System Programming Project - Deliverable 3

EasySave version 3. 0

24/02/2022

CESI Engineering School

Saint-Nazaire

Anne-Laure GAUDON : Pedagogical tutor

FISA A3 Computer Science - Group 2

COUTEAUX Corentin

DEMBELE Romaric

RAMBIER Ewen

Method of dissemination:
Free

TABLE OF CONTENTS

INTRODUCTION

The advent of digital technology in our societies has revolutionized the way of working. Data is increasingly backed up thanks to software that offers this functionality. It is in this context that EasySave, a software for backing up data that is in the form of computer files, is placed. The realization of this application requires prior software modeling, a number of tools and computer technologies. Also, a user manual of the software must allow an easier and more efficient use of the latter.

I. UML DIAGRAMS

To get a visual overview of software and its features from the perspective of a developer and a non-developer, a number of standardized diagrams exist. The Unified Modeling Language (UML) is used to create these diagrams. For our software, a presentation of 4 of these diagrams will be made.

I.1 Use case diagram

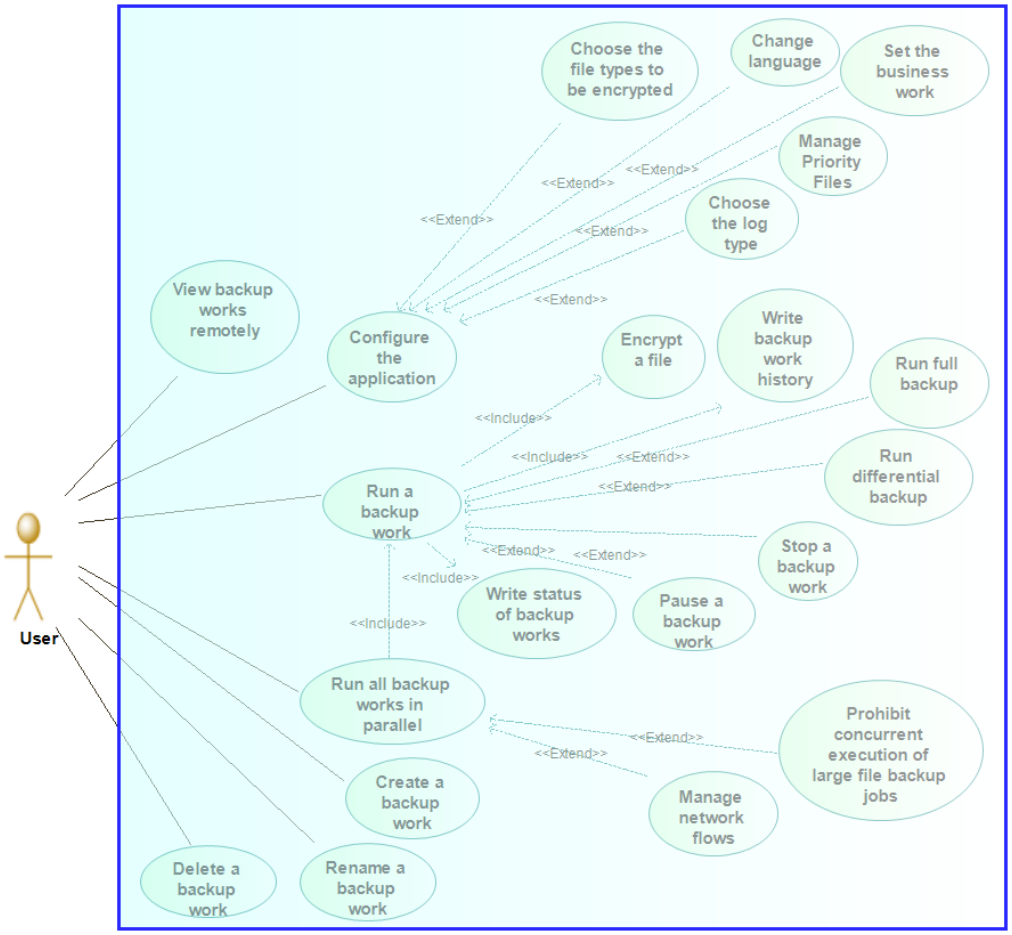


Figure 1: Use Case Diagram

The use case diagram provides an overview of the different features of the software. It is a diagram that can be presented to project actors who do not necessarily have a technical background.

