EXIA

Livrable 0 et Livrable 1 : (EasySave version 1.0):

Membres du groupe:
-BENAMEUR FARES
-SEDDIK AIT KHALED AMINE
-GESSOUM MOHAMED
-STAMBOULI LOKMAN
-BELOUCHRANI MOHAMED

EXIA

Contents

1. INTRODUCTION :

Dans le cadre de notre intégration au sein de l'éditeur de logiciels ProSoft, notre équipe a été chargée de piloter et de développer le projet EasySave. Ce projet consiste à concevoir un logiciel de sauvegarde performant, fiable et facilement maintenable, s'intégrant pleinement à la suite logicielle de ProSoft, tout en respectant les standards techniques et méthodologiques imposés par la direction.

Le logiciel EasySave est destiné à être commercialisé au prix unitaire de 200 € HT, avec un contrat de maintenance annuel incluant mises à jour et support technique, basé sur l'indice SYNTEC.

Ce contrat garantit une assistance 5 jours sur 7, de 8h à 17h. Notre mission s'étend au-delà du simple développement : elle comprend également la gestion des versions (majeures et mineures), la maintenance du code, ainsi que la production de documentations à destination des utilisateurs finaux et de l'équipe de support technique.

Ce document regroupe les premières étapes du développement du projet EasySave, un logiciel de sauvegarde confié à notre équipe par l'éditeur de logiciels ProSoft. Ce projet s'inscrit dans le cadre d'une démarche rigoureuse, encadrée par la direction des systèmes d'information (DSI), visant à fournir un outil fiable, maintenable et conforme aux standards de la suite logicielle ProSoft.

Le livrable 0 correspond à la phase de lancement du projet. Il comprend l'élaboration du cahier des charges version 1.0, la définition des besoins fonctionnels et techniques, ainsi que la mise en place de l'environnement de développement. À cette étape, l'espace de travail collaboratif a été préparé et l'accès au dépôt Git a été partagé avec le tuteur de suivi.

Le livrable 1 marque la livraison de la première version fonctionnelle (v1.0) d'EasySave. Il inclut :

- Le logiciel développé en C# avec .NET Framework,
- Les diagrammes UML représentant la structure et le fonctionnement du système (cas d'utilisation, classes, séquences, activités),
- La documentation utilisateur sur une page,
- La note de version (release note) décrivant les fonctionnalités livrées.

Ces deux livrables initiaux posent les bases du projet et s'inscrivent dans une démarche de qualité, de traçabilité et de maintenabilité, afin de garantir une évolution fluide du logiciel dans ses versions futures.

2. Diagramme UML :

Dans le cadre de la modélisation de notre projet, l'utilisation du langage UML (Unified Modeling Language) permet de représenter de manière claire et structurée les différents aspects du système étudié. Ce livrable présente plusieurs types de diagrammes UML, chacun jouant un rôle spécifique dans la compréhension et la conception du système.

2.1 Diagramme de cas d'utilisation (Use Case Diagram) :

Le diagramme Use Case présenté illustre les fonctionnalités principales du logiciel EasySave V1.0, un outil de sauvegarde de données. Ce diagramme met en évidence les interactions entre les acteurs (utilisateurs) et le système, ainsi que les relations entre les différents cas d'utilisation.

Fonctionnalités Principales

Cas d'Utilisation de Base

1. **Choix de la Langue**
    - Permet à l'utilisateur de sélectionner la langue de l'interface (exemple : anglais).
    - Inclus dans d'autres cas d'utilisation pour une expérience utilisateur cohérente.
2. **Nom Identique**
    - Vérifie la validité du nom de la source de sauvegarde.
    - Assure que les noms respectent les critères du système.
3. **Ajout de Sauvegarde (max 5)**
    - Permet d'ajouter jusqu'à cinq sauvegardes.
    - Valide la cible de sauvegarde pour s'assurer qu'elle est accessible et conforme.
4. **Chargement de Sauvegarde**
    - Affiche la liste des sauvegardes disponibles pour sélection.
    - Facilite la gestion des sauvegardes existantes.
5. **Mise à Jour de Sauvegarde**
    - Permet de modifier une sauvegarde existante en choisissant son nom.
    - Inclut la possibilité de mettre à jour les paramètres.
6. **Suppression de Sauvegarde**
    - Offre la fonctionnalité de supprimer une sauvegarde en sélectionnant son nom.
    - Assure une gestion efficace de l'espace de stockage.

7. **Exécution de Sauvegarde**
    - Permet de lancer une sauvegarde en choisissant son type (exemple : différentielle ou complète).
    - Adapte le processus en fonction des besoins de l'utilisateur.
8. **Sortie**
    - Ferme l'application de manière sécurisée.

