testing



| <div>  <div>BoxesIndividualsAdd DataAbout Us</div> <div>Signin</div> </div>                                                                                                                               |               |             |          |            |            |       |             |          |          |                         |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|-------------|----------|------------|------------|-------|-------------|----------|----------|-------------------------|
| <div> <div>SEARCH BOXES</div> <div> <div>CollectionNone</div> <div>OrderNone</div> <div>SubOrderNone</div> <div>FamilyNone</div> <div>SubFamilyNone</div> <div>TribeNone</div> <div>GenusNone</div> <div>SubgenusNone</div> <div>speciesNone</div> <div>subspeciesNone</div> </div> </div> |               |             |          |            |            |       |             |          |          |                         |
| Box                                                                                                                                                                                                                                                                                        | Collection    | Order       | Suborder | Family     | Subfamily  | Tribe | Genus       | Subgenus | species  | subspecies              |
| 1                                                                                                                                                                                                                                                                                          | Raymond_Wahis | Hymenoptera |          | Pompilidae | Pompilinae |       | Agenioideus |          | spp.     | <a href="#">Details</a> |
| 1                                                                                                                                                                                                                                                                                          | Raymond_Wahis | Hymenoptera |          | Pompilidae | Pompilinae |       | Agenioideus |          | cinereus | <a href="#">Details</a> |
| 3                                                                                                                                                                                                                                                                                          | Raymond_Wahis | Hymenoptera |          | Pompilidae | Pompilinae |       | Agenioideus |          | spp.     | <a href="#">Details</a> |
| 4                                                                                                                                                                                                                                                                                          | Raymond_Wahis | Hymenoptera |          | Pompilidae | Pompilinae |       | Homonotus   |          |          | <a href="#">Details</a> |
| 4                                                                                                                                                                                                                                                                                          | Raymond_Wahis | Hymenoptera |          | Pompilidae | Pompilinae |       | Priochilus  |          |          | <a href="#">Details</a> |
| 5                                                                                                                                                                                                                                                                                          | Raymond_Wahis | Hymenoptera |          | Pompilidae | Pompilinae |       | Priochilus  |          |          | <a href="#">Details</a> |
| 6                                                                                                                                                                                                                                                                                          | Raymond_Wahis | Hymenoptera |          | Pompilidae | Pompilinae |       | Priochilus  |          |          | <a href="#">Details</a> |
| 7                                                                                                                                                                                                                                                                                          | Raymond_Wahis | Hymenoptera |          | Pompilidae | Pompilinae |       | Priochilus  |          |          | <a href="#">Details</a> |
| 8                                                                                                                                                                                                                                                                                          | Raymond_Wahis | Hymenoptera |          | Pompilidae | Pompilinae |       | Priochilus  |          |          | <a href="#">Details</a> |
| 9                                                                                                                                                                                                                                                                                          | Raymond_Wahis | Hymenoptera |          | Pompilidae | Pompilinae |       | Priochilus  |          |          | <a href="#">Details</a> |

Rows per page: 10
1-10 of 16

Figure C.9: Home page before changes done after user testing

UNIVERSITÉ CATHOLIQUE DE LOUVAIN  
École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | [www.uclouvain.be/epl](http://www.uclouvain.be/epl)