Thus, we can see in the image above the main features of the software namely: running a backup job, sequentially running all backup jobs, creating, renaming, and deleting a backup job as well as configuring the software.

I.2 Activity diagram

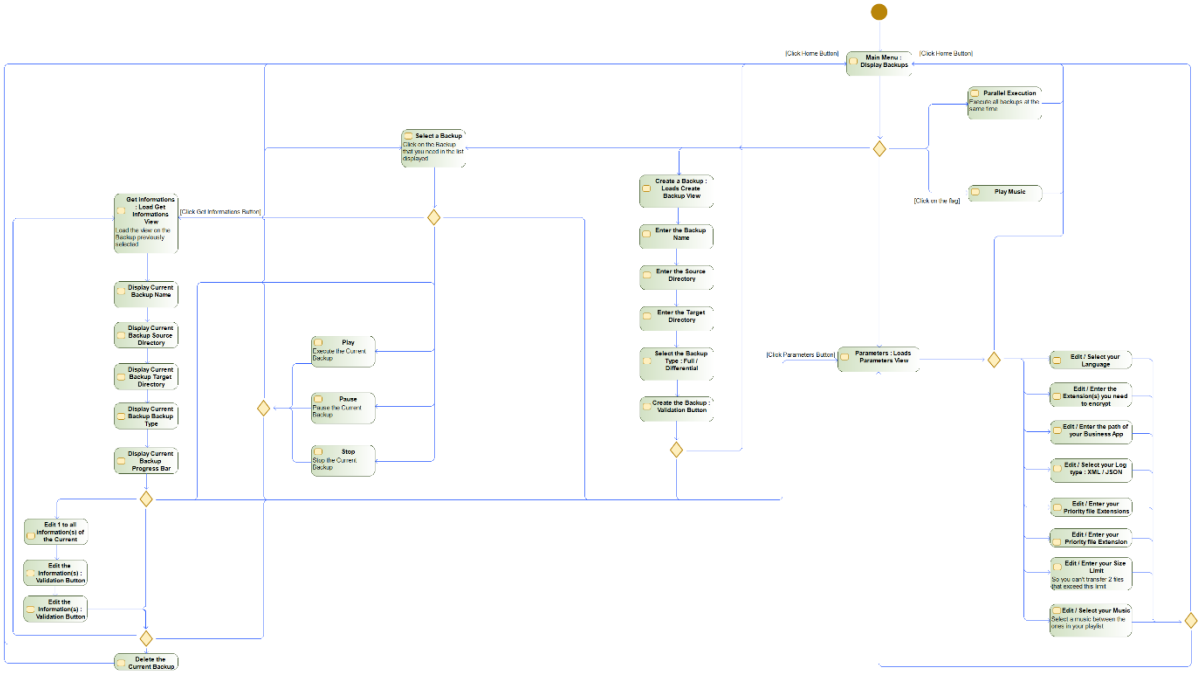


Figure 2: Activity Diagram

An activity diagram is a diagram showing the triggering of events based on system states. It allows to express a temporal dimension of the use cases of the software. Representing a software use case by an activity diagram is like algorithmically translating the use case.

I.3 Class diagram

To see the image better please click on the following link:
https://github.com/gamaticow/EasySave/blob/3.0/UML/class_diagram.png

