11/02/2025

PROJET GENIE LOGICIEL – Livrable 2-v1.1

Diagrammes



GROUPE 4

HOFSTETTER Léo – VASSILENKO mark – SANGLIER arthur – DEPPNER Aurélien

Table des matières

[Introduction 2](#_Toc190180745)

[Diagramme de cas d'utilisation 2](#_Toc190180746)

[Diagramme d'activité 4](#_Toc190180747)

[Diagramme de classes 6](#_Toc190180748)

[Diagramme de séquence 9](#_Toc190180749)

[Conclusion 12](#_Toc190180750)

# Introduction

Avant de développer l'application, nous avons réalisé plusieurs diagrammes pour avoir une vue d'ensemble de ce que doit faire l'application.

Ces diagrammes permettent de mieux comprendre le système et facilitent la communication avec les parties prenantes du projet. Ils sont également essentiels pour assurer une documentation technique structurée et garantir la maintenance et l’évolution du projet.

# Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation représente l’ensemble des interactions possibles entre les utilisateurs et l’application. Il permet de visualiser les différentes fonctionnalités offertes par le système et d’identifier les rôles des différents acteurs.

Voici les fonctionnalités de notre application :

* Créer une sauvegarde : L'utilisateur définit un nom, un répertoire source et cible, ainsi qu’un type de sauvegarde (complète ou différentielle).
* Exécuter une sauvegarde : L'utilisateur sélectionne une sauvegarde existante et la lance.
* Supprimer une sauvegarde : Permet de retirer une tâche enregistrée.
* Changer la langue : L'utilisateur peut modifier la langue de l'interface entre le français et l’anglais.
* Gérer les logs : L'application enregistre des informations sur les tâches exécutées dans des fichiers logs. Ces fichiers peuvent être consultés par l’utilisateur.

Ce diagramme est particulièrement utile pour la phase de spécification fonctionnelle et pour la présentation aux clients.

Voici notre diagramme de cas d’utilisation :

Une image contenant texte, carte, diagramme, Plan

Le contenu généré par l’IA peut être incorrect.

# Diagramme d'activité

Le diagramme d'activité permet de représenter le déclenchement des évènements en fonction de l'état du système. Il sert également à modéliser des processus pouvant s'exécuter en parallèle.

Dans notre projet, il montre le déroulement des différentes actions, comme la validation des paramètres avant la création d'une sauvegarde ou la mise à jour de l'état du système pendant l'exécution d'une sauvegarde.

L’utilisateur à trois possibilités :

* Lister les tâches de sauvegarde en saisissant le numéro 1. Il pourra ensuite saisir un nom de tâche qui sera ensuite exécutée.
* Créer une tâche de sauvegarde en saisissant le numéro 2. Il devra ensuite saisir différents paramètres :
  + Le nom de la sauvegarde.
  + Le répertoire source.
  + Le répertoire cible.
  + Le type de sauvegarde (complète ou différentielle.
* Supprimer une tâche de sauvegarde. Il pourra ensuite saisir le nom de la tâche qu’il veut supprimer.

Pour chacune de ces fonctionnalités, un message de validation ou un message d’erreur s’affiche en fonction du résultat de l’exécution.

De plus, l’utilisateur pourra modifier la langue de l’application (passer du français à l’anglais ou de l’anglais au français).

Voici notre diagramme d’activité :

Une image contenant texte, diagramme, Plan, schématique

Le contenu généré par l’IA peut être incorrect.

# Diagramme de classes

Le diagramme de classes définit l’architecture du projet en décrivant les classes et leurs relations. Il assure une séparation des responsabilités claire et facilite la maintenance du code.

L’architecture est divisée en plusieurs couches bien définies :

* 1️ Modèles (Models)

Ce sont les représentations des données stockées et manipulées dans l’application. Par exemple :

BackupTask : représente une tâche de sauvegarde avec ses paramètres (source, destination, type, liste de fichiers…).

BackupState : stocke l’état actuel d’une sauvegarde (en attente, en cours, terminée…).

