
UNIVERSITÀ
DI TORINO

**University of Turin**
*Computer Science Bachelor's Degree*

# EmergeMobile: Development of a WebApp supported by AI for the service of the biodiversity of French forests

Undergraduate Thesis

**Thesis Supervisor:**
Basile Valerio

**Candidate:**
Venturini Riccardo
Student ID: 947343

Academic Year: 2022/2023

I declare that I am responsible for the content of the paper I am submitting for the purpose of obtaining the degree, that I have not plagiarized in whole or in part the work produced by others, and that I have cited the original sources in accordance with the current regulations on plagiarism and copyright. I am also aware that if my declaration were found to be false, I could incur penalties under the law and my admission to the final exam could be denied.

*I would like to thank Prof. Basile, my thesis advisor, for guiding and supporting me throughout the creation of this paper and for inspiring me in my academic and professional career.*

*I extend my gratitude to Luca Tarricone, a close friend whose companionship I deeply value, for the excellent quality time spent together, encouragement towards significant changes in my life, and for the inspiring conversations we've had.*

*My heartfelt thanks go to Marta Spinosa, a dear friend whose loyalty and wonderful company have been invaluable in many important moments of my life.*

*I am grateful to Alexis Malagnino, a friend whom I am truly thankful to have met, for welcoming me to France during my journey, for the support, and for the fantastic adventures we've shared.*

*I express my gratitude to my mother and sister for their support and the freedoms they've granted me, which have been crucial for achieving my goals.*

*Special thanks to Boris Turcan and Cosmin Lavric, colleagues and, above all, friends, for their invaluable assistance during my academic career and for the engaging discussions we've had on various topics.*

*I would also like to thank Maxent Bernier, a fellow student at the French university, for collaborating with me on the EmergeMobile project.*

# Abstract

This work was carried out on behalf of the Office National des Forêts, a French public organization responsible for forests. Following the positive evaluation of the benefits brought by technology to their work context, the organization decided to purchase dedicated servers for internal data processing and to standardize the applications used through specific conventions.

In the present case, the existing platform dedicated to tree designation activities for felling had malfunctions and design flaws. This paper will analyze in detail the reconstruction of the web part and the database management, respecting the new conventions, and the update of the mobile part for better field application, with the support of the ChatGPT artificial intelligence and technologies such as React, SpringBoot, PostgreSQL, and Kotlin. Furthermore, a general analysis of the involved flora will also be provided, evaluating the impact of this operation on forest ecosystems.

ChatGPT was a constant presence during the project development period, offering support through a web interface to respond to a series of requests and inquiries. Its effectiveness was evident, considering the initial limited knowledge of some technologies by the development team and the significant results achieved at the end of the work. The provided responses helped in understanding the use of the required tools, greatly sped up debugging operations, and, despite some inaccurate responses, provided inspiration for better web searches. *All the information in this paper refers to ChatGPT-3.5 version.*

# Indice

# 1 Introduzione

The ONF (Office National des Forêts) is an organization entrusted by the government to oversee the management and maintenance of forests in French territory. Among the many services offered is that of "martelage," from the French term. This technique involves marking trees to be felled for the benefit of species biodiversity and nearby trees that will gain space to grow. Some of these trees are marked with a triangular sign on the bark to indicate that they cannot be cut until the end of their life. These trees are often designated based on the presence of living animal species inside them or because they have reached the end of their life cycle.

Thanks to this technique, woody material is integrated into the green economy circuit. Indeed, the ONF, being a state organization paid by municipalities for the maintenance of local forests, will subsequently earn a percentage on the sale of wood, while the remainder will go to the landowner. Moreover, the reintegration of wood into society promotes the storage of CO2 in the atmosphere, as 1 kg of dry wood can absorb up to almost 2 tons of CO2.[8] This procedure was called EMERGE, which stands for Élaboration de Modèles pour une Estimation Robuste et Générique du bois Énergie (translated: Development of Models for Robust and Generic Estimation of Wood Energy).

To carry out this operation, foresters needed software to collect data gathered in the field, perform volume calculations, and then store them in a database, allowing for statistical analysis and tracking of activities. The platform was named EmergeMobile and consists of an Android application for data submission to the server and a web interface for data retrieval and manipulation.

The project arose from the client's need to standardize the application in accordance with new internal conventions, improving its functionality and adaptation to field operations. Since the client lacked IT skills and there was no development team on-site, the project was carried out entirely by myself in collaboration with Maxent, a colleague from the computer science department of the Université Savoie Mont Blanc. The support of the popular artificial intelligence ChatGPT by OpenAI was also of fundamental importance.[1]

[1] Research organization on artificial intelligence aimed at promoting and developing friendly artificial intelligence so that humanity can benefit from it.

# 2 A general overview

## 2.1 Field Work

During the project development, the IT team was invited to the field to observe the use of the application during a typical forest worker's day.

Equipment is of paramount importance. The company provides them with fire-resistant, sturdy, and waterproof clothing, anti-distortion boots, a laser rangefinder to measure heights, a forestry caliper for circumferences, and an axe with a raised stamp.

Forests are not always easily accessible by car, so in the first phase, the workers meet in an accessible area to establish the day's mission and, if necessary, assess the terrain conditions. A first visualization of the map, either printed or digital, divided into areas and sub-areas, is made. The supervisor usually suggests a division of the areas to be examined among the workers, agreeing on the strategy for moving through these zones. For this task, it is often necessary to operate outside easily accessible trails, facing terrains that may present difficulties due to environmental conditions or slopes.

Trees are designated based on visual analysis, determining whether they need partial pruning, complete pruning, or maintenance until the end of their life. As mentioned earlier, it is important to ensure the presence of micro-habitats, visible to the naked eye, on the trunk of the tree to guarantee biodiversity within the forests. Indeed, many animals create dens to shelter, bring food, and raise their offspring. According to experiments conducted on some observers, there is generally a conformity rate close to 100% in intercepting animal dens and close to 0% in inventing them. However, there may be some difficulty in distinguishing them, but in the case of this operation, only establishing their presence matters. It is interesting to note that fatigue does not impact the results, as it is a very physical job, so the results of the day can be relied upon. It can be deduced, therefore, that visual analysis alone is very reliable. [1]

Once the tree to be preserved is selected, a triangular mark is made on the trunk surface, sometimes painted, sometimes engraved, to indicate the need for preservation until natural death. Engraving is usually preferred for its permanence, while paint is estimated to last about 15 years and is used when the operator's physical conditions are insufficient to make the scratch. When considering shrubs to be pruned, the group is taken into account, considering various distances, heights, and health conditions. The aim is to maintain the groups according to their average size. When signs of disease are intercepted, good practice is to remove the part in poor condition to promote the plant's recovery. Particular attention is paid to the condition of trees near practiced trails, possibly scheduling their complete removal.

After selecting the tree to be pruned, there are usually at least two operators present, one to measure its height using the laser rangefinder, capable of returning the result in meters according to the angle of inclination created during the measurement (taken at a distance of about 15 meters for good precision), and the colleague nearby to measure the circumference and mark the trunk. Three

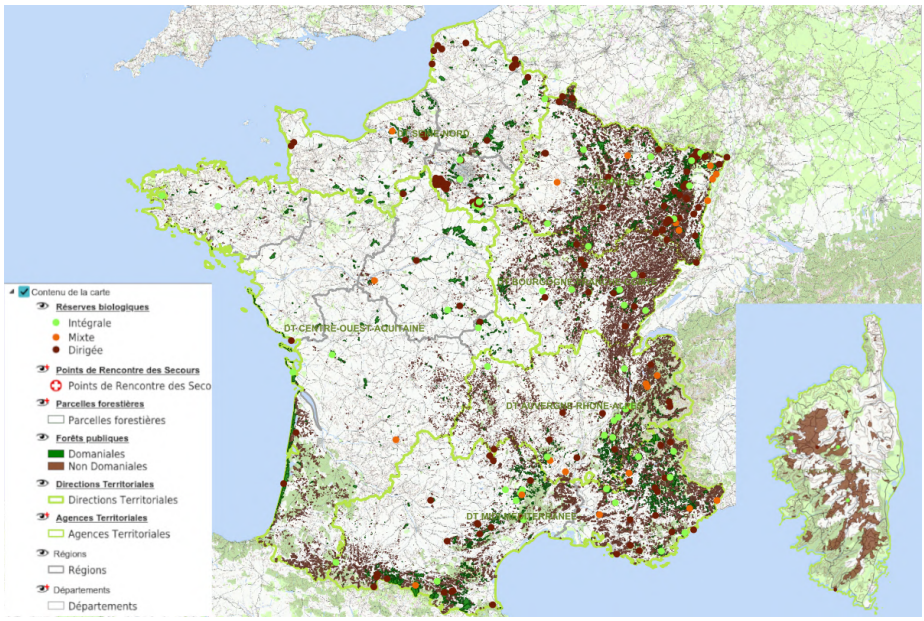



Figura 1: Map of forests managed by the ONF in metropolitan France.

marks are made with the axe by removing the first layer of bark, one upstream, one downstream, and one at the base of the trunk. At the end, a blow of the axe is given from the stamp side on the skinned area, which will become clearly visible on the surface after a few minutes. This ensures the distinction between marks made by the organization rather than those that an animal or even another person/organization might make. Upon completion of the procedure, it is often the operator who took the distance measurements who records the data in the application. In Chapter 2.4.1, the protocol for performing these operations will be detailed.

Once the inspection is completed, there is a final meeting of the foresters for a brief summary and discussion of the results proposed by the application regarding the classification of the sampled tree groups, according to the Algan scale, which will be discussed in Chapter 2.4.3.

## 2.2 The division of the French territory

In France, the territory is divided into regions, which are further divided into departments, and then into municipalities. The lands can be either domainal, meaning that authority and control over them are directly exercised by the government, or non-domainal, referring to territories where a state does not exert complete or direct control. In Figure 1, the map shows the forests traced by the organization.

The ONF helps maintain areas, often by selling its services to owners of non-dominal lands and, in the case of martelage, also allowing clients to profit from the sale of felled wood.

To ensure the congruence of the database, an analysis of the zones and their divisions was carried out, considering any hypothetical administrative changes. The zones are divided as follows:

- **Territorial Units:** These are the macro areas, not always confined within a single municipality and generally not subject to changes, except in cases where the forest is completely deforested or for hypothetical administrative reasons. Each forest ranger is assigned one that identifies their work area.
- **Forests**: These are the forested areas of the territorial units and are used to identify the work area of a daily mission.
- **Plots**: These are specific areas within the forests, used for team coordination during missions. Operations are carried out on multiple plots of land during the same day, and this division also helps track any incomplete work due to superior causes.

## 2.3 Trees in the Territory

The French metropolitan forest has been expanding over the years. According to the 2017 forest inventory by the National Institute of Geographic and Forest Information (IGN), its area has increased by 0.7% per year since 1985. From 14.1 million hectares in 1985, the metropolitan area now boasts 17 million hectares of forest.

With 190 species of trees, the French metropolitan forest represents nearly 75% of the species found in Europe (see Figure 2). Even the overseas departments and regions (DROM) [2] are rich in ecological diversity, with over a thousand species of trees, a diversity explained by the variety of environments.

Trees are categorized into two main categories: **broadleaves**, characterized by wide leaves, and **conifers**, also called needle-leaved or evergreens, characterized by resin production.[3].

[2] acquired territories from the French colonial empire outside metropolitan France

[3] A sticky substance useful for plant defense or healing from stress caused by environmental factors is called resin.

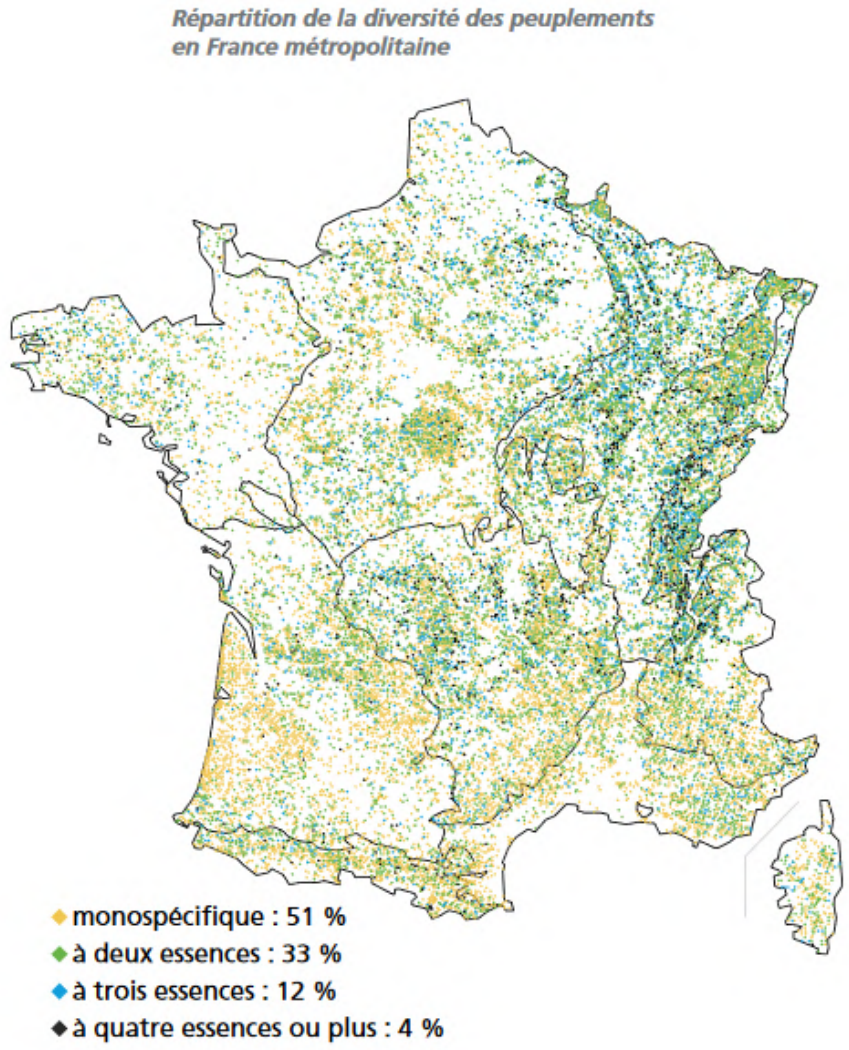



Figura 2: Distribution of tree species diversity in the French metropolitan territory.
*Information regarding the composition and diversity of populations is only taken from populations with tree cover with diameters larger than 7.5cm, exceeding 15%. They cover an area of 14.8 million hectares.