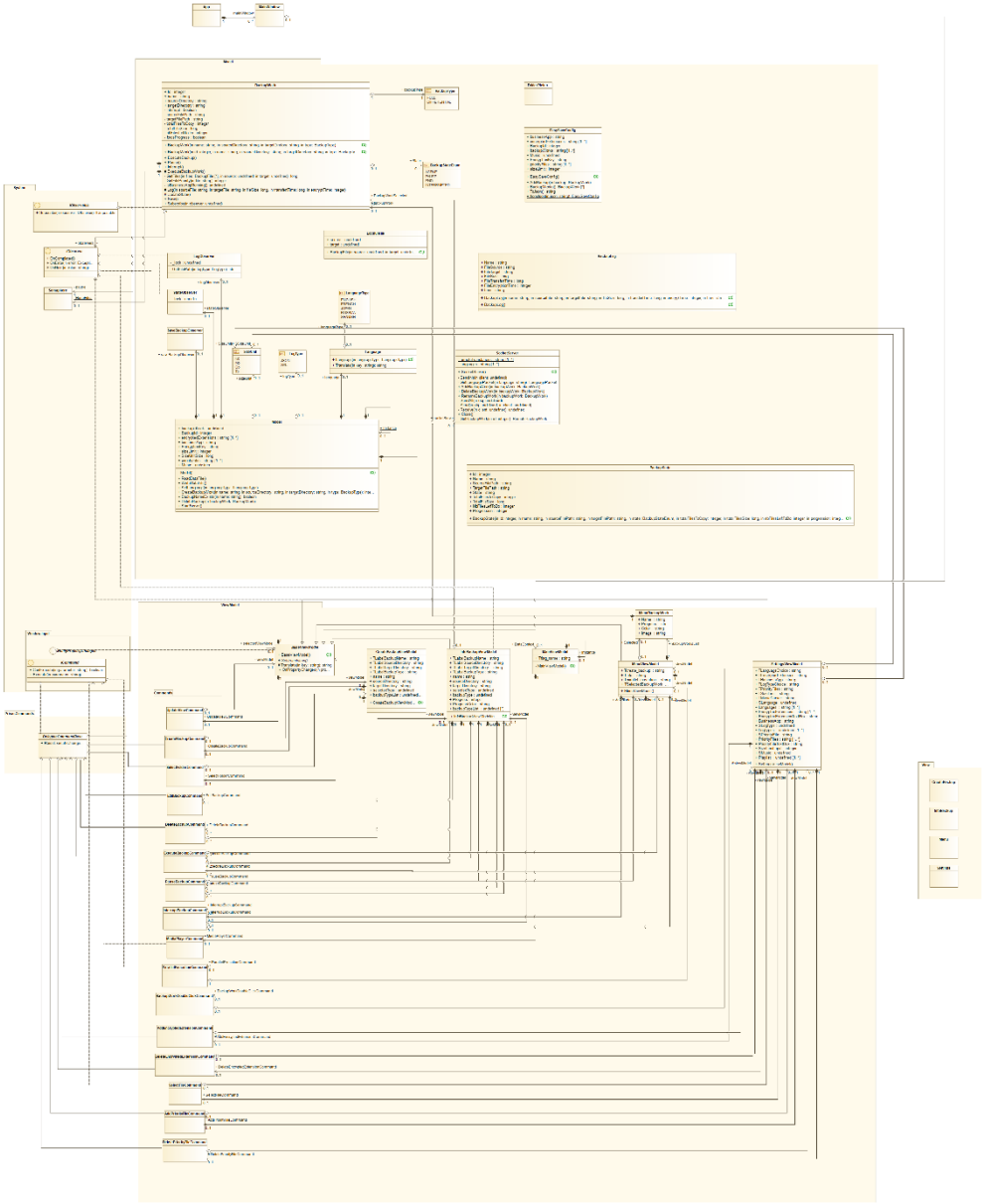


Figure 3; Class diagram

In order to have the structure of the software to be implemented, the class diagram intervenes by showing the classes and interfaces of the program as well as the relationships between the different classes that can be of type association, aggregation, composition and inheritance.

The class diagram of our software presents the classes by grouping them according to their nature in the MVVM (Model View ViewModel) architecture.

I.4 Sequence diagrams

The sequence diagram is a diagram illustrating the chronological interactions between objects in a system (in our case backup software) for a given software use case.