Extensions Spécialisées

Sauvegarde Différentielle

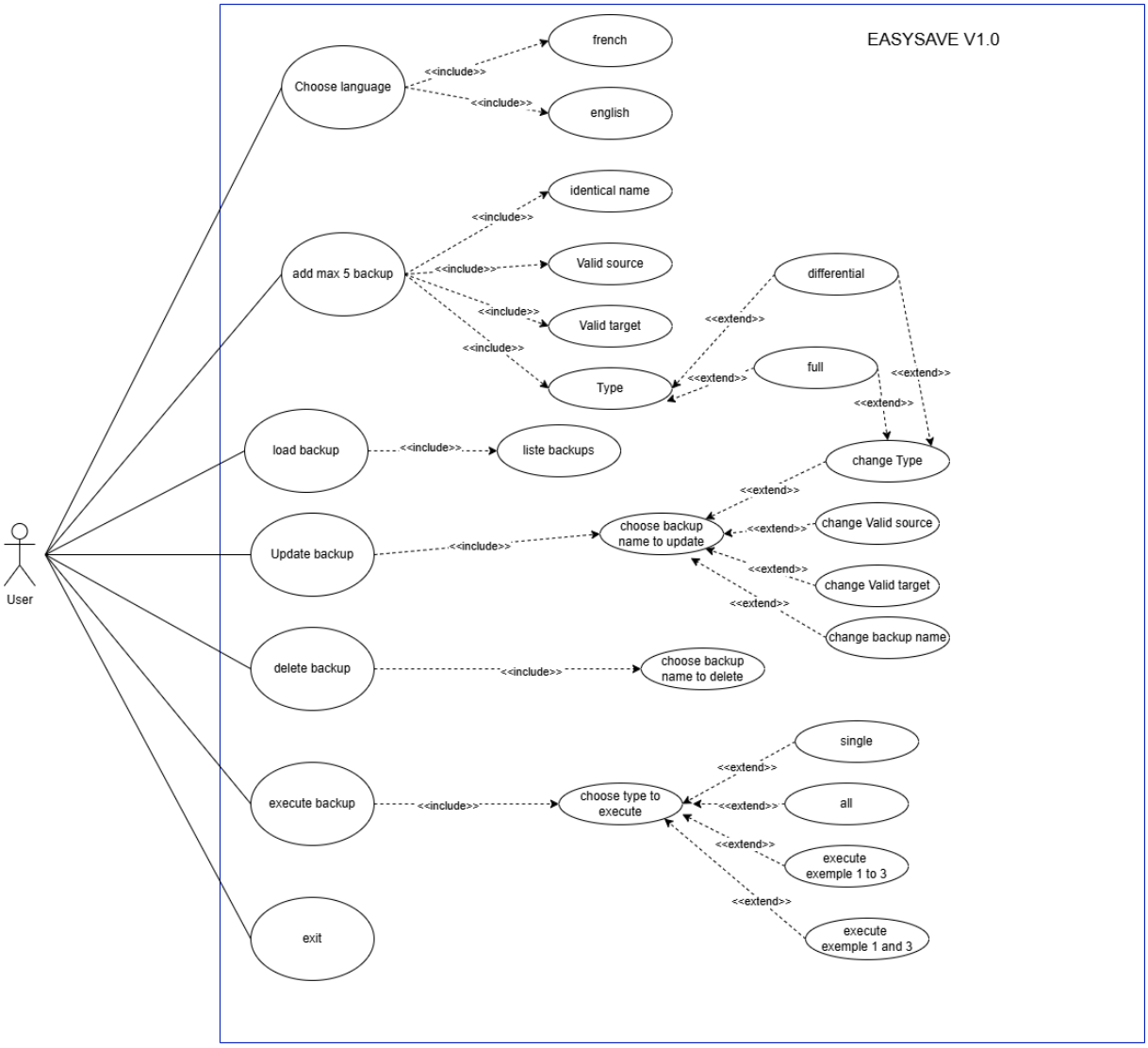
- **Étend** les fonctionnalités de base avec des options avancées :
    - **Sauvegarde Complète** : Effectue une copie intégrale des données.
    - **Changement de Type** : Permet de basculer entre différents types de sauvegarde.
    - **Modification de la Source et de la Cible** : Adapte les chemins source et cible.
    - **Renommage de Sauvegarde** : Offre la possibilité de renommer une sauvegarde existante.