In the country, 65% of the plants are broadleaf species, such as oaks, beeches, and chestnuts, while for conifers, the most common species are maritime pine, Scots pine, spruce, and fir. These two types of species represent 82% of the tree population in the territory.[2]

## 2.4 The Volume of Trees

The ultimate goal of the platform is to calculate the volume of designated trees, which is the amount of wood volume present in a single tree or a group of trees. It involves a statistical relationship (a model) established on a sample of trees, using easily measurable variables on a standing tree.

### 2.4.1 The protocol

Each forest worker is provided with a protocol to perform their job effectively:

- **Sampling**: A sample collected from approximately thirty trees of the same species is considered reliable. They are generally chosen randomly towards the central area of the plot.
- Taking **measurements** visible in figure 3:
  - For the **diameter**, measurements are taken in centimeters at two points: at the base of the trunk and at 1.30 meters from the ground ($c_{130}$). The average of the two measurements will be the diameter to be entered.
  - The **height** ($h_{tot}$) is measured up to the final shoot, taking into account a 10-centimeter tolerance of the rangefinder on the result.
  - The **height of commercial decrement** (optional) is taken with the rangefinder or by eye and only if one wants to compare the ONF estimate with two entries, thus with the estimate of two *mercurials*, with that of EMERGE.
- **Equipment**, as mentioned in chapter 2.1:
  - calibrated forest caliper in centimeters;
  - laser rangefinder, Vertex brand, and its transporter;
  - a notebook for data collection or the application;
  - guide for taking measurements;
  - 10-meter tape for calibration.
- **Data management**: The tariff choice will be calculated and displayed at the end of the information collection and can be modified if necessary.

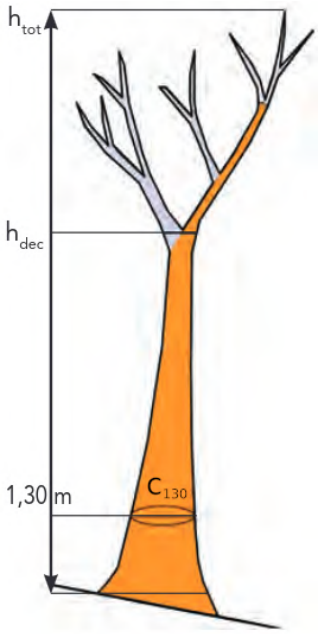



Figura 3: Graphic illustration of tree measurement procedures.

### 2.4.2 Volume calculation model.

"After gathering the necessary information, it is possible to develop the mathematical model for calculating the volume of wood, following dendrometric calculations[4] of EMERGE developed by the ONF. [3] To model the complete volume of the stem (in $m^3$), called 'StemVolume,' it is necessary to go through the redefined shape coefficient (unitless)."

First correction:

$$[1]\text{newStemShapeCoefficient} = \text{stemShapeCoefficient} \left(1 - \frac{1,30}{h_{tot}}\right)^2$$

con

$$\text{stemShapeCoefficient} = \frac{4\pi \cdot \text{stemVolume}}{h_{tot} \cdot c_{130}}$$

where $h_{tot}$ and $c_{130}$ are expressed in meters.

The robust wood volume of the stem (referred to as "VBFT") is chosen instead of the complete stem volume. The second correction is:

$$[2]\text{VBFT} = \text{stemVolume} \cdot \left(1 - \frac{c_{BF}^3}{c_{130}^3} \cdot \left(1 - \frac{1,30}{h_{tot}}\right)^3\right)$$

[4]The science that studies the techniques for calculating the biometric values related to a tree or a forest.

with $c_{BF}$ = Robust wood circumference ($c_{BF} = 0{,}07\ \pi$)

In conclusion, the shape coefficient is therefore redefined as follows:

$$[3] newStemShapeCoefficient = \frac{4\pi \cdot VBFT}{h_{tot} \cdot c_{130}} \cdot \frac{\left(1 - \frac{1{,}30}{h_{tot}}\right)^2}{\left(1 - \frac{c_{BF}^3}{c_{130}^3} \cdot \left(1 - \frac{1{,}30}{h_{tot}}\right)^3\right)}$$

Now, it needs to be modeled as a function of the total volume of the tree (or rather the corresponding shape coefficient, formTotNew) and the explanatory variables of shape differences. The model is expressed as follows:

$$[4] newStemShapeCoefficient = \left(d + e \cdot \ln(\frac{h_{dec}}{h_{tot} - h_{dec}}) + f\frac{\sqrt{c_{130}}}{h_{tot}} + \frac{g}{c_{130}}\right) \cdot formTotNew$$

The parameters $d$ and $e$ are dimensionless, $f$ is expressed in $m^{\frac{1}{2}}$ and $g$ in $m^{-1}$.

After the statistical adjustment, the model for the robust wood volume of the stem is deduced:

$$[5] VBFT = \underbrace{\frac{h_{\text{tot}} \cdot {c_{130}}^2}{4\pi} \frac{\left(a + b \cdot \frac{\sqrt{c_{130}}}{h_{\text{tot}}} + c \cdot \frac{h_{\text{tot}}}{c_{130}}\right)}{\left(1 - \frac{1{,}30}{h_{\text{tot}}}\right)^2}}_{\text{Total volume of the tree}} \times \underbrace{\left(d + e \cdot \ln\left(\frac{h_{\text{dec}}}{h_{\text{tot}} - h_{\text{dec}}}\right) + f\frac{\sqrt{c_{130}}}{h_{\text{tot}}} + \frac{g}{c_{130}}\right)}_{\text{Trunk part}} \times \underbrace{\left(1 - \frac{c_{\text{BF}}^3}{c_{130}^3} \cdot \left(1 - \frac{1{,}30}{h_{\text{tot}}}\right)^3\right)}_{\text{Robust wood correction}}$$

Generalization of the robust wood correction to any sectioning:

$$[6] stemVolume\ (c_{decoupe}) = stemVolume \cdot \left(1 - \frac{c_{decoupe}^3}{c_{130}^3} \cdot \left(1 - \frac{1{,}30}{h_{tot}}\right)^3\right)$$

Where

$$\text{stemVolume} = \text{Total volume of tree} \cdot \text{Trunk part}$$

### 2.4.3 Obtained data

The model allows obtaining the following three final pieces of information:

- **Volume of commercial decremental stem**, measured in $m^3$, is the volume of the stem from the base to the horizontal axis of the commercial decrement, obtained through the formula [6] in Chapter 2.4.2.

- **Crown volume**, determined according to the sales records of timber from public forests, the crown volume corresponds to the difference between the volume of the commercial decremental stem and the total volume of robust wood.
- **Total volume of robust wood**, calculated in $m^3$, is the volume from the base of the plant to the 7 cm decrement of the main horizontal axis and branches. It is calculated by summing the volume of the commercial decremental stem and the crown volume. Also determined according to the sales records of timber from public forests and is generally the quantity put up for sale.

With the calculation of the model, it is also possible to perform a graphical analysis by comparing the sample taken to the Algan (Figure 4) and Schaeffer rates. The former is more suitable for irregularly shaped logs, while the latter, characterized by simplicity of calculation, is excellent for logs of regular shape. Within the application, the selection will only be possible on the Algan scale, as it is more commonly used and versatile.

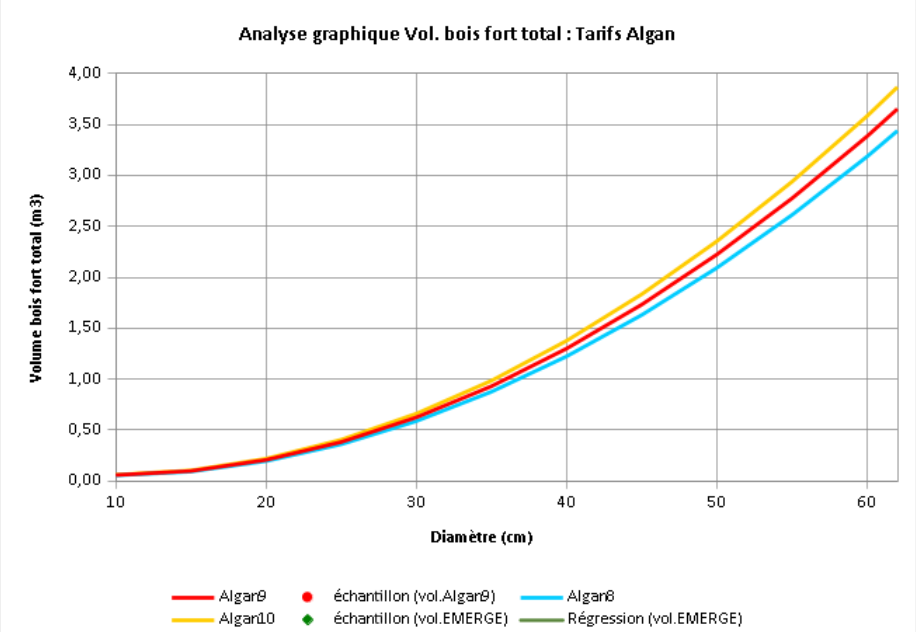



Figura 4: Graph, not filled in, comparing the result of the EMERGE sampling (in green) to the values of the Algan scale.

These analyses are essential for assessing and managing the woody resources of a forest. They help in planning timber harvesting activities, estimating revenues from wood sales, ensuring sustainable use of forest resources, and contributing to the responsible management of forests.

Except for the Douglas fir, whose different shape requires classification with a different type of tariff.[5]

## 2.5 The pre-existing web application

The project was initially developed by a group of university students and was implemented using the following technologies: Angular, a JavaScript framework, was adopted for web front-end development, while JavaScript and Kotlin were used for the back-end and the Android application, respectively. Additionally, for data storage and website hosting, the company purchased an external service called O2Switch[6], where the MySQL database was managed through the PhpMyAdmin interface.

The Use of the WebApp The use of the mobile application is designed to collect real-time data in the field. Each operator must specify their territorial unit, which enables them to carry out activities in the respective forests. To input the dimensions of the tree, operators select the forest lot to which the plant belongs and then enter parameters such as type, height, width, and estimated height of decrement. Once the plant collection is completed, the application suggests a tariff, as discussed in Chapter \ref{cap:gestione_dati}, which can be subsequently changed by the user. Upon completion, the detection is sent to the server, and for safety, a local copy is created on the phone in JSON format, which can be imported via the web interface. Additionally, there is the option to transfer monitoring to another operator within the same territorial unit.

The web interface is used for data manipulation and extraction. Through the website, users can import, view, or retrieve any work data. Administrators can manage the roles and tasks of other active users and modify administrative data related to the territories.

### 2.5.1 Bugs and Issues with the Existing System

With the use of the application, operators have reported several malfunctions and possible improvements. The main issues encountered with the Android application were as follows:

- During field operations, operators often face significant internet connection limitations on their devices, causing the application to become unusable.
- After entering more than 20 instances of the Douglas fir, the application would crash abruptly.
- Upon the first failure to send the survey to the server, the application would experience an abnormal crash.

[5] A Douglas fir, an evergreen conifer, is a majestic tree, among the tallest in the world in terms of height reached.

[6] A French company specializing in offering web hosting services, domain name registration, and other related services.

- No validation was performed on the assigned territorial unit, allowing the input of a non-existent number.
- Some forests had the same name, making it impossible to distinguish between them.
- Lack of loading feedback, leaving the user with the impression that something was not functioning.

The web application also had some malfunctions:

- Data manipulation functionality was absent, requiring direct access to the O2Switch platform via a browser, opening phpMyAdmin, and making manual changes to the database, risking inconsistencies that could lead to future issues with the proper functioning of the application.
- The JSON file import function did not perform any operations.
- The function to modify lots did not alter any data.
- When the administrator changed a user's role, it was not updated.
- Similar to the Android application, there was a lack of loading feedback.

# 3 The Role of Technologies in Development

The employed technologies included React, Kotlin, SpringBoot, and PostgreSQL, chosen based on the client's requirements to adhere to internal conventions. Later on, we will analyze their good compatibility and their application. Additionally, other supporting technologies were adopted for development, such as the ChatGPT artificial intelligence, development environments, and code sharing platforms.

## 3.1 React: front-end development

React, also known as React.js or ReactJS, is an open-source JavaScript framework for building user interfaces maintained by Meta and a community of individual developers and companies. The latest version of React enables programming 'functionally,' adhering to a functional programming style, moving away from the native object-oriented approach while still allowing for its implementation. A distinctive feature of this framework is the flexibility of integrating HTML directly into React statements. Each component returns an HTML element associated with any functions, allowing for the writing of HTML and React code within the same file. For this reason, JSX files, whose acronym stands for 'JavaScript XML,' are used during React development.

```
const ExampleComponent = () => {
  const condition = true;
  return (
    <div>
      {condition ? (
        <p>Il risultato è vero!</p>
      ) : (
        <p>Il risultato è falso!</p>
      )}
    </div>
  );
};
```

Additionally, as seen in the above example, `if` and `else` conditions are implemented directly within the `return` block, using the same ternary operators as traditional programming languages.

