

DELivErable 1

Group 2



SCHNEPP Anthony | RAJOELISOA Enorian | LEY Louise | HOOG Cédric

13 mai 2025

Prosoft

Table of contents

[**I.** **Context** 2](#_Toc198065931)

[**II.** **Introduction** 2](#_Toc198065932)

[**III.** **UML Diagrams** 3](#_Toc198065933)

[a. What are UML Diagrams 3](#_Toc198065934)

[b. Use Case Diagram 3](#_Toc198065935)

[c. Activity Diagram 4](#_Toc198065937)

[d. Sequence Diagram 6](#_Toc198065938)

[e. Class Diagram 7](#_Toc198065939)

[**IV.** **Conclusion** 8](#_Toc198065940)

# **Context**

The EasySave project aims to develop a comprehensive backup software solution. This project includes the creation of the software, ongoing version management (major and minor updates), and complete documentation for both end-users and technical support teams.

All code and documentation must be managed using the prescribed tools to ensure consistency and maintainability for future teams.

This approach ensures a structured, high-quality software release, aligning with ProSoft's standards for professional software solutions.

# **Introduction**

This deliverable will contain all the different diagrams that we did for the conception of the version 1.0 of the software.

For a quick overview of the software, here’s the different demands for the first version:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fonction** | **Version 1.0** | **Version 1.1** | **Version 2.0** | **Version 3.0** |
| Graphical user interface | Console | Console | WPF | WPF |
| Multi-language | English and French | English and French | English and French | English and French |
| Backup jobs | Limited to 5 | Limited to 5 | Unlimited |  |
| Daily log file | Yes in JSON only | Yes ( JSON , XML) | Yes ( JSON , XML)  (Additional information on encryption time) | Yes ( JSON , XML) |
| User can pause one or more jobs | No | No | No | Yes |
| Status file | Yes | Yes | Yes | Yes |
| Type of backup operation | Concurrent or sequential | Concurrent or sequential | Concurrent or sequential | Parallel |
| Stop if a software package detected | No | No | Yes (impossible start another job) | Yes (all current transfers are stoped) |
| Use of external CryptoSoft encryption software | No | No | Yes | Yes |
| Priority file management | No | No | No | Yes, with other tasks in queue |
| Simultaneous backups prohobited for large files | No | No | No | Yes |
| Remote display interface | No | No | No | Yes |
| Single-instance application | No | No | No | Yes |
| Monitoring of network load | No | No | No | Automatic reduction of flows |

The first version is a Console application that will allow a user to specify 5 different backup jobs, manage them and run them. It will also have 2 available languages (English and French). To finish, it will also log each job in a daily log file in JSON format and a status log file.

# **UML Diagrams**

## What are UML Diagrams

UML which signifies “Unified Modelling Language” is a standardized modelling language consisting of an integrated set of diagrams.

We call UML Diagrams the different diagrams that are used to visually represent the structure, behavior and interactions within a software system.

These diagrams provide a universal way to describe and document software architecture and design.

There are 2 subcategories of UML diagrams: structural and behavioral diagrams.

* The **structural diagrams** visualize the different components that make up a system and the relationship between them, basically showing the static aspects of a system.
* On the other hand, the **behavioral diagrams** represent what happens in a system, including how the components interacts between each other and with other systems.

To conclude this part, UML Diagrams help simplifying abstract and/or complex ideas by visualizing it, it also helps keeping the developers on the same page and collaborating.

## Use Case Diagram

The Use Case Diagram is a behavioral UML Diagram, the objective of this diagram is to represent all the different possible interactions between the user and the software. It helps visualize all the functionalities depending on the actor.

For our first version of the software, the user should be able to :

* Manage backup jobs by specifying the different parameters like the source directory and the target directory, the name of the job, the type (FULL to backup all of the files even if no changes has been made since the last backup or DIFFERENTIAL to backup only the modified files since the last backup job), … Managing backup jobs includes the deletion of a backup job
* Run specific jobs in sequential order and for each file in each job, add the logs to daily and status log files
* Modify application configuration, which includes modifying the language and the daily/status log paths