I.4.1 Choice of language

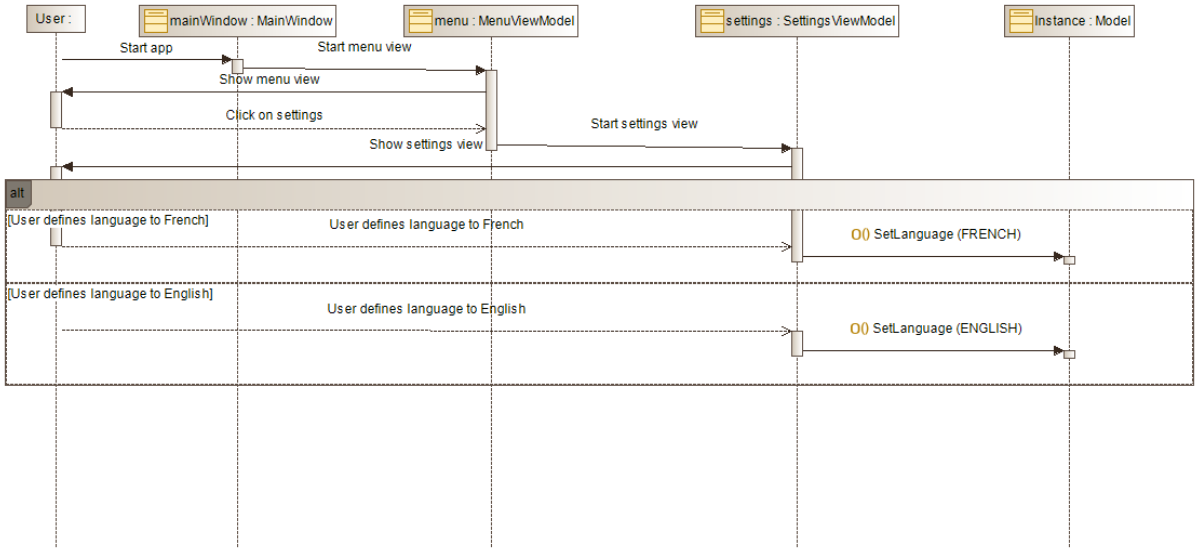


Figure 4: Sequence diagram – language choice

Figure 4 illustrates, from the sequence diagram, the necessary interactions between the objects of the software to make a language change (two possible language options – English and French).

I.4.2 Parallel execution

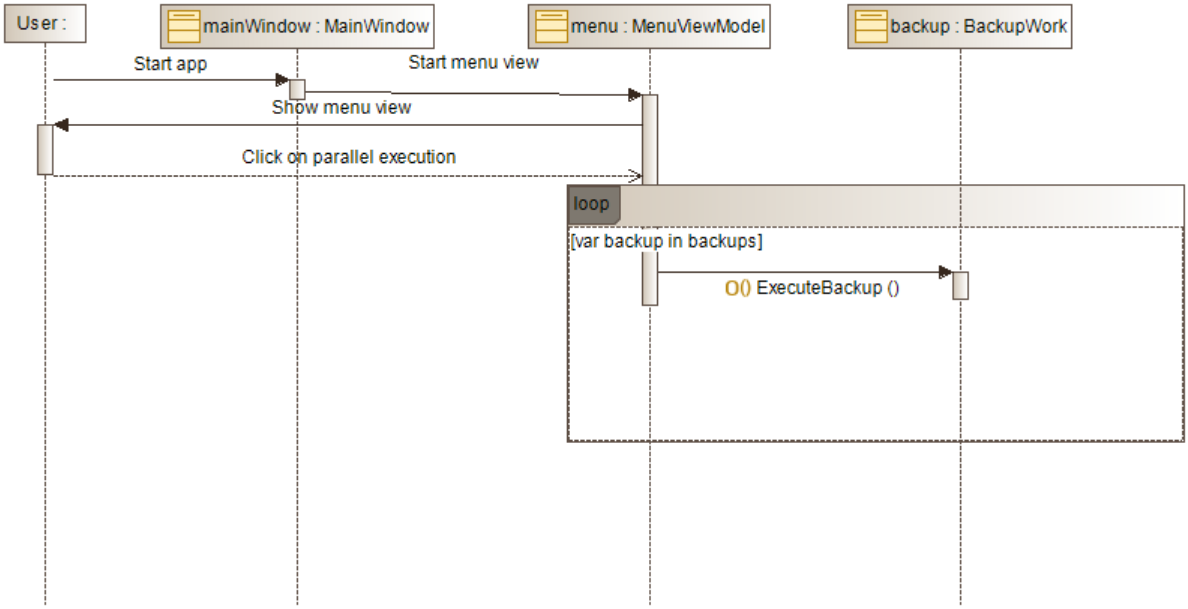


Figure 5: Sequence Diagram – Sequential Execution

Figure 5 illustrates the interactions required between the user and the software objects to perform a sequential backup job.