Another characteristic feature of this framework is the ability to fragment code into components, which are elements that perform a specific function and can be called multiple times within the code. Calling a component is done within the HTML part of the code.

```
import MyComponent from './MyComponent';

const App = () => {
  return (
```

```
    <div>
      <h1>Questo è il componente App</h1>
      <MyComponent />
    </div>
  );
};

export default App;
```

In this short example, a component is imported into a variable named `MyComponent`, and within `App`, it is called using the command `<MyComponent/>`. `App` is, in turn, a component and to be imported into another, it must have the final notation `export default`.

### 3.1.1 Color Choice

A fundamental preliminary operation for website development is the choice of colors that best fit the context. Various references are widely available online to facilitate this choice, offering specific "color palettes" typically consisting of no more than five colors, carefully matched according to precise design standards based on the color wheel (see figure 5).

The most common ones are as follows:

- **Complementary Color Theory**: This theory is based on using opposite colors on the color wheel. For example, red is complementary to green. Complementary palettes offer a strong visual contrast.
- **Analogous Color Theory**: This theory involves using adjacent colors on the color wheel. For example, blue, blue-green, and green are analogous colors. Analogous palettes create a harmonious and balanced look.
- **Triadic Color Theory**: This theory uses three equidistant colors on the color wheel. For example, red, yellow, and blue form a set of triadic colors. Triadic palettes offer a good balance between contrast and harmony.
- **Split-Complementary Color Theory**: This theory involves the base color and the two colors adjacent to its complement. For example, if the base color is blue, the complementary colors will be yellow and orange. This creates a strong but less intense contrast compared to the complementary combination.
- **Square (Tetradic) Color Theory**: This theory implies using four equally spaced colors on the color wheel. This combination offers many possibilities for variation and flexibility in design.
- **Monochromatic Palette**: It uses different shades, tones, and intensities of the same color. This approach offers a very harmonious and uniform appearance.

- **Neutral Palette**: A palette that uses neutral colors such as white, black, gray, beige, and brown. Neutral colors often serve as a background or complement to other more vibrant colors.

Figura 5: Color wheel.

### 3.1.2 Node.js

In the current part of the project, a second technology, Node.js, has been paired. It is an open-source, cross-platform runtime environment based on JavaScript that enables server-side execution. Built on Google Chrome's V8 engine, it ensures notable execution efficiency in a browser environment.

Google Chrome's V8 is an open-source system for executing web page scripts, known for its efficiency in low-resource execution. Leveraging its freedom of manipulation, Node.js developers had the idea to decouple the engine from the browser and instead use it within a server or computer. This allowed JavaScript to access file systems and interact with a database, making it useful for those choosing this language for the backend of their website.

Node.js is inherently asynchronous and uses a non-blocking, event-driven I/O model. Typically, a blocking system needs to handle each request until its completion, thereby occupying the CPU with active waiting for the database response. In this case, it is called "non-blocking" because the process handling the requests never waits for their completion, instead creating callbacks, providing the database with instructions to execute, and managing the next operation. When the database completes the instruction, it provides feedback to the calling process, which then provides the output. This mechanism has proven to be highly performant and adaptable even to less powerful processors.

Npm (Node Package Manager) is currently the largest ecosystem of open-source libraries in the world. Its advantage lies in the ease of installation and code reuse. Simply invoking the `npm install` command within the terminal is enough to download it, and it is available to anyone, thus reusable in different contexts.

### 3.1.3 Virtual DOM

Typically, websites have the Document Object Model (DOM), an object model that composes the structure and content of a document on the web. It serves to interact dynamically with the content of a web page. Using JavaScript, developers can access, modify, add, and delete HTML elements and attributes, as well as handle events like clicks, focus, and more. React uses a concept called Virtual DOM to minimize direct manipulations on the DOM and improve rendering efficiency. The Virtual DOM represents a "lightweight" copy of the DOM structure that is made at every state and/or property update of a component, then compared to the previous DOM and, thanks to a difference algorithm called diffing, only the changes are applied to the real DOM, avoiding its direct and total modification at every change, making the user interface significantly faster.

### 3.1.4 Project Organization

For the organization of the project, the following folder division was chosen:

- **classes**: to facilitate abstraction, it was chosen to create two classes representing shared objects that maintained the concept of an object;
- **components**: components are the shared elements reused by the pages;
- **pages**: container of each page of the website;
- **resources**: container of all external elements, such as images, videos, and other multimedia content;
- **style**: container of style sheets (CSS).

Files outside these folders are global files useful for launching the application. Additionally, a global file called "config.js" was created, which contains the IP address of the server to which API requests are made, very useful for easily switching servers, in this case, from the test server to the production server.

### 3.1.5 React Hooks

One of the distinctive advantages of React is represented by "Hooks," introduced in version 16.8 of the framework [4]. Hooks are special functions that allow integration into the state and lifecycle of React components, making it possible to use React without the need for classes. The most commonly used hook is `useState`, where each variable is initialized with an array of two elements, the variable itself, and the function to modify it.

```
const [email, setEmail] = useState('')
```

To modify its value, you simply need to pass the desired value to the `setEmail` method. It works the same way with all other types, implicitly declared.

```
const handleChangeEmail = e => {
    setEmail(e.target.value)
};
```

In this case, by passing the received value into a text box.

Another Hook used was `useEffect`, which allows the execution of side effects upon the initial rendering of a component or when certain associated variables change. With this function, you can update lists without having to reload the page, enabling real-time updates. It's important to use a unique key and index in lists. If duplicate keys are used, the corresponding list element will also be duplicated accordingly. This mechanism ensures the uniqueness of elements within the list. Below is an example of updating a list.

```
useEffect(() => {
    fetchStatementList()
}, [deleted])
```

The variable `deleted` is used as a switch and changes with each call of the method used for deleting an item. Each time its value changes, subsequent to the request for deletion of the item from the database, the function `fetchStatementList()` is invoked to retrieve the current state of the list, updating it without reloading the page. Additionally, it was discovered that the string comparator, to ensure correct sorting of the list on every browser, must return 1 in the positive case and -1 in the negative case. This is because Firefox accepts the `true` or `false` notation, but Chromium-based browsers, which are more sensitive, have cases 1, 0, and -1. Therefore, by adopting the former notation and executing the code on Chrome, the list will not be sorted.

During development, a common problem in React was encountered, related to the decomposition of the code of the same element, which caused components to re-render with each action. In this specific context, it resulted in a malfunction of the page, manifested by a rapid flickering lasting less than a second, causing the closing and reopening of open windows, disrupting navigation at that precise moment. To solve this type of problem, `useRef` was adopted. It consists of a variable that holds the previous state of the component, thus avoiding forcing the application to render the entire page again.

```
const TURef = useRef({})
```

In this case, it was used as a memory of the previous state.

```
const [TU, setTU] =
useState
(
    TURef.current[params.territorialunit_internalId]
```

```
    ||
    false
)
```

It is noted that the condition inside the new variable checks for the existence of a state saved within `TURef`, otherwise initialized to `false`.

```
const handleOpenTu = (territorialunit_internalId) => {
    TURef.current[territorialunit_internalId]=
    !TURef.current[territorialunit_internalId]
};
```

Subsequently modifiable, in the example, using an event handler, which sets the opposite value to the boolean variable `territorialunit_internalId` saved within `TURef`.

### 3.1.6 Handling Events

Critical for the proper functioning of the web interface are the event handlers, responsible for managing events. In React, they are invoked in camelCase in the HTML code, unlike traditional HTML:

```
<button onClick={(e) => handleExample}>
    Example
</button>
```

The desired action will be written inside the `handleExample` method. There is also the possibility to handle the event within the function using the arrow notation: `(e) =>`.

The event handlers used in this project are:

- **onClick**: triggered when the element is clicked, used within buttons and clickable elements.
- **onSubmit**: called when a form is submitted, used when the enter key is pressed in the JSON file import function.
- **onChange**: triggered in input, `select`, or `textarea` elements, used in text filtering boxes.
- **onFocus and onBlur**: occur when an element receives or loses focus. These elements greatly help in making a website user-friendly, as they were used to slightly change the color of buttons or elements when the mouse was hovered over them.

In React, it is advantageous that it is easy to pass the parent's handlers as parameters to a child component.

```
<Statement onDelete={handleDelete} onClose={closeStatement}/>
```

Subsequently, allowing within `Statement` in the example, to invoke the execution of the handlers that will be executed by the parent.

```
function Statement(params) {
  const handleClose = () => {
    params.onClose()
    };
[...]
```

The dot notation (`params.onClose()`) is used, common in many other programming languages.

### 3.1.7 API Requests

In React, there are multiple ways to make API requests to retrieve or send data to a server:

- **Fetch API**: It is a built-in JavaScript function in modern browsers that allows making HTTP requests. It is very flexible and can be used to perform GET, POST, PUT, DELETE, and other requests. For example:

  ```
  fetch('https://api.example.com/data')
    .then(response => response.json())
    .then(data => console.log(data))
    .catch(error => console.error('Error:', error))
  ```

- **Fetch with async/await**: It is possible to use the Fetch API in combination with async/await to make the code more readable and handle promises more efficiently. Here's an example:

  ```
  const fetchData = async () => {
    try {
      const response = await fetch('https://api.example.com/data')
      const data = await response.json()
      console.log(data)
    } catch (error) {
      console.error('Error:', error)
    }
  }
  ```

- **Axios**: Axios is a popular JavaScript library for making HTTP requests both on the browser and server sides. It is widely used in React projects due to its ease of use and the many features it offers. Here's an example of how to use Axios:

  ```
  import axios from 'axios'
  ```

```
axios.get('https://api.example.com/data')
  .then(response => console.log(response.data))
  .catch(error => console.error('Error:', error))
```

- **XHR (XMLHttpRequest)**: XMLHttpRequest is a built-in JavaScript object that allows you to send asynchronous HTTP requests. Although it is less commonly used compared to Fetch and Axios, it is still a valid option. Here's an example:

```
const xhr = new XMLHttpRequest()
xhr.open('GET', 'https://api.example.com/data', true)
xhr.onreadystatechange = () => {
  if (xhr.readyState === 4 && xhr.status === 200) {
    console.log(JSON.parse(xhr.responseText))
  }
}
xhr.send()
```

- **Framework-specific libraries**: Some React frameworks or libraries may offer methods to handle API requests, allowing for more structured handling of asynchronous requests within the React app.

In the current context, the first two solutions have been adopted. The fetch method sends an HTTP request to the address within the parentheses, in this example, it's of type GET, returning a promise that resolves with the `response` object, which represents the response. The `then` methods start a callback function, where in the first call, the response is parsed to JSON format, and in the second call, the `data` variable manipulates the response, in JSON format, according to its needs (in this case, printing it to the console log). The `catch` catches any potential error presented and handles it within the parentheses. In these cases, the type of HTTP error received is never discriminated, providing the user with a generic error.

With the `async` keyword, an asynchronous method is created, which does not interrupt the normal flow of the program but returns a "promise" while the `await` keyword allows waiting for its completion within the method. The mentioned mechanism is adopted in the login verification method and in the one for importing JSON files because it is necessary to wait for the HTTP response to verify its correctness and, if necessary, discriminate the received error. For example, in the login, it is important to distinguish the case where the email is nonexistent in the database from the case where there is no internet connection or there is an internal server error, other well-distinct cases.

### 3.1.8 Loading screens

For the loading screens, a component called `Loading` has been created, invoked by the pages and displayed according to a variable that will be set to the value `true` when the connection is established. Inside the component, a timer has

been inserted to delay execution by 1 second, a waiting time considered tolerable before showing that the operation takes longer than expected, and a counter to display the reload page notice after reaching three maximum attempts. `Loading` accepts as input a variable, `attempts`, representing the connection attempts made before declaring that something went wrong.

```
const retryConnection = () => {
    if(attemptsRef.current < 3){
        const interval = setTimeout(() => {
            fetchUTData()
            attemptsRef.current++
            setAttempts(attemptsRef.current)
        }, 3000)
        return () => {
        clearTimeout(interval)
        }
    }
}
```

In this case, an attempt is being made to retrieve territorial unit data from the server. Inside the error handling of the `fetchUTData()` method, a call to the `retryConnection()` method is made, which retries the connection after three seconds. After the third attempt, therefore, after 9 seconds without a positive response from the website, no further connection attempts are made, and the variable in the component is triggered to show the reload page message.

During development, following unexpected behavior of the function, it was discovered that the `<React.StrictMode>` library, which encloses the entire React project, caused inconsistencies with the timer, as the component was called multiple times consecutively, incrementing the count with each call. After several investigations, it emerged that the mechanism just described is a support that React provides to developers to receive errors that may not emerge at the single mounting of a component. This library is active only during development, while it is ignored during the project build.

### 3.1.9 Disadvantages of React

A disadvantage encountered with React is the limited UX (User Experience) support. It is possible to install various third-party libraries dedicated to the interface, but they are often poorly documented or not intuitively usable. In the current project, CSS code was written directly by the development team.

Another encountered aspect is the initial complexity. There are many new concepts to understand, such as the Virtual DOM, component lifecycle, variable states, and props (mechanism for passing data from a parent component to a child component).

Code fragmentation has also been problematic. While it is extremely simple to use and invoke other components, for more complex operations that requi-

re many lines of code, there is difficulty in dividing operations into multiple functions.

## 3.2 Spring Boot: Backend Development

Java Spring Boot is an open-source tool that simplifies the use of Java-based frameworks for creating services and web apps.