Sauvegarde Unique

- **Étend** les cas d'utilisation pour des scénarios spécifiques :
    - **Exécution Globale** : Lance toutes les sauvegardes configurées.
    - **Exemples 1 à 3** : Illustre des configurations types pour des besoins variés.
    - **Exécution Sélective** : Permet de lancer des sauvegardes spécifiques (exemple : 1 et 3).

Relations entre Cas d'Utilisation

- Les relations **<include>** indiquent que certains cas d'utilisation sont inclus dans d'autres pour une modularité accrue (exemple : choix de la langue intégré dans plusieurs fonctionnalités).
- Les relations **<extend>** montrent comment des cas d'utilisation spécialisés (comme la sauvegarde différentielle) étendent les fonctionnalités de base.


EASYSAVE V1.0
User
Choose language
<<include>>
french
<<include>>
english
add max 5 backup
<<include>>
identical name
<<include>>
Valid source
<<include>>
Valid target
<<include>>
Type
<<extend>>
differential
<<extend>>
full
<<extend>>
<<extend>>
load backup
<<include>>
liste backups
Update backup
<<include>>
choose backup name to update
<<extend>>
change Type
<<extend>>
change Valid source
<<extend>>
change Valid target
<<extend>>
change backup name
delete backup
<<include>>
choose backup name to delete
execute backup
<<include>>
choose type to execute
<<extend>>
single
<<extend>>
all
<<extend>>
execute exemple 1 to 3
<<extend>>
execute exemple 1 and 3
exit

2.2 Diagramme d'activité :

Le diagramme d'activité présenté ci-dessus illustre le fonctionnement global du logiciel EasySave, en modélisant les différents scénarios d'interaction utilisateur à travers les principales fonctionnalités du système. Il s'agit d'une représentation dynamique, décrivant les flux de contrôle entre les actions, conditions et décisions possibles lors de l'utilisation du logiciel.

1. Démarrage de l'application

L'activité commence par le choix de la langue, suivi de l'affichage du menu principal. L'utilisateur peut alors sélectionner l'une des fonctionnalités proposées :

- Ajouter une nouvelle sauvegarde
- Charger une sauvegarde
- Supprimer une sauvegarde
- Mettre à jour une sauvegarde
- Exécuter une sauvegarde

2. Ajout d'une nouvelle sauvegarde

Si l'utilisateur choisit d'ajouter une sauvegarde, une vérification est d'abord effectuée pour s'assurer que le nombre de sauvegardes existantes n'excède pas 5. Si la limite est atteinte, un message d'erreur est affiché.

Sinon, l'utilisateur renseigne :

- Le nom de la sauvegarde
- Le chemin source
- Le chemin de destination
- Le type de sauvegarde (complète ou différentielle)

Une vérification de validité est effectuée. Si les informations sont correctes, la sauvegarde est ajoutée ; sinon, un message d'erreur s'affiche.

3. Chargement d'une sauvegarde

Cette option affiche la liste des sauvegardes existantes si au moins une est présente. Dans le cas contraire, un message indiquant l'absence de sauvegarde est retourné.

4. Suppression d'une sauvegarde

L’utilisateur saisit le nom de la sauvegarde à supprimer. Si celle-ci est trouvée, elle est supprimée ; sinon, un message d’erreur s’affiche.

5. Mise à jour d’une sauvegarde

L’utilisateur saisit le nom de la sauvegarde à modifier. Si elle est trouvée, il peut mettre à jour les éléments suivants (ou les ignorer) :

