19/02/2025

PROJET GENIE LOGICIEL – Livrable 2-v2

Diagrammes



GROUPE 4

HOFSTETTER Léo – VASSILENKO mark – SANGLIER arthur – DEPPNER Aurélien

Table des matières

[Introduction 2](#_Toc190851920)

[Diagramme de cas d'utilisation 2](#_Toc190851921)

[Diagramme d'activité 5](#_Toc190851922)

[Diagramme de classes 7](#_Toc190851923)

[Diagramme de séquence 11](#_Toc190851924)

[Conclusion 16](#_Toc190851925)

# 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

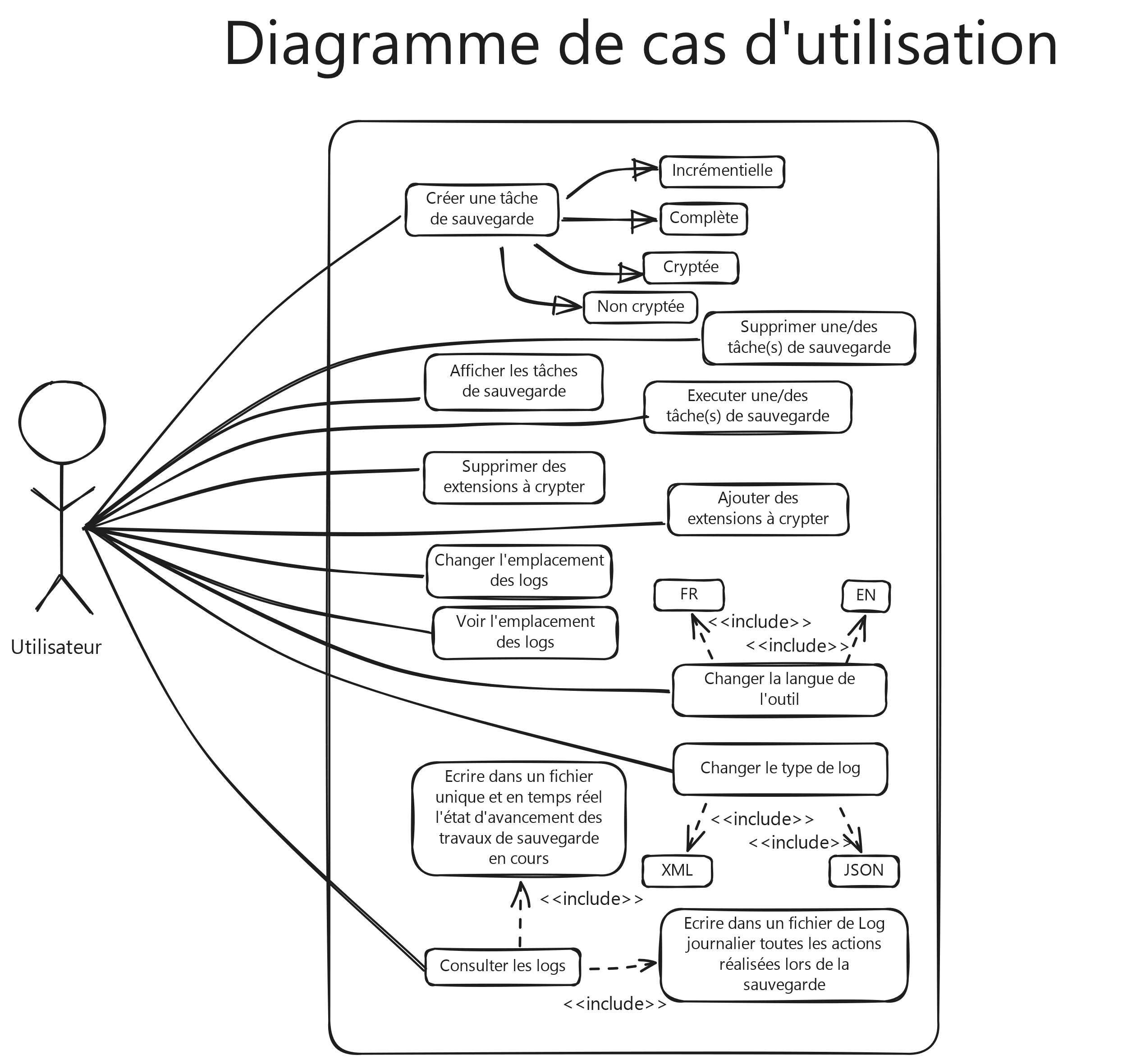
Le diagramme de cas d'utilisation représente l’ensemble des interactions possibles entre les utilisateurs et l’application. Il permet de visualiser les différentes fonctionnalités offertes par le système et d’identifier les rôles des différents acteurs.