I.5 Component diagram

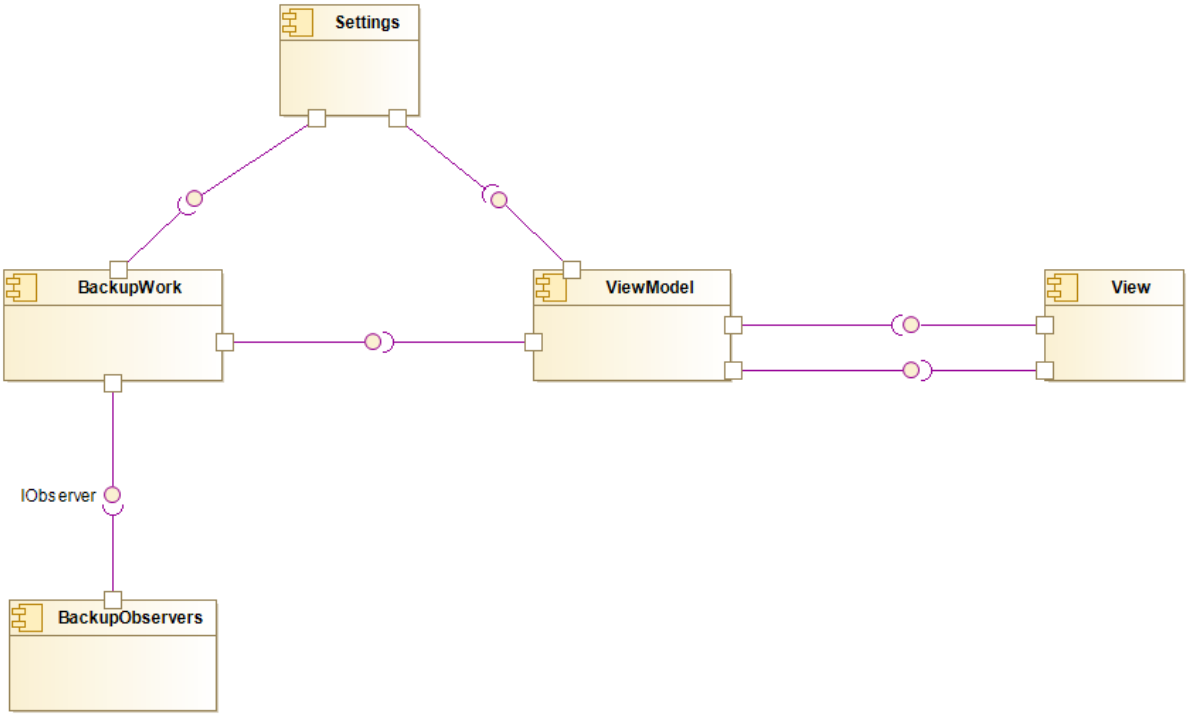


Figure 6: Component Diagram

The component diagram is a diagram showing an overview of the system components and the dependencies between those components. Components can be classes, class sets, packages, software libraries, and so on.

Our diagram (Figure 6) shows the components (BackupWork, ViewModel, View BackupObservers, Settings) and their mutual dependencies.

II. TOOLS AND TECHNOLOGIES USED

II.1 Design patterns

In order to facilitate software design, design patterns that have already proven themselves to software developers, come into play. There are a multitude of design patterns; each solving a specific problem. According to Wikipedia, a design patron is a caracteristic arrangement of modules, recognized as good practice in response to a software design problem.

Regarding version 1 of our backup software, two design patterns namely the **MVVM** architecture (Model - View - View-Model) and the "**Observer"** pattern have been implemented.

The MVC architecture is an architecture motif segmented into three layers namely:

- A template containing the data to display.
- A view containing the presentation of the software interface.
- And a model view. this component is the link between the model and the view. It takes care of managing data links and possible conversions. This is where binding comes in.

The "Observer" design pattern is used to send signals to software modules called observers to take actions based on these signals (notifications). In the case of our backup software, it is used for updating logs.