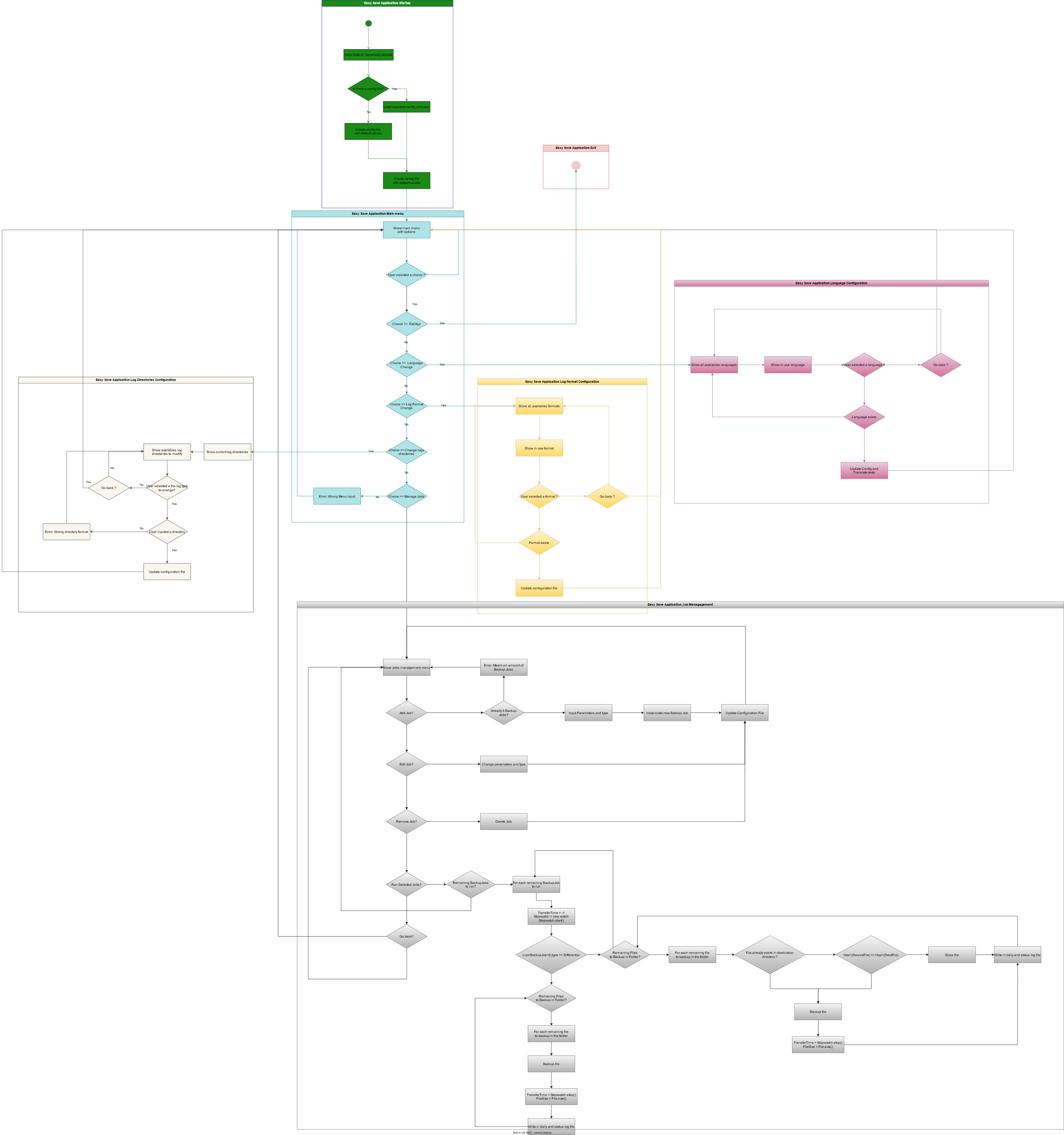
We simply just represented it in an easy way, here’s how our diagram looks (also available in the GitHub in our branch “Diagrams” under the folder Version1):

## 

## Activity Diagram

The Activity Diagram is a behavioral UML Diagram, the objective of this diagram is to represent the software and the different actions by showing the flow. It depicts the sequence of actions, decision points, and concurrent activities. It is basically the use case diagram but showing the flow of what is happening inside the application when the user interacts.