Voici les fonctionnalités de notre application :

* Créer une sauvegarde : L'utilisateur définit un nom, un répertoire source et cible, un type de sauvegarde (complète ou différentielle), il peut aussi spécifier si la sauvegarde doit être cryptée.
* Afficher les sauvegardes : L’utilisateur peux afficher toutes les sauvegardes qui ont été créés.
* Exécuter une sauvegarde : L'utilisateur sélectionne une sauvegarde existante et la lance. Il peut également décider d’exécuter plusieurs tâches de sauvegardes en maintenant Ctrl ou Maj.
* Supprimer une sauvegarde : L’utilisateur peut retirer une tâche enregistrée. Il peut également décider d’exécuter plusieurs tâches de sauvegardes en maintenant Ctrl ou Maj.
* 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 en temps réel et dans des fichiers logs journalier. Ces fichiers peuvent être consultés par l’utilisateur.
* Modifier les types de logs : Choisir entre XML et JSON
* Modifier le répertoire de logs : Choisir le dossier dans lequel seront écrit les logs.
* Ajouter et supprimer les extensions à crypter : Permet à l’utilisateur d’ajouter et de supprimer dans une liste les extensions qui seront cryptées pour les sauvegardes cryptées.
* Modifier les logiciels métiers qui bloqueront l’exécution d’une sauvegarde : Permet à l’utilisateur de modifier les logiciels métiers qui bloqueront l’exécution d’une sauvegarde.

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

Voici notre diagramme de cas d’utilisation au format SVG :