There are two general approaches, creating methods that send direct SQL queries or creating repository classes that offer predefined methods to make queries to the server. In the present case, the first approach was chosen as the main one, considering the greater initial knowledge of PostgreSQL compared to that of Spring Boot, although the second methodology is adopted for some operations. The advantage of sending queries directly to the database is to have full control over the requests, allowing for efficiency improvement and providing better customization. In this way, the side effects are the length of the code, the need to pay attention to the possibility of SQL injection attacks, managing some characters interpreted by PostgreSQL, such as the apostrophe, and a greater number of errors to be handled manually.

In the specific context, for communication with the database, the Spring JDBC (Java Database Connectivity) library has been adopted, an API that enables the transfer of information.

### 3.2.1 Annotations

Annotations in Spring Boot are metadata added to the source code to provide information and instructions. Their use is fundamental for the correct startup of the project. In the current context, the following have been used:

- **@SpringBootApplication** is the main annotation. It is inserted into the primary class to start the Spring Boot application.
- **@Autowired** is used for dependency injection. Spring Boot manages the initialization of objects marked with this annotation.
- **@RestController** returns data in the body of the HTTP response.
- **@RequestMapping("/")** is used to map an HTTP request to a specific method of the controller. Between the parentheses, the path corresponding to a specific method is inserted.
- **@CrossOrigin(origins = "*")** is used to enable cross-origin requests (CORS) from a specific domain. It is a security measure to prevent requests from domains other than those intended. In the illustrated case, requests from any domain are accepted.
- **@PostMapping("/url")** maps a controller method to a URL path handling the POST method on that path.

- **@GetMapping("/url")** maps a controller method to a URL path handling the GET method on that path.
- **@RequestBody** is used to map the body of the HTTP request to a Java object.
- **@RequestParam("parameters")** is used to extract parameters from the HTTP request and map them to the method's parameters.
- **@ResponseBody** indicates that the return value of a method should be serialized directly into the body of the HTTP response.
- **@PathVariable** is used to extract values from URL parameters.
- **@Repository** indicates that a class is a repository.

### 3.2.2 Project Organization

The project has been divided as follows:

- **controller:** divided into three controllers, one for POST methods (requests to insert or modify data in the database), one for GET methods (requests to obtain information), and one for token creation;
- **mapRow:** contains classes for each table that allow mapping of data in a format compatible with the Android application;
- **model:** objects used by controller methods to contain, manipulate, and send information;
- **repository:** interfaces that handle SQL requests in a non-explicit manner, following Spring Boot's predisposition;
- **service:** classes offering global services to the application:
  - **databasequeries:** contains classes for sending queries to the database of methods present in the controller;
  - **excelsheets:** service for exporting data in XLSX format.

Outside of the folders, there is a configuration file useful for the proper execution of the application and the Maven configuration.

### 3.2.3 Project Configuration

Within the project, there is a configuration file containing information to allow communication with the local database.

```
spring-boot.run.fork=true
```

The parameter `spring-boot.run.fork` controls whether the application should be run in a separate JVM or in the same JVM where Maven was started. In this case, since the server hosts multiple applications, it was decided to isolate their executions to avoid conflicts.

```
server.port=8081

spring.datasource.url=jdbc:postgresql://localhost:5432/Emerge
spring.datasource.username=postgres
spring.datasource.password=postgres
```

The chosen port is 8081, which is the port queried by web and Android applications.

Below is the configuration of the PostgreSQL database, hosted on the same server from which the back-end application will be launched. For this reason, the IP address is set to "localhost". The database name is "Emerge", and username and password are also set. This file is not uploaded to GitLab for security reasons.

```
spring.servlet.multipart.enabled=true
spring.servlet.multipart.max-file-size=20MB
spring.servlet.multipart.max-request-size=20MB
```

This configuration refers to the upload function. It allows uploading files up to 20 megabytes maximum, which is also chosen as the limit for the entire request, including any related data.

### 3.2.4 Protocol for Client-Server Communication

Spring Boot offers various communication protocols between client and server. The one chosen in this project is the most common, namely HTTP (Hypertext Transfer Protocol).

### 3.2.5 Error Handling

Knowing its structure made it possible to view the body of the received responses and any errors through the browser interface. The main errors encountered during development were:

- **400 Bad Request**: The server cannot understand the request due to incorrect syntax or a malformed request from the client.
- **404 Not Found**: The server cannot find the requested resource, or the specified URL is not valid.
- **403 Forbidden**: The server understands the request but refuses to authorize access to the requested resource. The client does not have the necessary permissions.

- **500 Internal Server Error**: It is a generic error indicating an issue internal to the server without providing specific error details (which can be viewed from the Spring Boot application execution console).
- **505 HTTP Version Not Supported**: The server does not support the HTTP protocol version used in the request.

The **200** response indicates that the HTTP request has been successfully processed by the server.

### 3.2.6 Structure of API Requests

Each request includes a string containing the text of a query. Requests are divided into two categories: those that modify the database and those that retrieve information, respectively POST and GET requests.

Example of a GET request:

```
@GetMapping("/databaseEditing/getAllForests")
@ResponseBody
public ResponseEntity<String> getAllForests() {
    String sql = """
                    SELECT fo.*
                    FROM forests fo
                    ORDER BY fo.internalid
                 """;

    SelectQuery query = new SelectQuery(jdbcTemplate,sql);
    return query.getHttpJson();
}
```

The data retrieval is performed within a method of type `ResponseEntity`, which is the class needed to represent HTTP responses sent by the server. The SQL string is inserted into an object of type `jdbcTemplate`, a class provided by the library for the automatic handling of operations, usually manual, to be performed on a database (e.g., opening and closing connections). This allows the `fetch` operation (querying the database) to retrieve the necessary data (performed within the `SelectQuery` class). Finally, to check the status of the response, which will be further discussed in chapter 3.2.4, the `HttpHeaders` object is used, which contains the HTTP headers for the proper completion of the request. The response is in JSON format, which will be subsequently passed as a `String` to the front-end.

Example of a POST request:

```
@PostMapping("/promoteModerator")
@ResponseBody
public void promoteModerator(@RequestParam("user_internalid")
int user_internalid) {
    String query =
```

```
            "UPDATE users u " +
            "SET isModerator=true, updatedAt =
            Current_timestamp(0) " +
            "WHERE u.internalid = " + user_internalid;
    InsertQuery sql = new InsertQuery(jdbcTemplate, query);
    sql.send();
}
```

Similarly to the analyzed GET request, the process is the same for POST requests except for its conclusion. In this case, the method is of type `void`, and the `send` method is used to send modification requests to the database.

Example of a POST request:

```
 public boolean send() {
    int actionStatus = jdbcTemplate.update(query);
    boolean actionSucceeded = actionStatus > 0;
    return actionSucceeded;
}
```

The string is inserted into an object of type `jdbcTemplate`, as done previously, and then the `update` command is executed on the created object to confirm the update in the database.

To adapt this logic to the structure of the Android application, a slightly different strategy was implemented. The `RowMapper` classes were created. These are interfaces useful for mapping a single row of a SQL query result to a Java object. Mapping allows the application to correctly match the received information with the columns of the local database.

```
@GetMapping("users/fromUt/{userUT}")
@ResponseBody
public ResponseEntity<List<User>> getUserMail(@PathVariable
String userUT) {
    String sql = """
                    SELECT u.email, t.ccod_ut AS userUT \s
                    FROM users u, territorialunits t\s
                    WHERE u.territorialunit_internalid =
                    t.internalid AND t.ccod_ut = ?""";

    List<User> users = jdbcTemplate.query(sql,
    new UsersRowMap(),
    userUT);
    return ResponseEntity.ok(users);
}
```

In this case, the structure is similar to a typical GET request, with the difference that the output is a list of serialized objects of type `User`, mapped as follows.

```
public class UsersRowMap implements RowMapper<User> {
```

```
    @Override
    public User mapRow(ResultSet rs, int rowNum)
    throws SQLException {
        User user = new User();
        user.setEmail(rs.getString("email"));
        user.setUserUT(rs.getString("userUT"));
        return user;
    }
}
```

This type of structure allows sending the application information in a compatible format, ensuring correct interpretation. In the mobile app's database, the 'User' table will have columns named `email` and `userUT`.

The function of annotations was addressed in Chapter **??**.

### 3.2.7 Importing JSON Files

The method for importing JSON files into the data store is designed to also accept JSON files from the previous version, thus without the fields added with the data migration, which in their absence will assume null or false values for boolean variables. The library that facilitated file processing is `Jackson ObjectMapper`. It is used for serialization and deserialization of Java objects into JSON format and vice versa. In this case, it is used to transform the contents of the `Multipart` file into a JSON object for subsequent processing.

```
[...]
ObjectMapper objectMapper = new ObjectMapper();
JsonNode node = objectMapper.readTree(file.getInputStream());
String userMail = node.get("userMail");
JsonNode statement = node.get("statement");
[...]
```

In the fragment of the method shown, you can see the creation of a new object of type `ObjectMapper`. The method, from the library, called `readTree`, allows reading the information contained in the file via an input stream. Once the variable of type `JsonNode` is obtained, it is possible to access the information using the `get("variable_name")` method. In the example shown, a string is retrieved for the variable containing the email and another `JsonNode` of a nested JSON object. Subsequently, after retrieving the information within the variables, everything is inserted into the SQL queries using the same procedure as the POST methods analyzed in Chapter 3.2.6.

### 3.2.8 Token Generation

For the token return, a check is first made on the list of existing users. Within this operation, for simplicity, a repository class is preferred, the only use made within the application, to search for the user within the list in the database. In case an nonexistent email is inserted, an error will be returned:

`HttpStatus.UNAUTHORIZED`. In the affirmative case, instead, a token will be generated with the following information:

- id
- email
- admin status
- moderator status
- accepted status
- the token's signature.

The signature is made with the HS256 signature algorithm and a secret key, therefore not shared, ensuring that the JWT (JSON Web Token) can be verified in the future using the same secret key.

```
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.security.Keys;

private SecretKey generateKey() {
    return Keys.secretKeyFor(SignatureAlgorithm.HS256);
}

private String generateToken(
    Integer id,
    String email,
    boolean isAdmin,
    boolean isModerator,
    boolean isWebAccepted) {

    SecretKey secretKey = generateKey();

    String token = Jwts.builder()
            .claim("id", id)
            .claim("email", email)
            .claim("isAdmin", isAdmin)
            .claim("isModerator", isModerator)
            .claim("isWebAccepted", isWebAccepted)
            .signWith(SignatureAlgorithm.HS256, secretKey)
            .compact();

    return token;
}
```

For the proper execution of the function, the import of three libraries is necessary. In order, the first one is used to insert the necessary information inside the token to return it in JWT format, the second one contains the signature algorithms used to digitally sign the token, and the last one provides static methods for generating keys used for token signing and verification.

### 3.2.9 Exporting XLSX Files

The functions for exporting start a download from the server to the client's computer of an XLSX format file.

In the project, there are three cases of file export:

- the data collection present within the range of two dates;
- individual readings;
- the archive of deleted data.

For archiving deleted data, it was chosen to permanently save a file, in XLSX format, inside the server, where information will be progressively added, while for exporting existing files, a temporary XLSX file is created, which will then be downloaded by the user and deleted locally. To distinguish the download of a single reading or multiple readings within a two-date range, the overloading property of Java is exploited, i.e., the possibility of having multiple constructors of the same class but with different signatures (i.e., types and order of parameters).

For generating XLSX files, the `ExcelSheet` library was used. The use of this class was employed for creating the format of the sheet required by the client, where for each page and for each column, a customized attribute can be inserted. The manipulation of files, thus compiling the cells of Excel sheets with information retrieved from the server, is possible thanks to the `XSSFWorkbook` class, a library that allows working with all Microsoft Office files.

Once the export command is initiated, an object called `ExcelFileManager` is created, representing the creation of the file. Inside it, methods for file manipulation are implemented, with the help of the previously mentioned libraries. Below is the function to create the file, which is the first to be called:

```
private void createFile() {
    try {
        XSSFWorkbook newWorkbook = new XSSFWorkbook();
        for (String sheetsName : sheetsNames) {
            newWorkbook.createSheet(sheetsName);
        }
        addColumnHeaders(newWorkbook);
        File newFile = new File(getPath());
        newFile.createNewFile();
        FileOutputStream outputStream =
        new FileOutputStream(newFile);
```

```
        newWorkbook.write(outputStream);
        outputStream.close();
        newWorkbook.close();
    }catch(Exception e) {
        throw new RuntimeException(e);
    }
}
```

The `newWorkbook` object represents the file in an abstract manner. The `sheetsNames` variable is an array of strings containing the names of the tables; a page is created for each table within the file. Subsequently, the `addColumnHeaders` method sets the cells of the header row in a different style format to ensure better visibility. Using two loops, for each page and for each column, the first cell is set to the passed value using an array containing default values.

In practice, with the `File` class, the physical file is saved within the path provided by `getPath()`, containing a local path on the server. In Java, writing to memory is done using a byte stream. For this reason, a `FileOutputStream` is opened to write to the device's memory, which is then closed. In case of an error, a `RuntimeException` is thrown with the error message printed to the console.

The compilation of the created XLSX file follows a similar logic. Information to be inserted is stored in arrays, retrieved with SQL queries, following the same methodology seen in Chapter 3.2.6. Subsequently, with appropriate index control, the information is inserted into the corresponding cells using the following commands:

```
Cell currentCell = row.createCell(attributeIndex);
currentCell.setCellValue(attributeValue);
```

Using the `Cell` library, the `createCell` method is called on the selected row at the `attributeIndex` index, virtually creating a new cell. The data, contained in the `attributeValue` variable, is inserted using the `setCellValue` method.