- Nom
- Chemin source
- Chemin de destination
- Type de sauvegarde

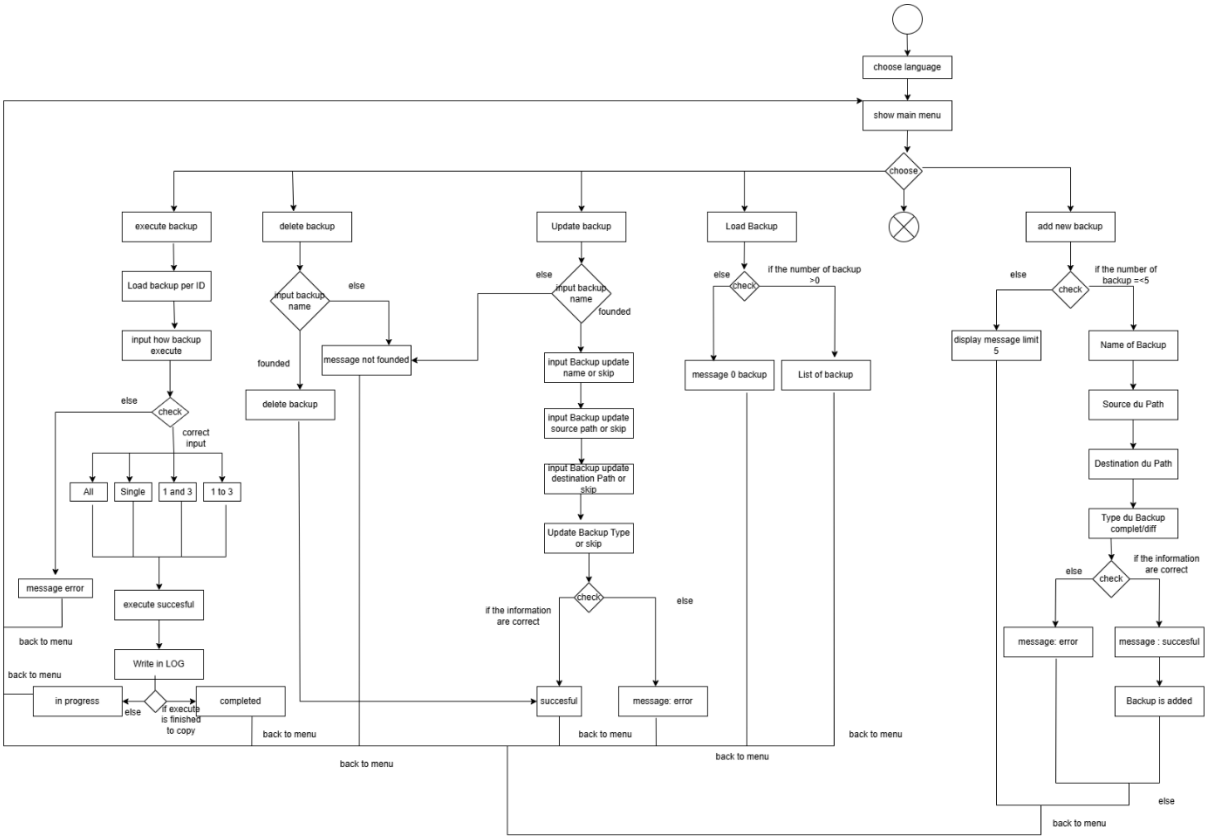
Une vérification est ensuite faite. Si les données sont correctes, la mise à jour est enregistrée ; sinon, un message d’erreur apparaît.

6. Exécution d’une sauvegarde

L’utilisateur sélectionne une sauvegarde à exécuter via son ID. Il choisit ensuite le mode d’exécution parmi plusieurs options (exécuter toutes, une seule, certaines, etc.). Après validation, la sauvegarde s’exécute et son avancement est affiché (en cours, puis terminée). Un enregistrement est également effectué dans un journal (LOG).

7. Retour au menu

Après chaque action (réussie ou non), le système retourne automatiquement au menu principal pour permettre une nouvelle interaction.



2.3 Diagramme de classe :

Le diagramme de classes ci-dessus présente l'**architecture logicielle du projet EasySave** selon une approche orientée objet en C#. Il met en évidence les différentes classes, leurs attributs, méthodes, ainsi que les relations qui les lient dans le cadre de la version 1.0 du logiciel.

1. Architecture générale (MVC)

Le modèle suit une structure **MVC (Model - View - Controller)** :

- **Model** : contient la logique métier (sauvegarde, traçabilité, gestion des fichiers…)
- **View** : assure l'interaction avec l'utilisateur (saisie, affichage)
- **Controller** : coordonne les actions entre la vue et le modèle

2. Composants principaux

Program
Point d'entrée principal du programme via la méthode Main(), qui initialise le contrôleur.

Controller
Coordonne les actions entre l'utilisateur (View) et la logique métier (Model).

Contient des méthodes pour exécuter, ajouter, supprimer, lister, mettre à jour les sauvegardes.
Contient des références à la classe View et à la classe BackupService (composant métier central).

View
Gère l'affichage des menus, des messages, et la collecte des entrées utilisateur.
Contient de nombreuses méthodes comme ShowMainMenu(), ShowMessage(), ou GetBackupName() pour interagir avec l'utilisateur.

Backup
Représente une sauvegarde avec les attributs Source, Target, Name, et Type.
Constructeur permettant d'instancier un objet de type Backup.

BackupService
Contient toute la **logique métier** liée à la gestion des sauvegardes.
Méthodes : ajout, mise à jour, suppression, exécution, validation, transfert, etc.
Utilise des classes internes comme FileStatsTracker pour gérer les performances ou JsonHelper pour la sérialisation.