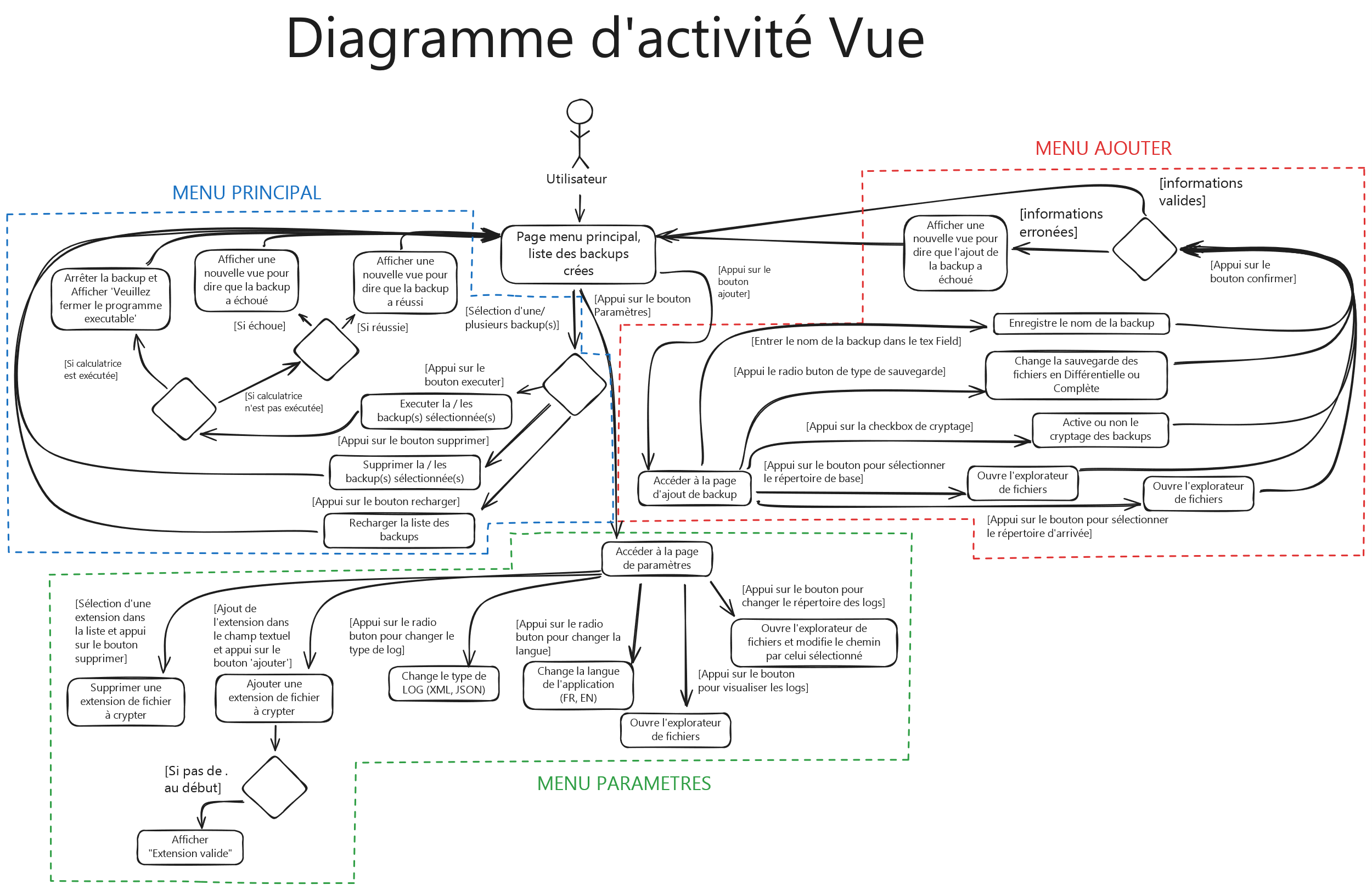


# Diagramme d'activité

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.

Voici notre diagramme d’activité au format SVG :



# Diagramme de classes

Le diagramme de classes représente une architecture bien structurée, avec plusieurs couches assurant une séparation des responsabilités claire. Cette organisation facilite la maintenance, l’extension et l’évolution du projet.

**1. Couches de l'Architecture**

L’application est structurée en plusieurs couches bien distinctes :

1. Modèles (Models)

Les classes modèles représentent les données essentielles utilisées dans l’application.

* BackupModel : Représente une tâche de sauvegarde avec ses informations principales (nom, chemin source/destination, type de sauvegarde).
* BackupState : Stocke l’état d’une sauvegarde (en cours, terminée, erreur, etc.).
* FileTransferInfo : Contient des détails sur les fichiers transférés (chemin, taille, durée de transfert).

1. Contrôleurs (Controllers)

Les contrôleurs gèrent la logique métier et orchestrent les interactions entre les modèles et les services.

* BackupController : Responsable de la gestion des sauvegardes (ajout, exécution, suppression).
* LogController : Gère les logs et informe les observateurs en temps réel (Pattern Observer).
* LangController : Gère le changement de langue de l’application.

1. Services

Les services contiennent la logique principale de l’application et offrent des fonctionnalités avancées.

* BackupService (Implémente IBackupService) :
  + Gère l’exécution et le chargement des sauvegardes.
  + Responsable de la création et suppression des tâches.
* ConfigurationService :
  + Gère la configuration de l’application (fichier JSON).
  + Permet de charger et récupérer les paramètres.

1. Gestion des Fichiers

Les classes responsables du traitement des fichiers et de la gestion des transferts.

* FileController :
  + Assure la copie des fichiers (CopyDirectory, CopyModifiedFiles).
* File :
  + Offre des méthodes utilitaires pour copier et modifier des fichiers.

1. Gestion des Logs

L’application met en place une gestion avancée des logs.

* LogController :
  + Centralise la gestion des événements et erreurs.
* LogFileListener (Pattern Observer) :
  + Écoute les événements de logs et les écrit dans un fichier.
* RealTimeLogListener :
  + Stocke les logs en temps réel et permet leur consultation immédiate.

**2. Design Patterns Utilisés**

L’application repose sur plusieurs design patterns qui permettent d’avoir une architecture modulaire et évolutive.

1. Singleton

Pattern utilisé dans : ConfigurationService, LogController, BackupController, LangController.

* Objectif : Garantir qu’une seule instance de ConfigurationService existe dans toute l’application.
* Pourquoi ?
  + Évite la duplication des paramètres de configuration.
  + Assure une lecture cohérente des fichiers de configuration JSON.

1. Observer

Pattern utilisé dans : LogFileListener, RealTimeLogListener

* Objectif : Permet de notifier plusieurs observateurs lorsqu’un événement se produit (gestion des logs).
* Pourquoi ?
  + Facilite l’ajout de nouveaux observateurs sans modifier la logique principale.
  + Permet d’avoir des logs en temps réel et des fichiers journaux séparés.

Exemple de fonctionnement :  
1️ LogController enregistre un événement de log.  
2️ Il notifie tous les observateurs (LogFileListener, RealTimeLogListener).  
3️ Chaque observateur traite l’information (ex : écrit dans un fichier log).

1. Strategy

Pattern utilisé dans : IType, CompleteType, DifferentialType

* Objectif : Permet de choisir dynamiquement entre plusieurs stratégies de sauvegarde.
* Pourquoi ?
  + Permet d’ajouter facilement d’autres types de sauvegarde sans modifier le code principal.
  + Applique le principe Open/Closed (OCP).

Exemple de fonctionnement :  
1️ BackupController demande une sauvegarde.  
2️ Selon le type sélectionné (CompleteType ou DifferentialType), la stratégie adéquate est exécutée.