Upon completion of the file preparation, the following method is executed to initiate the download:

```
public void makeHttpOutput(HttpServletResponse response)
throws IOException{
    InputStream is = new FileInputStream(getPath());

    response.setHeader("Content-Disposition",
    "attachment; filename=\""+ exportFileName +".xlsx\"");
    org.apache.commons.io.IOUtils.copy(is,
    response.getOutputStream());
    is.close();
    response.flushBuffer();
    response.getOutputStream().close();
}
```

The `response` variable represents the HTTP response to be provided. Unlike writing to memory, reading requires opening an input stream using the `InputStream` class. The `is` object represents the file to be provided in the response. Subsequently, the HTTP response header is set using the `setHeader` method. The command `"Content-Disposition"` specifies whether the content should be displayed in the browser or treated as a downloadable file, and the second variable indicates that the content should be treated as a downloadable file with the file name setting. The `org.apache.commons.io.IOUtils.copy` instruction, from the Apache Commons IO library, performs a byte-by-byte copy from the input file to the output stream of the HTTP response. Finally, the byte stream is closed to release resources, and the `flushBuffer` method is called to ensure that all data has been written to the output stream, which is then closed as well.

Except for the method for downloading the archive file, in the other requests, the `delete()` method is subsequently called on the downloaded file to permanently delete it from the server.

## 3.3 Kotlin: Android Development

Kotlin is a high-level, general-purpose, multi-paradigm, and open-source programming language developed by JetBrains. It is designed to interoperate fully with Java and includes a large number of libraries.

In the current context, the code was inherited from the previous version, with improvements implemented. For the sake of simplicity, it was chosen to adapt the new backend to the previous configuration of the application, avoiding the modification of previously well-functioning functionalities within the Android application.

### 3.3.1 Project Organization

The project inherited the following organization:

- **data:** contains files necessary for managing the local database and client-server communication;
- **repositories:** contains models for manipulating information in the local device database;
- **ui:** stands for "user interface" and contains everything needed for the proper functioning of the interface;
- **utils:** contains variables or methods globally referenced in the project.
- **res:** resource folder, where XML code for application design is located.

The project is built and executed using Gradle, a build automation tool and dependency management tool.

### 3.3.2 Fragment Application

The approach of structuring code into fragments and linking them through mapping is a common way to manage navigation between different screens (fragments or activities) in an Android application. This approach allows for a more modular and organized management of the application flow.

The `androidx.navigation.safeargs.kotlin` plugin simplifies navigation management. Here's a more detailed explanation of how it works with an example:

- **Creating the Navigation Graph**: Initially, a "navigation graph" representing the application flow is created. This graph contains all the destinations (screens) of the app and the connections between them.
- **Defining Destinations**: Each destination (screen) in the navigation graph is defined with a unique name, e.g., "StatementAddForestFragment". This name uniquely identifies the fragment or activity.
- **Defining Actions**: Actions in the navigation graph define transitions between destinations. For example, `actionStatementAddForestFragmentToStatementFragment()` is an action defining the transition from the "StatementAddForestFragment" to another destination called "StatementFragment".
- **Using Actions**: By using `androidx.navigation.safeargs.kotlin`, a specific action can be called to navigate to a particular destination. In the example, `StatementAddForestFragmentDirections.actionStatementAddForestFragmentToStatemen` represents the action to navigate from the "StatementAddForestFragment" to the "StatementFragment" destination.
- **Providing Arguments**: It's also possible to provide arguments through the action to pass data between destinations. These arguments are defined in the navigation graph and can be used to customize the behavior or data shown in the new destination.

Using this approach simplifies navigation within the application, making it clearer how different fragments or activities are related to each other and how they can communicate by sending data between them. Moreover, it helps keep the code cleaner, well-organized, and easily maintainable, providing crucial support during the initial project setup, enabling a quicker understanding of the functionalities. Figure 6 illustrates this approach.

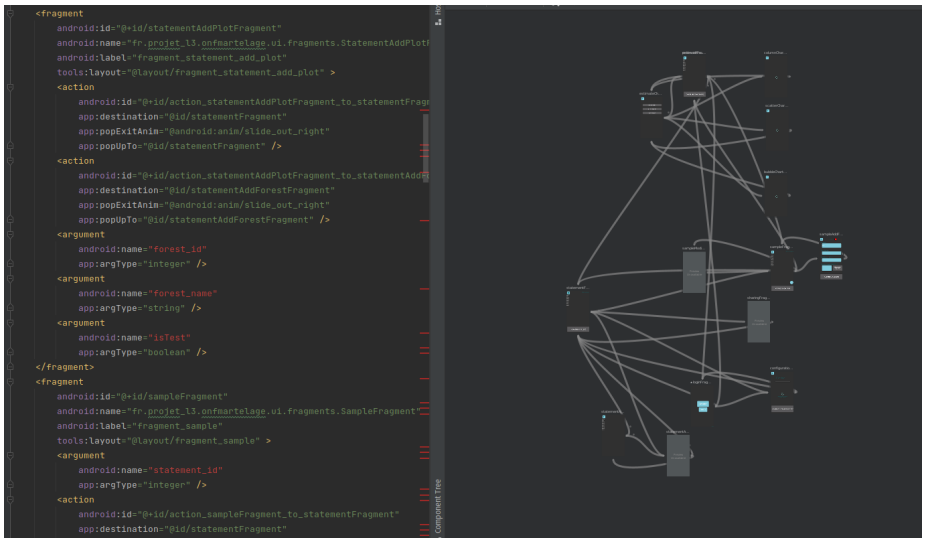



Figura 6: Code of a fragment alongside a graphical representation of the general mapping.

### 3.3.3 Natural Sorting

The elements in the list of lots within forests are of type `String`, causing incorrect sorting of numbers, for example, the three strings "1", "11", and "2", sorted as written, are not considered as integers. To solve this problem, a comparator was implemented that performs natural sorting algorithm, enabling the natural comparison of strings.

```
private fun naturalSortComparator(): java.util.Comparator<Plot>{
    return Comparator{plot1, plot2 ->
        val pattern = Pattern.compile("(\\d+)|(\\D+)")
        val matcher1 = pattern.matcher(plot1.ccod_prf)
        val matcher2 = pattern.matcher(plot2.ccod_prf)

        while (matcher1.find() && matcher2.find()) {
            val token1 = matcher1.group()
            val token2 = matcher2.group()

            if (token1.matches("\\d+".toRegex()) &&
            token2.matches("\\d+".toRegex())){
                val num1 = token1.toInt()
                val num2 = token2.toInt()
                if (num1 != num2) {
                    return @Comparator num1.compareTo(num2)
                }
            } else {
```

```
                if (token1 != token2) {
                    return @Comparator token1.compareTo(token2)
                }
            }
        }
        return @Comparator 0
    }
}
```

Explanation of the algorithm:

- Creation of a regex pattern for splitting the string into numeric (`\d+`) and non-numeric (`\D+`) groups.
- Creation of two variables, using the `match` method, to return the result of the regex pattern on the `ccod_prf` variable of the two Plot objects.
- Execution of the comparison based on natural sorting:
  - For each iteration:
    - Extraction of a token, for each Plot object, which can be a numeric or non-numeric character.
    - If both tokens are numeric, they are converted to integers and compared numerically.
    - If at least one of the tokens is non-numeric, a comparison based on the strings themselves is performed, also called lexicographic ordering.
  - At the end of the iteration, the comparator will return a value:
    - If the tokens are numeric and different, the result of the numeric comparison is returned.
    - If the tokens are non-numeric and different, the result of the lexicographic comparison is returned.
    - If the tokens are equal or the string is exhausted, 0 is returned.

The `@Comparator` annotation refers to the `CustomComparator` class and is used to indicate a custom comparator.

The worst-case scenario occurs when the strings plot1 and plot2 are similar and contain the maximum sequence of characters. The comparator analyzes the strings character by character, considering both numeric and alphabetical characters. In cases where there are numbers at the beginning of the strings, these will be converted to integers and compared. If the numbers are very large or differ only in the most significant digits, this may require a significant number of operations.

It can be concluded, therefore, that the complexity of the algorithm is $O(n \cdot m)$, where $n$ is the length of the input strings and $m$ is the maximum number of digits that can be present in a number within the strings.

### 3.3.4 Offline mode

In chapter 2.5.1, the difficulty of using the application by operators in forests where internet connection is weak or absent was discussed. It was identified that in the previous code, a token containing the necessary access information was saved, enabling potential access in the absence of a connection. However, upon further analysis, it was discovered that the token check was performed after checking for internet connection, which prevented access in the specified case. The immediate fix was to remove the internet connection check from the login activity.

In the existing application, there was already an implementation for local data storage within the database using the Room library, one of the most popular Kotlin libraries for internal management of SQLite databases. Room allows for the declarative definition of data models, DAOs (Data Access Objects), and the database, providing a higher level of abstraction compared to direct SQL usage. The reason why it was impossible to obtain the list of forests and plots, which are blocking for subsequent operations, was that the implemented database was not utilized for storing this information. Therefore, with every access to the activity, the application sent a request to the server to obtain the data.

As a solution to this problem, it was decided to move the requests inside the activity used to select the territorial unit. Upon confirmation, a complete download of every forest and associated plot for the selected unit is executed, blocking any other possible operation until the download is complete. The downloaded information is then saved inside the local database, which will be queried upon every access, thus not necessarily requiring internet access to retrieve the data. Additionally, the application checks for any additions or modifications of territories. When internet connection is available, the application sends a request to verify that the list has not been modified.

Another issue related to the connection was that the application crashed when attempting to send data to the server in its absence. It was discovered that this was simply due to the order in which the instructions were called. Upon encountering a problem, the error was handled in a `catch` block, interrupting the currently executing thread. However, the problem was that subsequent to this instruction, the thread itself was interrupted, which, already being interrupted, caused a fatal error. Simply moving the normal thread interruption before the error handling part resolved the conflict. Additionally, a notification indicating the ongoing submission of readings was created to confirm to the user that the application is running in the background. In Kotlin, to perform an operation, even after the application is closed, the `GlobalScope.launch` command can be used, which starts a coroutine thread to asynchronously handle a particular operation.

### 3.3.5 Room and the local database

As mentioned in the previous chapter, the Room library supports the management of local databases within devices. Its methodology utilizes annotations for

easier integration of SQL code, used within DAO (Data Access Object) objects, declared as `interfaces`.

One of the main advantages of Room is the mandatory creation of a new version of the database with each structural change. The library, to keep track of the updates made, forces the developer to create a new version where additions, modifications, or deletions of tables are included. Essential is the maintenance of the integrity of previous versions in this process of structural evolution of the database.

DAOs can return the desired information using data classes as containers. This approach is motivated by the structural similarity between data classes and database tables. Data classes provide a clear representation of domain entities, and the variables within them can be associated with the columns of the tables, offering an orderly way to organize and manipulate data.

For communication with the database, repository classes are employed, which are normal classes containing methods aimed at invoking DAO classes. These represent a significant abstraction between the data source and the main body of the application, establishing a practical connection, allowing for structural changes to the database without requiring changes to the primary code logic.

### 3.3.6 Douglas bug solution

For the calculation of the tariff dedicated to the Douglas fir, after the insertion of 20 specimens, an external access to the list was performed, creating the `IndexOutOfBoundsException` error. Additionally, the average was miscalculated by working directly on integers and, in some cases, resulting in divisions by zero. The code was updated as follows:

```
private fun douglasGetSublist(douglasList:List<Sample>):
List<Sample> {
    val size = douglasList.size
    return if (size > DOUGLAS_COUNT_THRESHOLD) {
        val subListSize = (2.0 * size / 3.0).toInt()
        douglasList.sortedByDescending {it.height}
        .subList(0, subListSize)
    } else {
        douglasList
    }
    val sum = douglasList.sumOf { it.height }
    hauteurDominanteRecommended = sum / douglasList.size
}
```

The variable `DOUGLAS_COUNT_THRESHOLD` has been assigned the value 20, which is a threshold value established by the calculations performed internally by the organization. This method returns a list of Douglas tree samples that will be used to determine the dominant height. If the number of samples exceeds the threshold, it calculates the size of the sublist that needs to be extracted. The

size of the sublist is 66% (2/3) of the total size of the Douglas tree samples. Finally, it calculates the average based on the received sublist.

## 3.4 PostgreSQL and the database

PostgreSQL is a full-featured, open-source object-relational database management system (DBMS) that uses the SQL language. For easier use, the pgAdmin4 graphical interface was employed, specifically developed to make data manipulation more intuitive, fast, and direct, thus avoiding the need to write queries for basic commands. The server on which queries could be executed was provided towards the end of the project, due to unforeseen circumstances. To work on it even without its presence, PostgreSQL provides the option to simulate a server locally on the computer, at the localhost address and a port number not occupied by other processes, thus allowing a website to execute queries on the local database.

In the current project, the request for modifying the database did not significantly impact the basic structure. The following information was added:

- indicating whether it is a test;
- the year, filled in by the user, in which the survey was conducted;
- the ability to make a species visible or invisible in the tree species list.

During the migration of the database from MySQL to PostgreSQL, some aspects were considered to improve efficiency and readability:

- returning only the necessary data;
- using JOINs instead of subqueries;
- for greater code uniformity, queries were written to use only LEFT JOINs.

In figure 7, the UML diagram of the current situation is shown.