This is our Activity Diagram for the 1st version of the software:



If the image is not readable enough, here is the SVG file containing the diagram, also available in GitHub in our branch “Diagrams” under the folder Version1 :



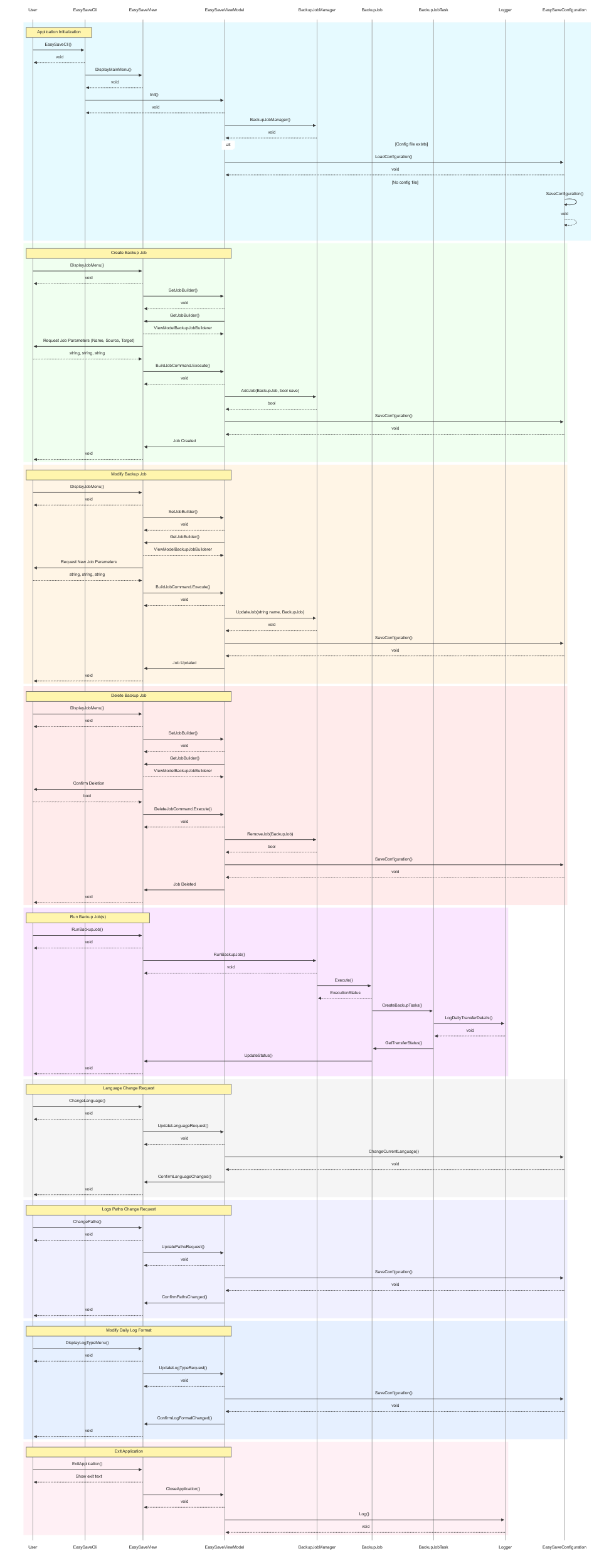
This diagram shows the different things that will happen when a user interacts with the app with what is offered to him. For example, it will describe how the Language Change will work in the software.

## Sequence Diagram

The Sequence Diagram is a behavioral UML Diagram that focuses on representing the flow of actions within a software system. It captures the sequence of tasks, decision points, and parallel processes that occur as users interact with the application. Unlike use case diagrams, which emphasize external interactions, activity diagrams provide a more detailed view of the internal processes, making them useful for visualizing complex workflows and business processes.

To illustrate this, we mapped out the various actions a user can perform and captured the corresponding interactions within the software through a sequence of method calls.

Here is our sequence diagram that is composed of the different tasks that the user can do:



If it is not readable enough, here is the mermaid link to the diagram, also available in GitHub in our branch “Diagrams” under the folder Version1 :