II.2 DevOps

II.2.1 Azur DevOps GIT

In order to have a versionnage of our backup software during the design as well as to allow collaborative work we used GIT which is a decentralized version management software.

II.2.2 GitHub

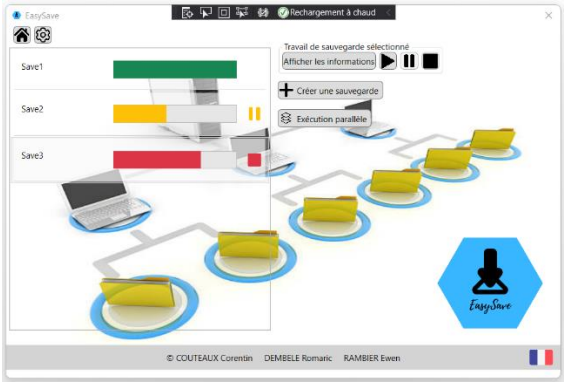
The remote development management of our software was carried out with GitHub.

II.3 Development Platform

The development of the first version of our software was carried out on Visual Studio 2019 with the .Net Core Framework (5. 0) and using Nuget packages : Newtonsoft.Json (13.0.1), Prism.Core 8.1.97, and OptimizedPriorityGueue (5.1.0).

III. USER DOCUMENTATION

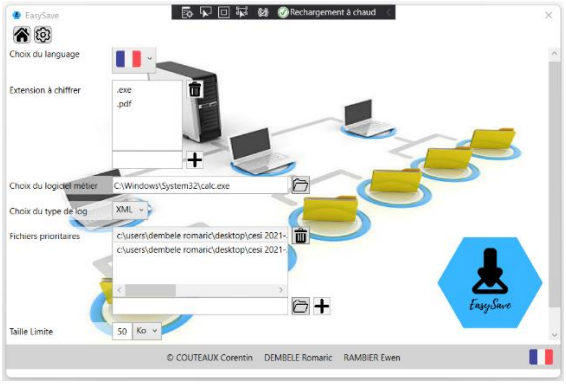
EasySave 3.0 offers the following features:



Main menu of the software

From this menu, the user has the possibility to create a backup job, run it, pause it or stop it when it runs. We then obtain the colors respectively green, yellow or red depending on the status of the backup job (running, paused or stopped). (see Figure 7)

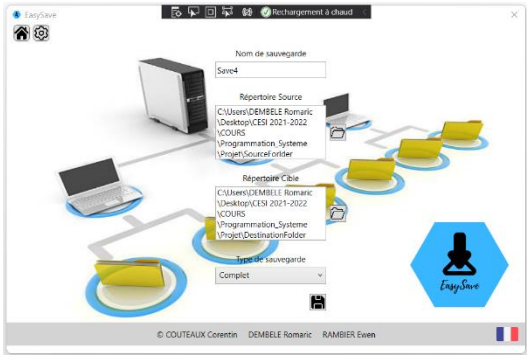
Figure 7: Menu principal



Configuring software settings

The software offers the possibility to configure certain parameters that will be considered when performing backup jobs. These parameters include, among others, the choice of language, the choice of file types that will be encrypted, the choice of business software, the type of log file that will be generated, priority files and the size limit of a file that can participate in a simultaneous backup. (Figure 8)

Figure 8: Parameters

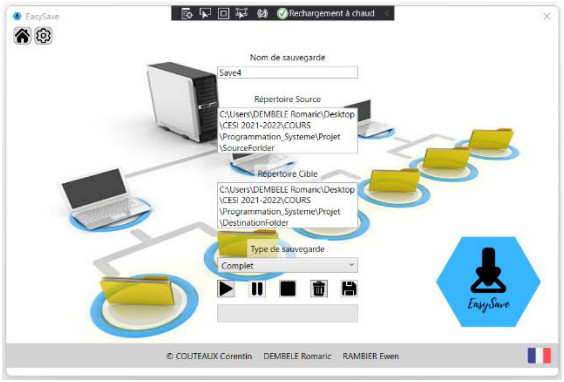


Page for creating a backup job

On this page, the user enters the information necessary to create a backup job namely, the name of the backup, the source directory, the target directory, the type of backup (full or differential). (Figure 9)