FileStatsTracker
Permet de **suivre les statistiques de transfert de fichiers** (vitesse, taille totale, durée).
Contient des méthodes pour initialiser, réinitialiser ou charger des données statistiques.

3. Module de gestion des journaux (LOG)

Logger (interface) et **FileLogger (implémentation)**
Fournissent une couche d'abstraction pour la **journalisation des sauvegardes exécutées**.
Attributs : chemins de logs, format JSON.
Méthodes comme LogTransfer() ou UpdateBackup() permettent d'enregistrer les informations sur les transferts réalisés.

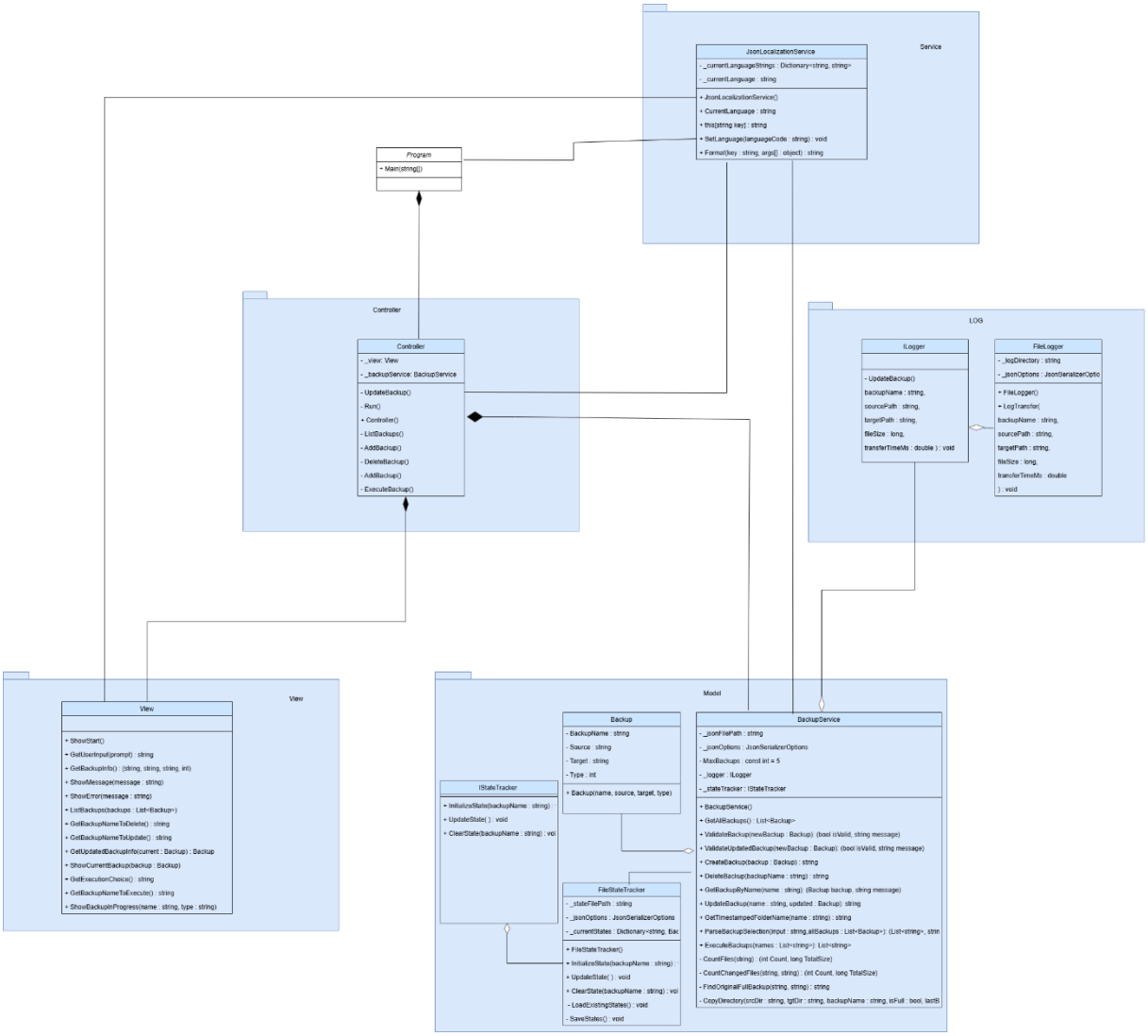
4. Service de localisation

JsonLocalizationService
Gère la **traduction des messages** en fonction de la langue choisie.
Méthodes pour charger, formater et changer la langue d'affichage.
Utilise un dictionnaire de traductions (Dictionary<string, string>).