* 2️ Contrôleurs (Controllers)

Ces classes gèrent la logique métier et les interactions entre l’utilisateur et les modèles.

JobController : orchestre la création, exécution et suppression des tâches de sauvegarde.

LogController : gère les événements de logs et notifie les observateurs (pattern Observer).

* 3️ Services

Ils encapsulent la logique applicative et facilitent l’implémentation des fonctionnalités. Par exemple :

FileManager : gère les fichiers et répertoires à sauvegarder.

LogService : stocke les événements et erreurs dans des fichiers de logs.

Voici le détail des Classes et Design Patterns

* BackupTask (Modèle de Sauvegarde)
  + Représente une tâche de sauvegarde.
  + Stocke les informations essentielles :
    - Répertoire source et destination.
    - Type de sauvegarde (complète ou différentielle).
    - Liste des fichiers à inclure.
* LogController (Pattern Observer)
  + Implémente le pattern Observer pour la gestion des logs.
  + Maintient une liste d’observateurs (ILogListener), qui sont notifiés à chaque mise à jour.
  + Permet un suivi en temps réel des actions du système.

Exemple de fonctionnement :

1️ Une tâche de sauvegarde s’exécute.

2️ LogController enregistre un événement.

3️ Il informe tous les observateurs (LogFileListener), qui écrivent les logs dans un fichier.

* IType (Pattern Strategy)
  + Définit une interface commune pour les stratégies de sauvegarde.
  + Implémente deux stratégies concrètes :
    - CompleteType : sauvegarde complète.
    - DifferentialType : sauvegarde différentielle (ne prend que les fichiers modifiés).

Exemple de fonctionnement :

1️ L’utilisateur choisit un type de sauvegarde.

2️ JobController applique la stratégie correspondante (CompleteType ou DifferentialType).

3️ La sauvegarde est effectuée selon la logique du type sélectionné.

* BackupSnapshot & SnapshotController (Pattern Memento)
  + BackupSnapshot stocke l’état d’une sauvegarde à un instant T.
  + SnapshotController permet de restaurer un état précédent en cas d’erreur.

Exemple de fonctionnement :

1️ Une sauvegarde est créée et modifiée.

2️ À chaque changement, un instantané (snapshot) est enregistré.

3️ Si besoin, l’utilisateur peut revenir en arrière pour annuler une modification incorrecte.

Voici notre diagramme de classes :

[Une image contenant texte, diagramme, reçu, Plan

Le contenu généré par l’IA peut être incorrect.](https://viacesifr-my.sharepoint.com/:i:/g/personal/leo_hofstetter_viacesi_fr/ETajQmbwmDJDrZMR3LeqbvQB0F_aTjzujXAdsmGZCKieNQ?e=cFrgLR)

[Lien vers le diagramme :](https://viacesifr-my.sharepoint.com/:i:/g/personal/leo_hofstetter_viacesi_fr/ETajQmbwmDJDrZMR3LeqbvQB0F_aTjzujXAdsmGZCKieNQ?e=cFrgLR)

(Attention ce lien expire le 19 mars 2025, s’il tel est le cas à l’heure où vous lisez ces lignes, contacter HOFSTETTER Léo)

# Diagramme de séquence

Le diagramme de séquence permet de visualiser le déroulé des échanges entre les composants du système. Il met en évidence le flux des appels de méthodes et la communication entre les objets dans un ordre chronologique précis.

Voici les différents scénarios :

1️ Création d’une sauvegarde

* + L’utilisateur demande la création d’une sauvegarde.
  + JobController reçoit la requête et valide les informations fournies.
  + BackupTask est créé avec les paramètres (source, destination, type).
  + BackupSnapshot enregistre l’état initial avant l’exécution.
  + SnapshotController sauvegarde l’instantané pour permettre une restauration ultérieure.
  + LogController génère une entrée de log.
  + L’état du système est mis à jour et un message de confirmation est affiché à l’utilisateur.