Figure 9: Job Description Page

backup

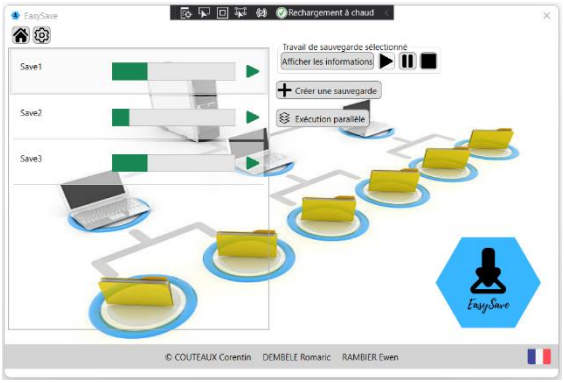


backup job

Actions on a backup job

On the main menu (see Figure 7), there is a "Show information" button that allows you to display the information about a backup job selected from the list of jobs always from the main menu. If the actions (run, pause, stop, delete, and edit) will be available for this selected job. (Figure 10)

Figure 10: Information page of a



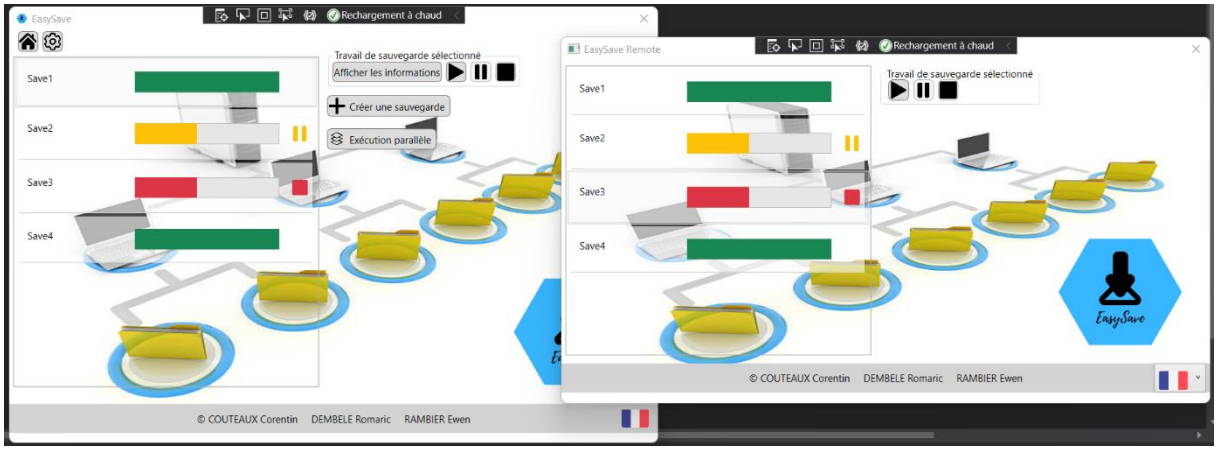
Parallel execution

All backup jobs can be started with a single click by pressing the "Parallel Run" button. The execution of all this work will then be done simultaneously. (Figure 11)

Figure 11: Parallel Execution

Remote console

The user also has the possibility to follow in real time from a remote console, the progress of the backup work and can act on them remotely. (Figure 12)