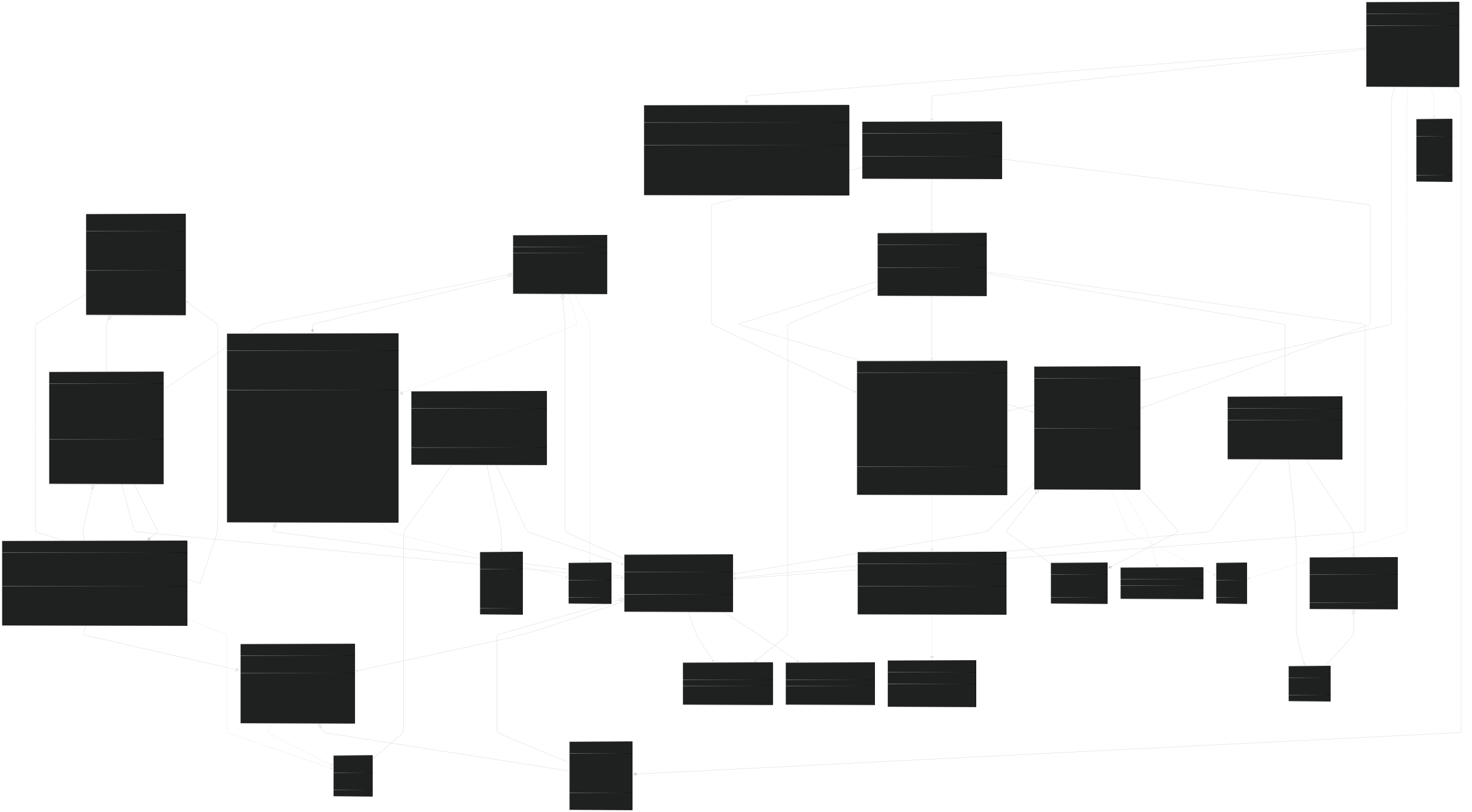


## Class Diagram

The Class Diagram is a structural UML diagram that focuses on representing the static architecture of a software system. It captures the key components, including classes, their attributes, methods, and the relationships between them. Unlike sequence diagrams, which illustrate the dynamic behavior of the system, class diagrams provide a high-level view of the system's structure, making them essential for understanding data models and the organization of code.

To achieve this, we identified the main entities within the application, defined their attributes and methods, and mapped their associations to illustrate how they interact within the system.

Here is our Class Diagram:

****

If it is not visible or readable enough, here is the SVG file of the diagram, also available in GitHub in our branch “Diagrams” under the folder Version1 :

****

# **Conclusion**

In this deliverable, we managed to represent the software through 4 different diagrams that allowed us to have a more global overview of the project and better understanding of the different tasks. Doing all of these diagrams made us able to identify the crucial interactions, anticipate potential challenges in the upcoming versions and ensuring a more robust and maintainable software design.