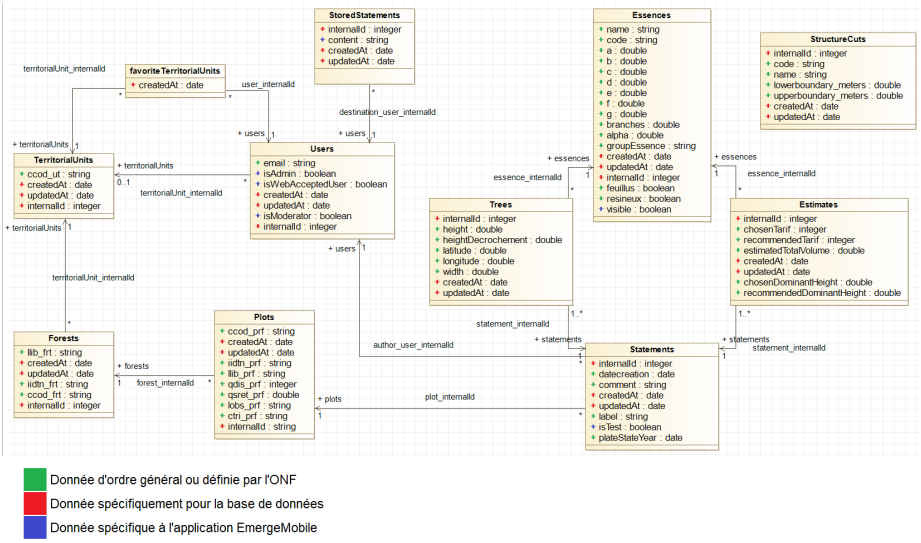



Figura 7: UML diagram of EmergeMobile database version 2.

## 3.5 Other tools

### 3.5.1 Integrated Development Environment (IDE)

The development environments used were Visual Studio Code, IntelliJ IDEA, and Android Studio. The choice of IDE for development is of fundamental importance because they can offer various features and plugins that can be important in terms of time. All three offer a wide availability of extensions and excellent compatibility with code sharing systems, such as GitHub, which will be discussed in chapter 3.5.2.

- **Visual Studio Code** is free, lightweight, and open-source. It offers a wide range of extensions and supports many programming languages. Advanced code autocompletion features based on artificial intelligence make coding more comfortable. It has proven extremely useful for writing React code.
- **IntelliJ IDEA** finds full compatibility with SpringBoot. It features a highly advanced editor that offers functionalities such as intelligent refactoring, code analysis, quick navigation, smart suggestions, code autocompletion, and support for modern language standards. With this IDE, project management is extremely simple, especially in the current context using Maven, allowing for intuitive and immediate execution and stopping.
- **Android Studio** is the official open-source development environment for Android applications. It has an extremely intuitive user interface, allowing design construction through both written code and a graphical interface. It also offers emulators to run applications in the absence of an available

physical device. It uses Gradle as the project software manager and has excellent compatibility with the programming language used, Kotlin. A very powerful feature is the ability to convert code from Kotlin to Java and vice versa.

Android Studio was developed from IntelliJ IDEA. Previously, Android developers used the latter with plug-in support for Android development, until Google decided to create a separate, dedicated IDE for Android development without changing the user experience.

### 3.5.2 Git as Team Support

Code sharing platforms like GitHub and GitLab are crucial in team development, as they allow well-organized code sharing. In this work, the internal GitLab of the organization was used.

There are many ways to organize code within these platforms; in this specific case, the choice was made to use a main branch, called "develop," which must contain code that compiles without errors, and for each feature, create a new branch to integrate with the main one at the end.

This approach proved to be of fundamental importance during the project's creation because the client could follow the development step by step, providing transparency on the progress of the development that allowed for timely feedback from the client, contributing to directing the project in a targeted manner.

Since the new configuration was not immediately available, a strategy was adopted that allowed for the improvement of the application in the current system and, in parallel, the adaptation of the application to the new one, so that it could be applied at the time of migration.

Git offers the advantage of being able to compare any changes, allowing for the maintenance of the history of operations performed. The disadvantage in the difficulty of deleting the history is when code with passwords present is mistakenly uploaded; this would create a significant security issue, often requiring the deletion of the project and its recreation.

### 3.5.3 Login

In the initial login screen, a button has been added to display the protocol, as described in Chapter 2.4.1, to simplify the work for new foresters. For authentication, the application uses the Azure Active Directory (Azure AD) service, connecting to the organization's Microsoft service, accessible only with a corporate account.

### 3.5.4 Home

Within the main page, after logging in, there is a list of ongoing surveys. After the update, it is possible to choose whether to carry out a survey that will

be considered valid or just for testing purposes. The latter is used to simulate potential issues or test features without entering unreliable data into the database.

Test surveys are marked with a warning message indicating their unreliability, preventing ambiguity.

Viewing the entered data is done by simply clicking on the desired sampling. Displaying operations such as sending to another operator, adding a label, or deleting require a long press. To avoid multiple press errors, the long press methodology was chosen even for the permanent delete command.

### 3.5.5 Territorial unit configuration

As addressed in Chapter 2.5.1, it often happened that in the absence of a connection, the list of forests could not be displayed. On the territorial unit selection page, which must be confirmed for correctness first, upon confirmation, there is a wait for the download of all the relevant forests and lots, which will be saved locally for future viewing even in the absence of a connection (Figure **??**).

At the end of the operation, the user is assisted in restarting the application to ensure immediate proper functioning. By confirming the prompt, the application will automatically restart.

This operation, ideally, is to be performed when assigning or changing administrative control of a territory, usually done in the office, where there is a good internet connection.

### 3.5.6 Forest Selection

Within the forest selection page, necessary for survey creation, a button has been added to search by name, simplifying the search operation. Additionally, the forest code information has been added to discriminate those with the same name (Figure **??**), as emerged in Chapter 2.5.1. After selecting the forest, a similar list with the corresponding lots appears but without the search button since they are contained in a few quantities.

### 3.5.7 Entering Tree Measurements

For tree entry, it is advisable to activate GPS as location data is useful, although it is not a blocking operation for the continuation.

During the measurement entry, a button has been added that proposes a decline height to simplify calculations for operators in the absence of an evident trunk decline. The starting point of the invisible decline is 90

This simple functionality has been added to simplify the work for foresters, often in unfavorable conditions for correct calculations, thereby reducing the likelihood of making an error.

With the update, a minimum and maximum height and diameter limit have been set, a range established to avoid accidental errors, adapting it to the calculation of the Algan scale analysis, as in Figure 4.

Upon completing the operation, the tree is added to the list. The information is also kept visible in the preview to identify any compilation errors (Figure **??**).

### 3.5.8 Tariff Selection

The tariff selection is the last operation to be performed before being able to send the survey.

As shown in Figure **??**, there is a list of collected tree samples divided by species, with the number of measurements taken and a proposed tariff, modifiable under the "retenu / retenue" (retained) entry.

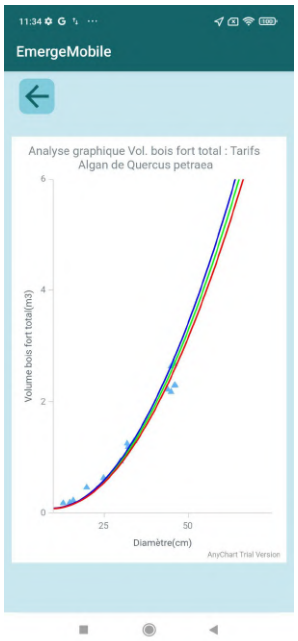
This operation is very sensitive because it involves statistical and economic analyses; generally, the modification of the proposed tariff is made by more experienced foresters.

### 3.5.9 Final Confirmation

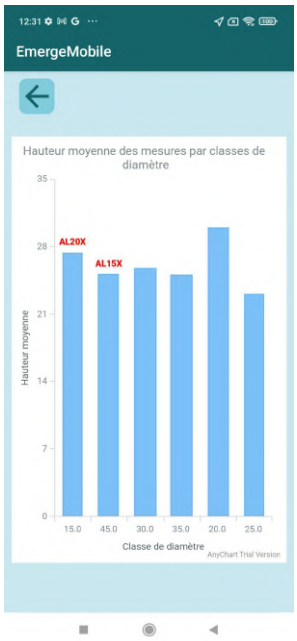
Prior to the final submission, after confirming the tariff selection, the user is asked to enter an optional comment and the year of survey entry. The latter parameter has been added to allow the insertion of data from past years through the application while maintaining the full reliability of the information.

### 3.5.10 Statistics

The Android application also allows the visualization of statistics related to the collected and not yet sent data, visible in Figure 8. This function has not been modified in the present work.




(a) Graphical analysis of wood volume based on the Algan scale.




(b) Height analysis classified by diameter.

Figura 8: Statistics viewable in the application.

## 3.6 EmergeMobileWeb

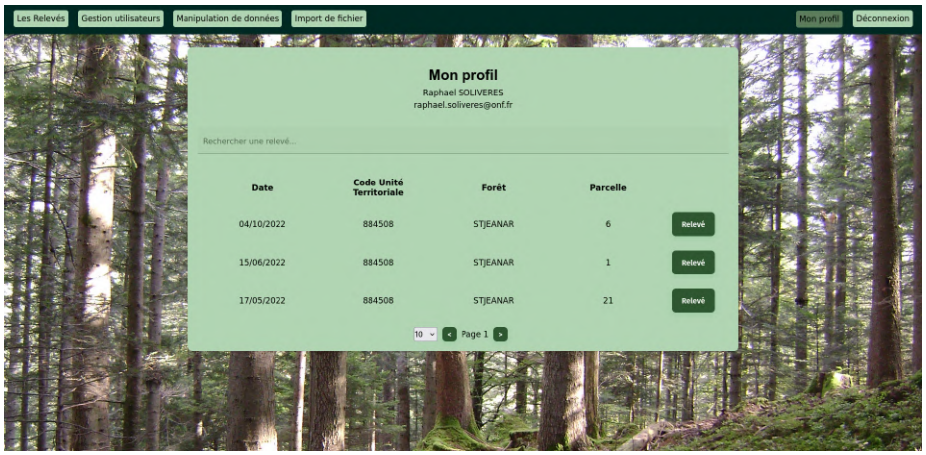
EmergeMobileWeb is the interface used to extract and manipulate data collected through the Android application. It is accessible only through an internal connection within the organization.

The web interface has been rebuilt from scratch, taking inspiration from the structure of the previous version to avoid forcing users to completely change their habits.

### 3.6.1 Home

In Figure 9, the change in the main screen is shown. Compared to the previous version, it was decided to show, after logging in, the list of user's surveys and move the navigation buttons inside the top header. This method has been established to provide a more direct global overview of the user's profile, showing main information and surveys, sorted in descending order by insertion date.


(a) Previous interface of the home page.




(b) New interface of the home page.

Figura 9: Graphical modification of the home page.

### 3.6.2 Data Manipulation

In Chapter 2.5.1, it was mentioned that an administrator, to make modifications, had to resort directly to the database, and not having the necessary IT skills could lead to inconsistencies, potentially causing system crashes. The update allows a user with admin permissions to modify certain parameters directly through the web interface, without having to access the data store directly. Figure 10 shows the selection of tables containing the information that a manager wants to modify, accessing a tabular formatting shown in Figure 11.

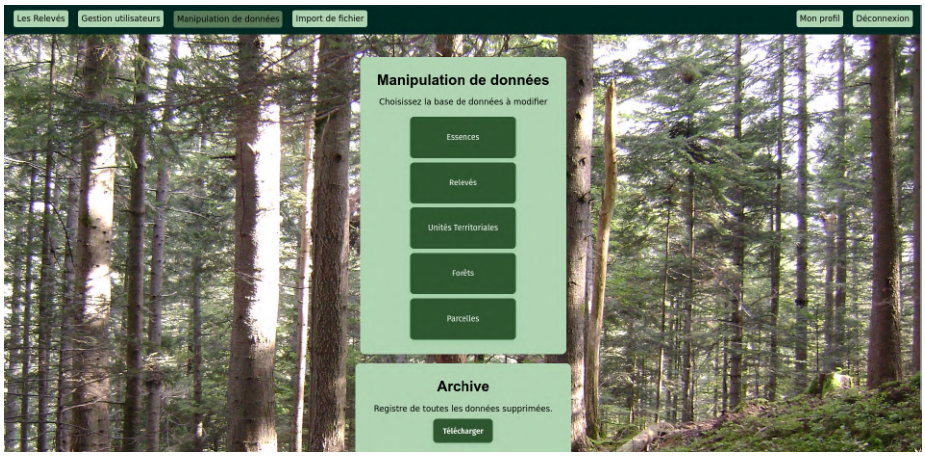



Figura 10: Data management page including editable tables and the possibility to download stored data.