.... Figure 12 : Application

IV. PROPOSALS FOR IMPROVEMENT FOR VERSION 4.0

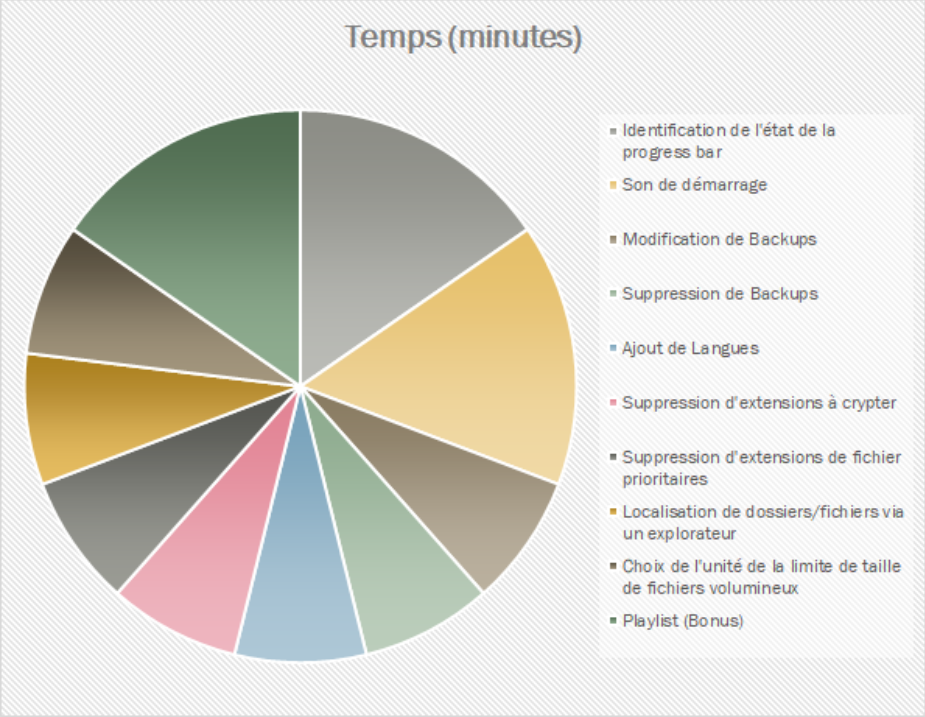


Figure : Enhancements already implemented in version 3.0

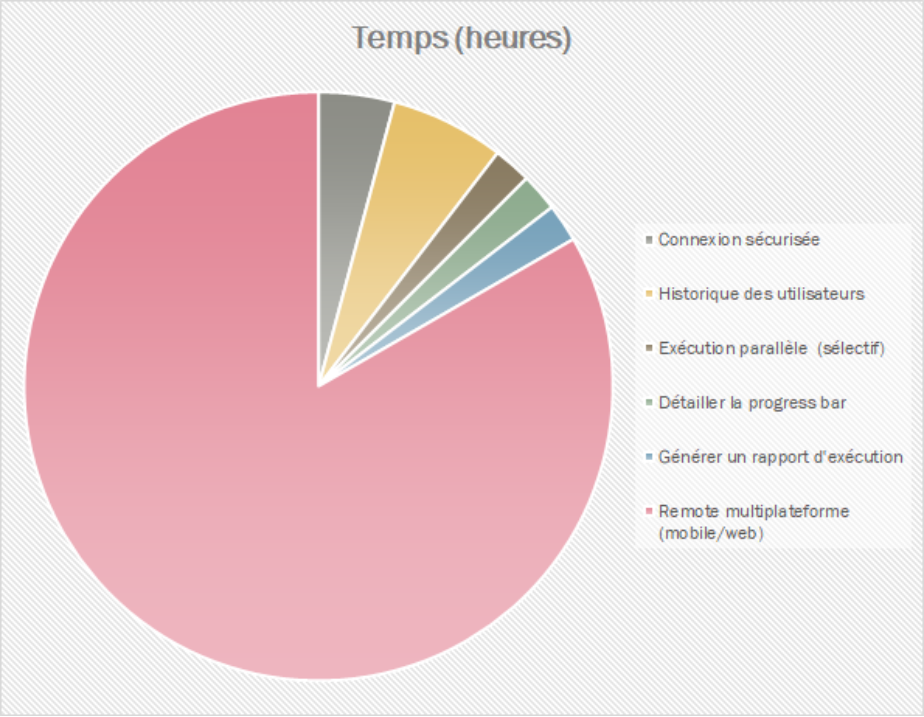


Figure : Improvements for version 4.0

CONCLUSION

By way of conclusion, through the diagrams of use cases, activity, classes, sequence and components, the modeling of the EasySave backup software allowed an easier implementation, considering all the specifications of the specifications. When it comes to software versioning and collaborative work, the GIT and GitHub tools have been a great asset.

The segmentation of the software code allowed to benefit from a modularity and scalability of the software. Thus, version 3.0 benefited from this modularity of the software thus allowing to develop the graphical application by reusing the elements of the console application.