2 Exécution d’une sauvegarde

* + L’utilisateur sélectionne une tâche existante.
  + JobController récupère les informations de la tâche et définit la stratégie de sauvegarde (CompleteType ou DifferentialType).
  + FileManager liste les fichiers concernés et les envoie à BackupTask.
  + BackupTask copie les fichiers vers la destination.
  + LogController enregistre chaque étape (début, progression, fin).
  + StateManager met à jour l’état en temps réel (En cours → Terminé).
  + Une notification est envoyée à l’utilisateur une fois la sauvegarde complétée.

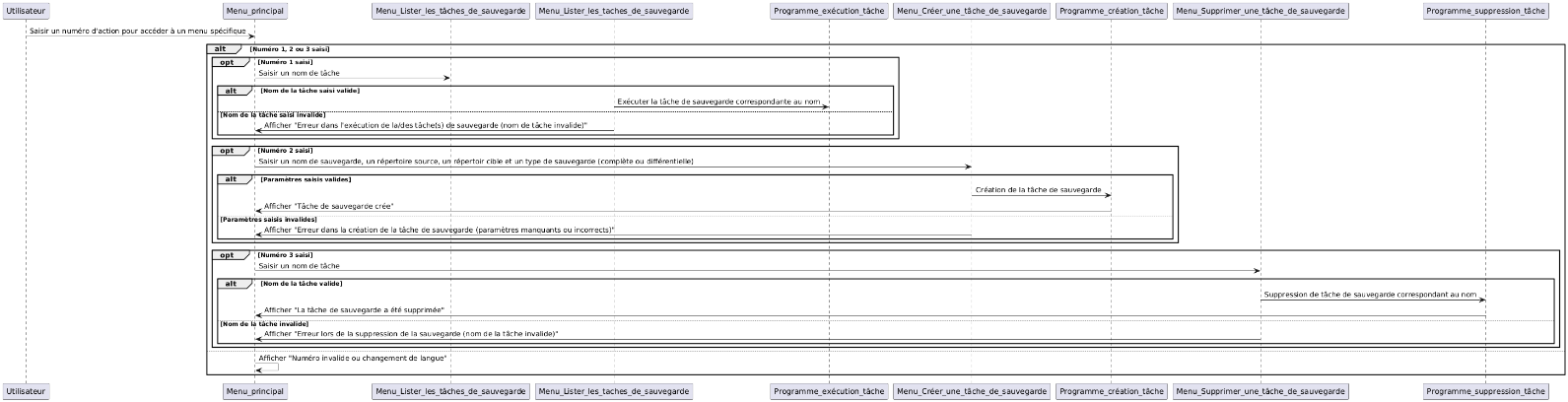
3️ Suppression d’une sauvegarde

* + L’utilisateur demande la suppression d’une sauvegarde.
  + JobController vérifie que la sauvegarde existe.
  + BackupSnapshot crée un instantané de l’état avant suppression.
  + JobController supprime les fichiers concernés via FileManager.
  + StateManager met à jour l’état et supprime la référence de la sauvegarde.
  + LogController ajoute un log indiquant la suppression.
  + Un message de confirmation est affiché à l’utilisateur.

4️ Gestion des logs (Pattern Observer)

* + Un événement système est déclenché (exécution, suppression, erreur).
  + LogController capture l’événement et notifie tous les observateurs (ILogListener).
  + LogFileListener écrit les logs dans un fichier journal.
  + SnapshotController peut créer un point de restauration si nécessaire.
  + L’état du système est mis à jour avec les nouvelles entrées de logs.

Voici notre diagramme de séquence :



[Lien vers le diagramme](https://cdn.discordapp.com/attachments/1333359207818203220/1334583499046387813/Diagramme_de_sequence.png?ex=67ac387d&is=67aae6fd&hm=6a7899a3ab22ac726c1c4bda8e774de166c39cf0673fdd891feeedbbe71a3999&)

# Conclusion

Ces diagrammes forment une documentation essentielle pour le développement du projet. Ils permettent :

* Une meilleure compréhension des fonctionnalités du système.
* Une visualisation claire des interactions entre les composants.
* Une application des bonnes pratiques de conception avec des Design Patterns robustes.

Cette approche assure également une évolutivité et une maintenabilité optimales du projet.