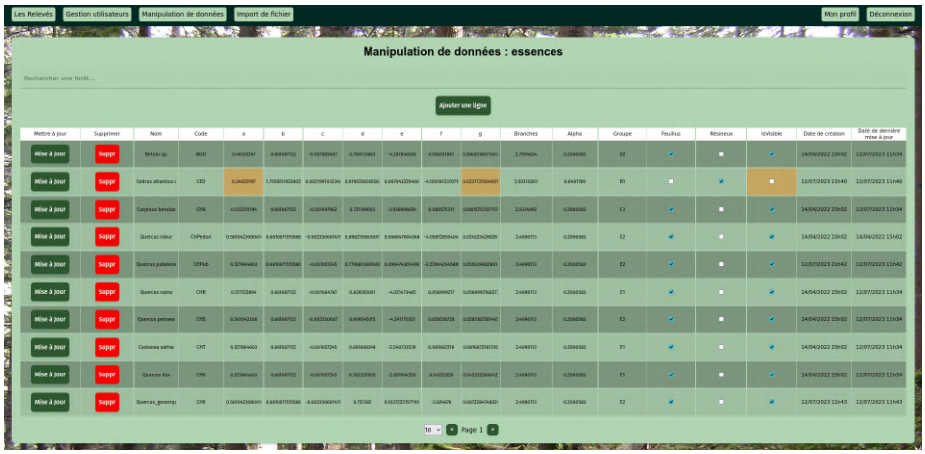


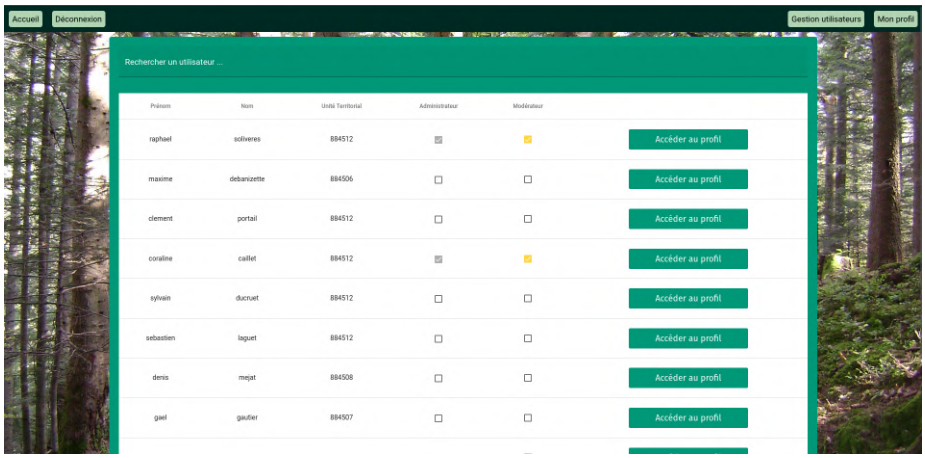

Figura 11: Table "essences" with editable parameters.

Understanding the modified parameters is intuitive, where the relevant cell changes color until confirmation through the save command. Modification is possible individually on each line, thus maintaining greater control over the changes made. The visualization of cells with a large number of characters is also facilitated by widening the input field of the selected string.

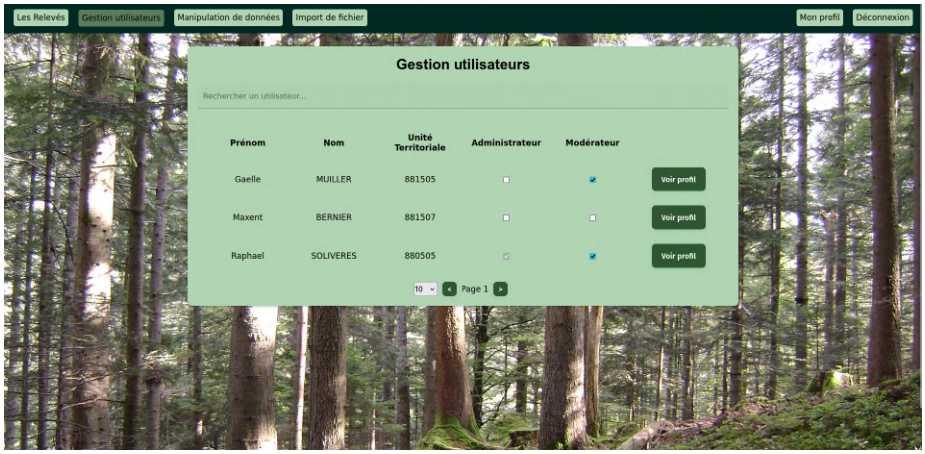
Visible at the bottom of Figure 10, the new feature of archiving deleted surveys has been added, downloadable through the present button, allowing for a history and the ability to restore any mistakes.

### 3.6.3 User Management

User management, visible only to administrators, has been rebuilt exactly based on the previous design, thus retaining every functionality but adapting the design.




(a) Previous interface of user management.



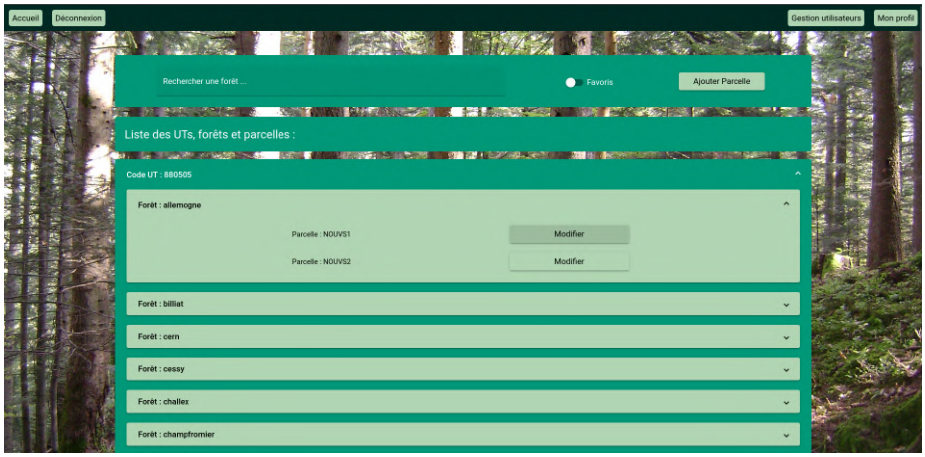

(b) New interface of user management.

Figura 12: Graphical modification of the user management page.

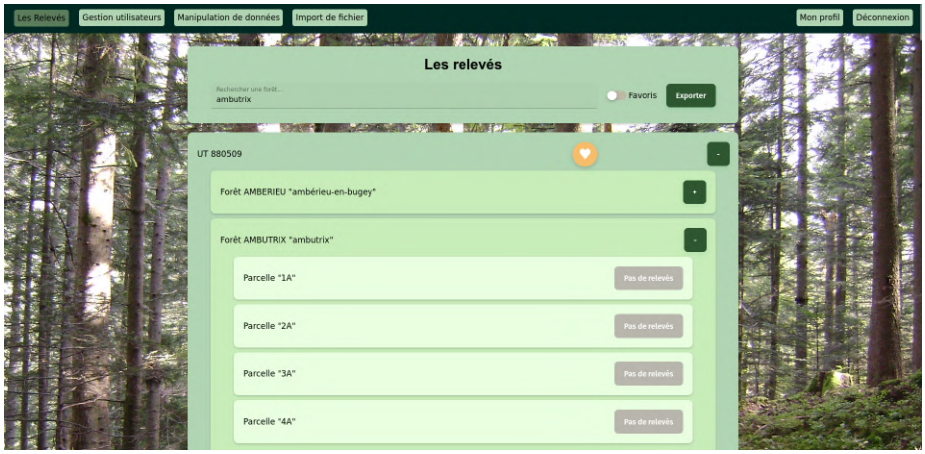
### 3.6.4 Surveys

Compared to the native version, it was decided to merge the information from the three pages: "Mes Parcelles" (My Plots), "Mes Forêts Favorites" (My Favorite Forests), and "Mes Relevés" (My Surveys) into one, illustrated in Figure 13, misleadingly named "Les Relevés" (the Surveys) to avoid misunderstan-

dings. Within this page, it is possible to add or remove territorial units from favorites, filter them, get the list of territories, and view their global surveys that have been uploaded.



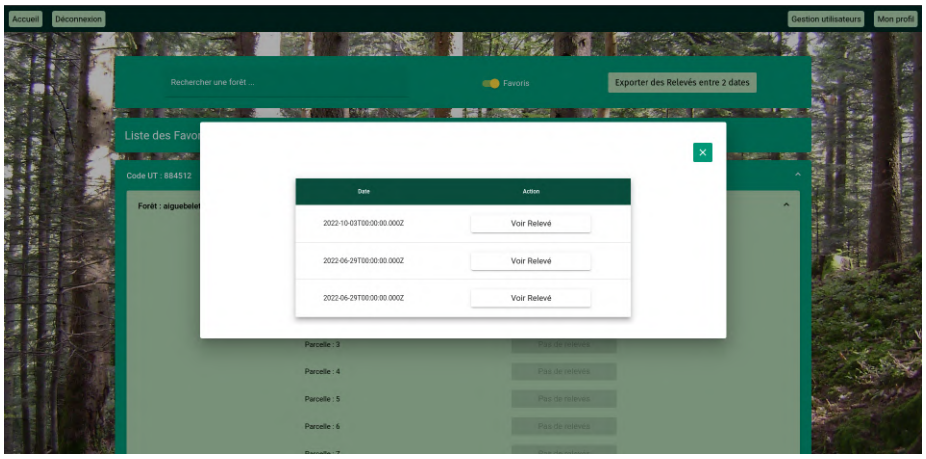

(a) Previous interface of the territories list.




(b) New interface of the territories list.

Figura 13: Graphical modification of the territories page.

In plots with surveys, their visualization has been made more intuitive by adding the collector's name, changing the date format, and improving the design with more attractive and visible colors. The update is visible in Figure 14.

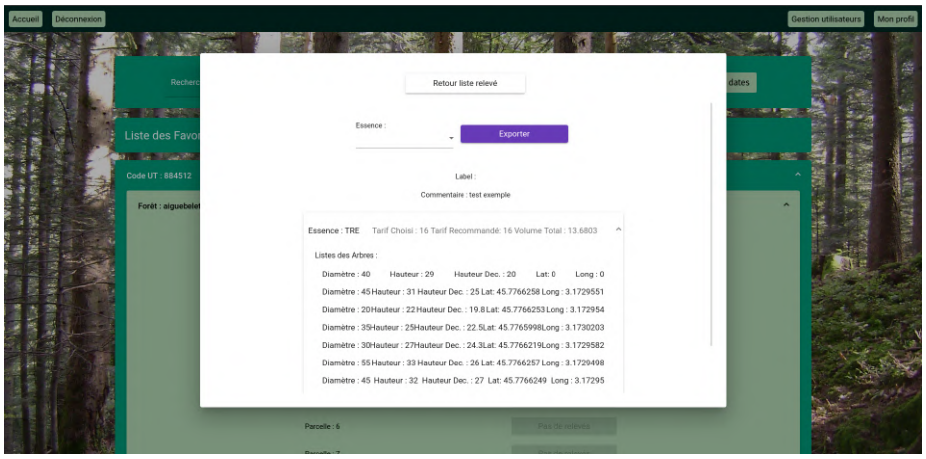



(a) Previous list of surveys for a plot.

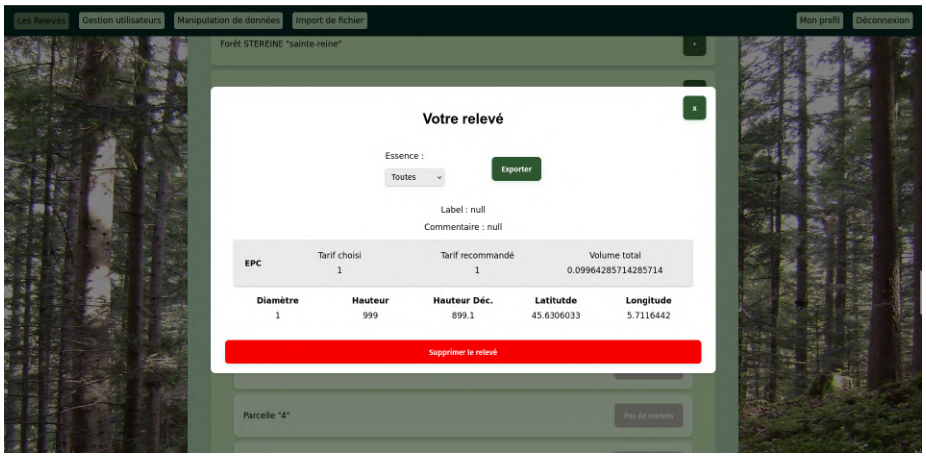

(b) New list of surveys for a plot.

Figura 14: Modification of the surveys list.

Opening a survey follows the same previous format but with the addition of the delete button, visible only to administrators (Figure 15).




(a) Previous interface of a survey.

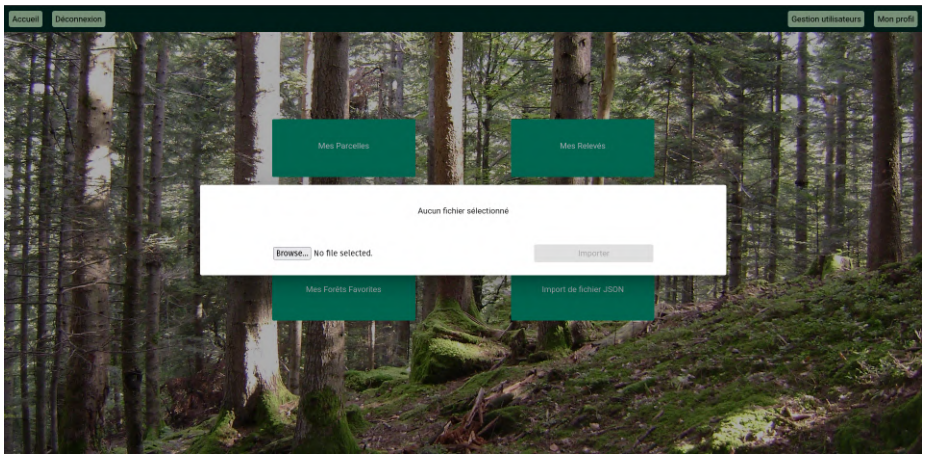



(b) New interface of a survey.