5. Relations entre les classes

- **Agrégation** entre Controller et BackupService, et entre BackupService et FileStatsTracker
- **Composition** entre BackupService et Backup, illustrant une forte dépendance entre ces éléments

- **Association** simple entre les classes Controller, View, Logger et les autres composants métier



2.4 Diagramme Séquence :

2.4.1 Diagramme Séquence **Add Backup :**

Ce diagramme de séquence illustre les interactions entre les différentes couches du logiciel EasySave lors de l'ajout d'une nouvelle tâche de sauvegarde. Il met en évidence l'enchaînement des messages échangés entre l'utilisateur, la vue (interface utilisateur), le contrôleur, et le modèle (logique métier), dans un contexte où l'utilisateur souhaite créer une nouvelle sauvegarde.

Participants

- **User** : L'utilisateur final du logiciel.
- **View** : L'interface utilisateur en charge de l'affichage et de la saisie des informations.
- **Controller** : Le coordinateur principal entre l'interface et la logique métier.
- **Model** : La couche métier qui gère les données et la validation.

Scénario principal

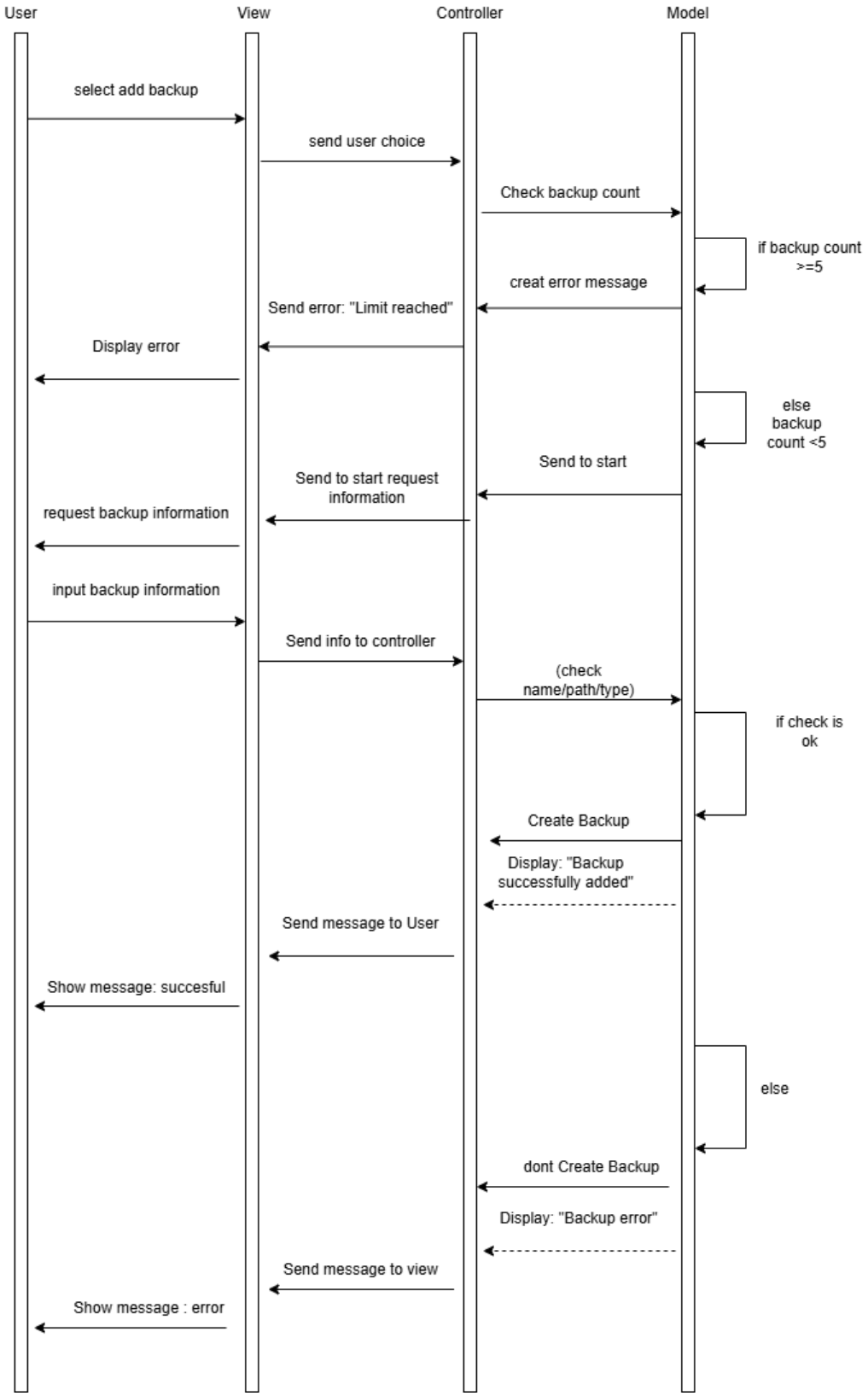
1. L'utilisateur sélectionne l'option **"Ajouter un backup"** depuis l'interface.
2. La **vue** transmet ce choix au **contrôleur**.
3. Le **contrôleur** interroge le **modèle** pour vérifier le nombre actuel de sauvegardes enregistrées.
4. Si la limite de 5 sauvegardes est atteinte, le modèle retourne une erreur. Celle-ci est transmise à la vue, qui l'affiche à l'utilisateur.
5. Si le nombre est inférieur à 5, le contrôleur demande à la vue de collecter les informations nécessaires (nom, chemin source, chemin de destination, type de sauvegarde).
6. L'utilisateur saisit ces données, qui sont ensuite envoyées au contrôleur.
7. Le **contrôleur** transmet les informations au **modèle** pour validation (nom, chemin, type).
8. Si toutes les vérifications sont correctes :
    - Le **modèle** crée la nouvelle sauvegarde.
    - Il retourne un message de succès : **"Backup successfully added"**.
    - Le message est transmis à la vue puis affiché à l'utilisateur.
9. En cas d'erreur de validation (nom déjà utilisé, chemin invalide, etc.), un message d'erreur est généré par le modèle et affiché à l'utilisateur.