Voici notre diagramme de classes :

[Lien vers le diagramme :](https://viacesifr-my.sharepoint.com/:u:/g/personal/leo_hofstetter_viacesi_fr/EWr-cODYEPJChgy8TS5M88oBlmctEsgMeLNWN0hmuU_fgg?e=O14fiY)

(Attention le lien OneDrive expire le 21 mars, si tel est le cas quand vous lisez ces lignes, contactez HOFSTETTER Léo)

# Diagramme de séquence

Le diagramme de séquence illustre les interactions entre les composants du système en suivant un ordre chronologique. Il permet de visualiser le flux des appels de méthodes et la communication entre les objets.

Voici les différents scénarios :

1️) Lister les sauvegardes  
2️) Créer une sauvegarde  
3️) Exécuter une sauvegarde  
4️) Supprimer une sauvegarde  
5️) Changer la langue de l'application  
6️) Changer le format des logs

7) Modifier le logiciel métier qui bloque l’exécution d’une Backup

8) Changer le répertoire des logs.

**1️ Lister les sauvegardes**

Objectif : Afficher toutes les sauvegardes existantes.

Déroulement des actions :

1. L'utilisateur demande à voir la liste des sauvegardes.
2. JobController interroge BackupService pour récupérer toutes les sauvegardes.
3. BackupService charge les sauvegardes à partir du fichier de configuration (ConfigurationService).
4. Les informations sont envoyées à JobController.
5. L’interface utilisateur affiche la liste des sauvegardes disponibles.

**2️ Création d’une sauvegarde**

Objectif : Ajouter une nouvelle sauvegarde.

Déroulement des actions :

1. L’utilisateur entre les paramètres de la sauvegarde et valide.
2. JobController reçoit la requête et vérifie les informations.
3. BackupTask est créé avec :
   * Nom
   * Source
   * Destination
   * Type de sauvegarde (Complète ou Différentielle)
4. BackupSnapshot enregistre un instantané avant l’exécution.
5. SnapshotController sauvegarde l’instantané pour une future restauration.
6. LogController génère une entrée de log indiquant la création de la sauvegarde.
7. L’état du système est mis à jour.
8. Un message de confirmation est affiché à l’utilisateur.

**3️ Exécution d’une sauvegarde**

Objectif : Démarrer une sauvegarde existante.

Déroulement des actions :

1. L’utilisateur sélectionne une tâche existante et lance l’exécution.
2. JobController récupère les informations de la tâche et sélectionne la stratégie de sauvegarde (CompleteType ou DifferentialType).
3. FileController liste les fichiers concernés.
4. Les fichiers sont envoyés à BackupTask pour être copiés vers la destination.
5. LogController enregistre chaque étape
6. StateManager met à jour l’état en temps réel :
   * "En cours" → "Terminé"
7. Une notification est envoyée à l’utilisateur.

**4️ Suppression d’une sauvegarde**

Objectif : Supprimer une sauvegarde existante.

Déroulement des actions :

1. L’utilisateur demande la suppression d’une sauvegarde.
2. JobController vérifie que la sauvegarde existe.
3. BackupSnapshot crée un instantané de l’état avant suppression.
4. JobController demande à FileController de supprimer les fichiers concernés.
5. StateManager met à jour l’état et supprime la référence de la sauvegarde.
6. LogController ajoute une entrée de log pour indiquer la suppression.
7. Un message de confirmation est affiché à l’utilisateur.

**5️ Changer la langue de l’application**

Objectif : Modifier la langue de l’interface.

Déroulement des actions :

1. L’utilisateur sélectionne une nouvelle langue dans les paramètres.
2. LangController reçoit la requête et met à jour la configuration.
3. ConfigurationService enregistre la nouvelle langue dans le fichier JSON de configuration.
4. Tous les éléments de l’interface sont mis à jour pour refléter la nouvelle langue.
5. Un message de confirmation est affiché.

**6️ Changer le format des logs**

Objectif : Modifier la façon dont les logs sont enregistrés (ex : texte, JSON, XML).

Déroulement des actions :

1. L’utilisateur choisit un nouveau format de logs dans les paramètres.
2. LogController met à jour la configuration du format.
3. ConfigurationService enregistre le nouveau format dans le fichier JSON.
4. Tous les nouveaux logs seront désormais enregistrés dans le nouveau format choisi.
5. Un message de confirmation est affiché.

Nous avons réalisé le diagramme de séquence pour la fonctionnalité permettant d’ajouter une sauvegarde.

Une image contenant texte, capture d’écran, nombre, Parallèle

Le contenu généré par l’IA peut être incorrect.

[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.