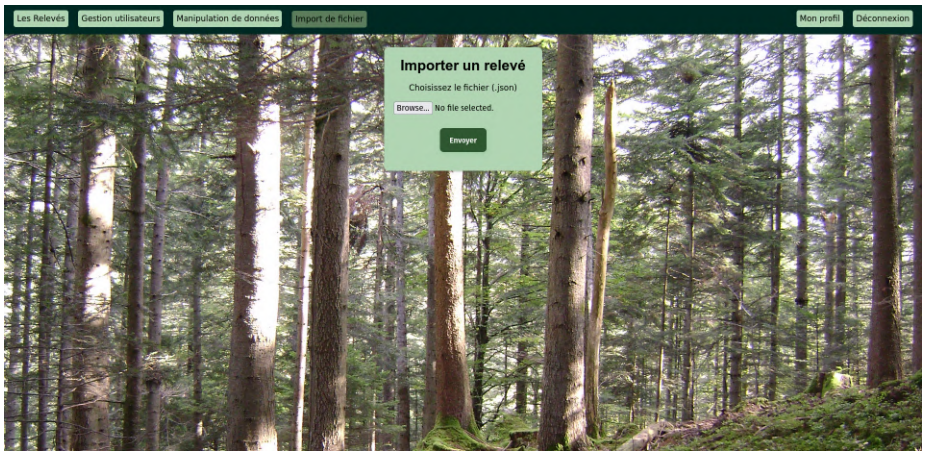
Figura 15: Graphical modification of the survey page.

### 3.6.5 Importing Surveys in JSON Format

Although the import format has been kept the same, as seen in Figure 16, additional information has been added, such as the required file format to upload. Confirmation messages for the operation have also been added (Figure 17). In the work, the importance of feedback for smoother usability has been frequently discussed.

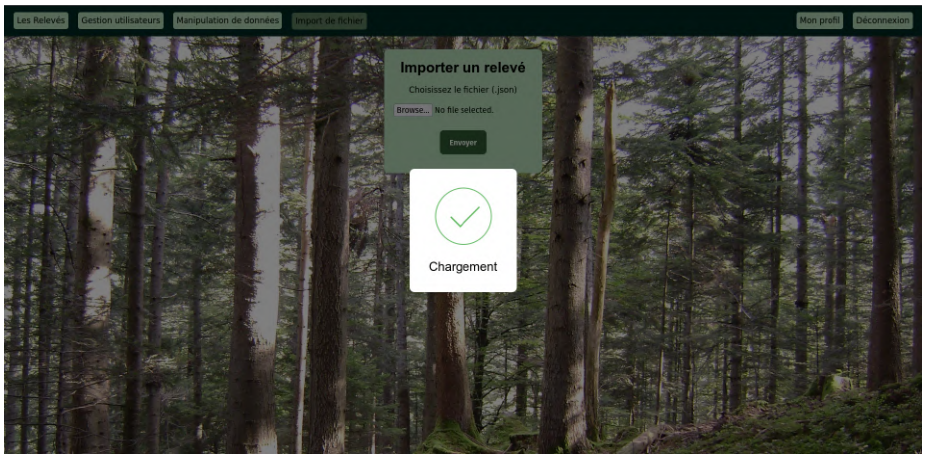



(a) Previous interface of JSON file import.

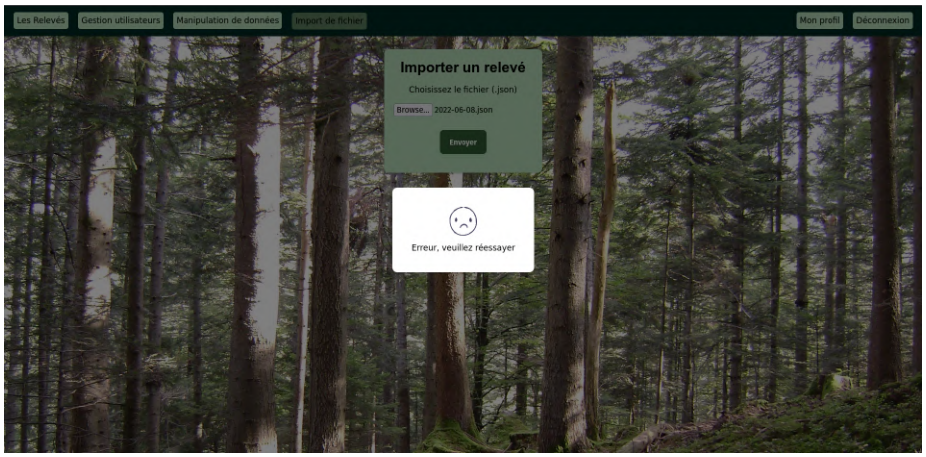



(b) New interface of JSON file import.

Figura 16: Graphical modification of the JSON file import page.




(a) Import success notification.

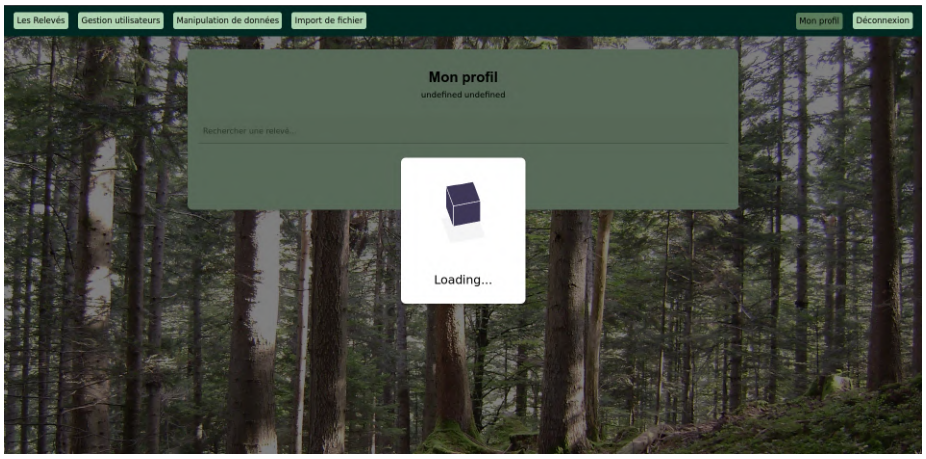



(b) Import failure notification.

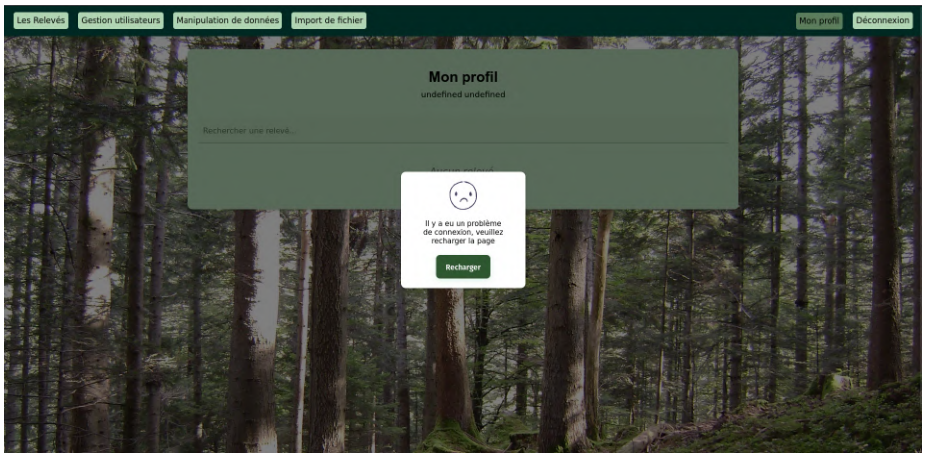
Figura 17: Import file outcome notifications.

### 3.6.6 Upload Feedback

Communication with the user about what is happening is crucial. In some cases, the connection may take longer than expected to load information, in others, there may be a large amount of data to load, or there may be an absence of an internet connection. The user should never have the impression that something is not working within the application. In Figure 18, the loading screen is shown, which, following the connection attempts discussed in Chapter 3.1.8, displays a warning notifying the existence of a problem and prompts the user to reload the page.




(a) Loading screen.




(b) Failed connection screen.

Figura 18: Transition from loading to connection failure.

## 3.7 AI Support

There has been much talk recently about an artificial intelligence tool made publicly available, ChatGPT. This tool finds significant use in the computer science sector and has been a fundamental support for the realization of this project. In this work, it was queried via the web interface, not only for code provision but also for conducting other types of research. Although Google searches have proven to be a valuable resource for finding information, compared to ChatGPT, it has been demonstrated that they can take up to 65% more time to reach the same conclusion [7]. For this reason, the methodology adopted was to make an initial attempt using this powerful support tool and, in case

the response was not adequate, subsequently resort to traditional methods of inquiry.

The accuracy of the conclusions provided has also been discussed, often containing inaccuracies. With the introduction of COVID-19 vaccines, considering the large number of news articles discussing conspiracy theories, ChatGPT was able to attribute any lack of reliance on the information found [5]. For this reason, I believe the data received should be verified but can still be considered a good research base.

Its code-writing function is very powerful. In the following example, we will see the response (Figure 19) to the request: "In React, create a central blue button that pulses on each click and reloads the page when pressed. Additionally, place a title above the button with the text 'Welcome' and set the background color of the page to yellow. Provide the CSS code as well," and its direct application (Figure 20).


Figura 19: ChatGPT response.

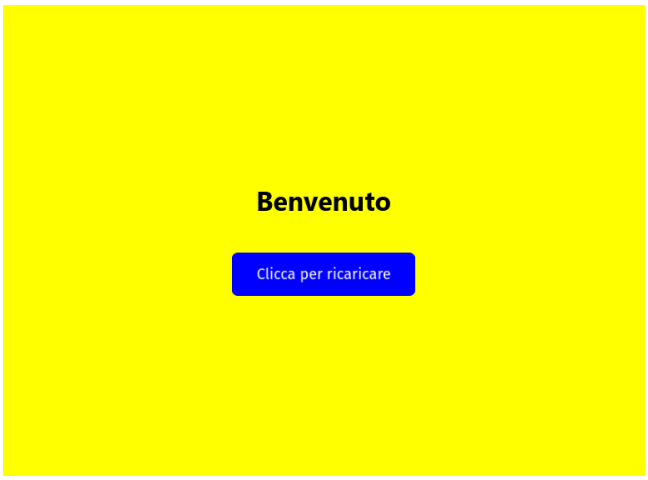



Figura 20: Result of the code provided by ChatGPT.

The interpretation of bugs played a fundamental role in shortening development times. Before its latest update, a study compared it with other AI systems and bug-solving methodologies, such as Codex and CoCoNut, as well as standard APR approaches. Out of forty bugs proposed to solve, although CoCoNut was able to solve twenty-one, ChatGPT and Codex were able to solve nineteen, still confirming their effectiveness. Conversely, standard APR methods only managed to solve seven [6].

Using ChatGPT for code creation can, however, carry risks of error and inefficiency. Considering that the data in ChatGPT's database, acquired during the training period, is updated until September 2021 and has not received updates beyond that, responses often contain obsolete functions, requiring the developer to search the internet to find updated equivalents. In some cases, it may provide inefficient code or code containing bugs, and debugging may take longer than designing the method itself, resulting in much development time for a non-optimal function. Furthermore, while its use can speed up learning, excessive reliance on it may instead create a sort of "dependency" for the programmer, limiting the improvement of their programming and design skills.

Significant slowdowns were noted in conversations with a high volume of requests, leading to a significant delay in response times. This situation made such conversations inefficient and necessitated the creation of new conversations, causing the loss of all previously provided information. It is relevant to underline that ChatGPT leverages previous responses to develop a deeper understanding of context and consequently improve future responses.

# 4 Conclusion

The thesis demonstrates the excellent compatibility among the technologies used, which, in the present context, have proven effective for the ecology of the planet. EmergeMobile, which is a system with medium complexity of implementation, has an important impact on reality, simplifying the work of foresters in the French territory and increasing the efficiency of time management.

ChatGPT's artificial intelligence has expedited the learning, development, and debugging process, proving to be an excellent support for programmers. Although in many cases the responses did not precisely meet the posed need, they still provided excellent experimentation and research insights.

A good practice would be to exploit the advantages of SpringBoot more, communicating with the server through dedicated interfaces, leaving SQL requests written in the current version for operations that require more customization, such as JSON file import. The management of the XLSX file for storing deleted surveys also needs improvement, as currently, by never deleting the data within it, the file size may grow to significant proportions on the server's local disk, necessitating manual deletion.

# Riferimenti bibliografici


[1] Pauline COUTADEUR. Etude du potentiel indicateur des micro-habitats des arbres: lien avec la biodiversité forestière et test d'effet observateur. 2010.

[2] Office National des Forêts. Les forêts de nos territoires. 2023.

[3] Albert Maillet. Emerge dendrométrie. *RenDez-Vous techniques*, 2014.

[4] Meta. Panoramica sugli hooks.

[5] Malik Sallam, Nesreen A Salim, B Ala'a, Muna Barakat, Diaa Fayyad, Souheil Hallit, Harapan Harapan, Rabih Hallit, Azmi Mahafzah, and B Ala'a. Chatgpt output regarding compulsory vaccination and covid-19 vaccine conspiracy: A descriptive study at the outset of a paradigm shift in online search for information. *Cureus*, 15(2), 2023.

[6] Dominik Sobania, Martin Briesch, Carol Hanna, and Justyna Petke. An analysis of the automatic bug fixing performance of chatgpt. 2023.

[7] Ruiyun Xu, Yue Feng, and Hailiang Chen. Chatgpt vs. google: A comparative study of search performance and user experience. *arXiv preprint arXiv:2307.01135*, 2023.

[8] Ernst Zürcher. La forêt et le bois: des caractéristiques et des propriétés exceptionnelles face à l'effet de serre – quelques arguments décisifs. *Schweizerische Zeitschrift fur Forstwesen*, 157(11):519–522, 2006.