Conditions traitées

- Nombre de sauvegardes ≥ 5 → erreur « limite atteinte ».
- Vérification des champs (nom, chemin, type) → réussite ou échec de la création.

Objectif

Ce diagramme démontre l'importance de la coordination entre les composants du système pour garantir que seules les sauvegardes valides soient ajoutées, tout en offrant un retour immédiat à l'utilisateur.


User
View
Controller
Model
select add backup
send user choice
Check backup count
if backup count >=5
creat error message
Send error: "Limit reached"
Display error
else backup count <5
Send to start
Send to start request information
request backup information
input backup information
Send info to controller
(check name/path/type)
if check is ok
Create Backup
Display: "Backup successfully added"
Send message to User
Show message: succesful
else
dont Create Backup
Display: "Backup error"
Send message to view
Show message : error

2.4.2Diagramme Séquence **Update Backup** :

Ce diagramme de séquence décrit le processus d'interaction entre l'utilisateur, l'interface (View), le contrôleur (Controller) et la logique métier (Model) lors de la **mise à jour d'une sauvegarde existante** dans le logiciel EasySave.

Participants

- **User** : L'utilisateur qui souhaite modifier une sauvegarde.
- **View** : Interface de saisie et d'affichage.
- **Controller** : Coordonne les actions entre la vue et le modèle.
- **Model** : Gère les vérifications et la modification des sauvegardes.

Scénario principal

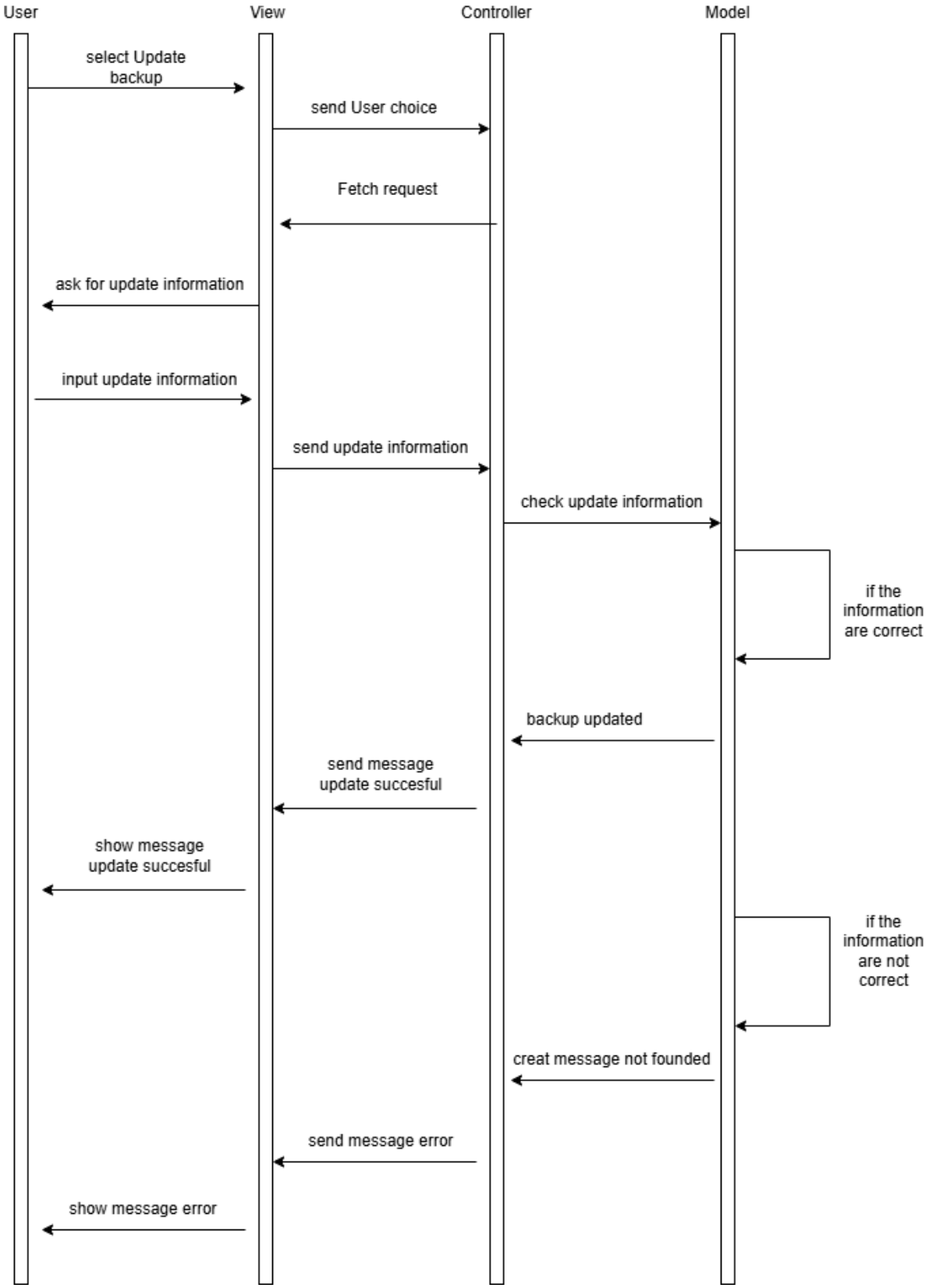
1. L'utilisateur sélectionne l'option **"Update backup"** via le menu principal.
2. La **vue** envoie ce choix au **contrôleur**, qui répond par une requête de récupération des informations à modifier.
3. La **vue** demande alors à l'utilisateur les nouvelles données à saisir : nom, chemin source, chemin cible, type.
4. Une fois les informations saisies, elles sont transmises par la vue au contrôleur.
5. Le **contrôleur** les transmet à la **logique métier**, qui effectue une **vérification de cohérence** (existence de la sauvegarde, validité des chemins, type reconnu).
6. Si les données sont valides :
    - La sauvegarde est mise à jour.
    - Le modèle retourne un message de succès : **"Update successful"**.
    - Ce message est transmis et affiché à l'utilisateur.
7. Si une erreur est détectée (sauvegarde introuvable, données invalides, etc.) :
    - Le modèle génère un message d'erreur.
    - Ce message est transmis via le contrôleur à la vue, puis affiché à l'utilisateur.

Conditions traitées

- Informations correctes → sauvegarde mise à jour avec succès.
- Informations incorrectes → message d'erreur retourné à l'utilisateur.

Objectif

Ce diagramme montre comment le système garantit la cohérence des données lors de la mise à jour d'une tâche de sauvegarde. Il met aussi en évidence le **rôle central du contrôleur** dans la transmission des données et la **validation métier** assurée par le modèle.


User
View
Controller
Model
select Update backup
send User choice
Fetch request
ask for update information
input update information
send update information
check update information
if the information are correct
backup updated
send message update succesful
show message update succesful
if the information are not correct
creat message not founded
send message error
show message error

2.4.3 Diagramme Séquence **Delete Backup** :

Ce diagramme de séquence décrit les échanges entre les composants du système lors de la **suppression d'une tâche de sauvegarde**. Il illustre les étapes que le système suit pour supprimer une sauvegarde existante, en partant du choix utilisateur jusqu'à la confirmation ou l'échec de l'action.

Participants

- **User** : L'utilisateur qui initie la suppression.
- **View** : L'interface graphique ou console affichant les options et messages.
- **Controller** : Reçoit les actions utilisateur et interagit avec la logique métier.
- **Model** : Effectue les vérifications et les suppressions effectives.

Scénario principal

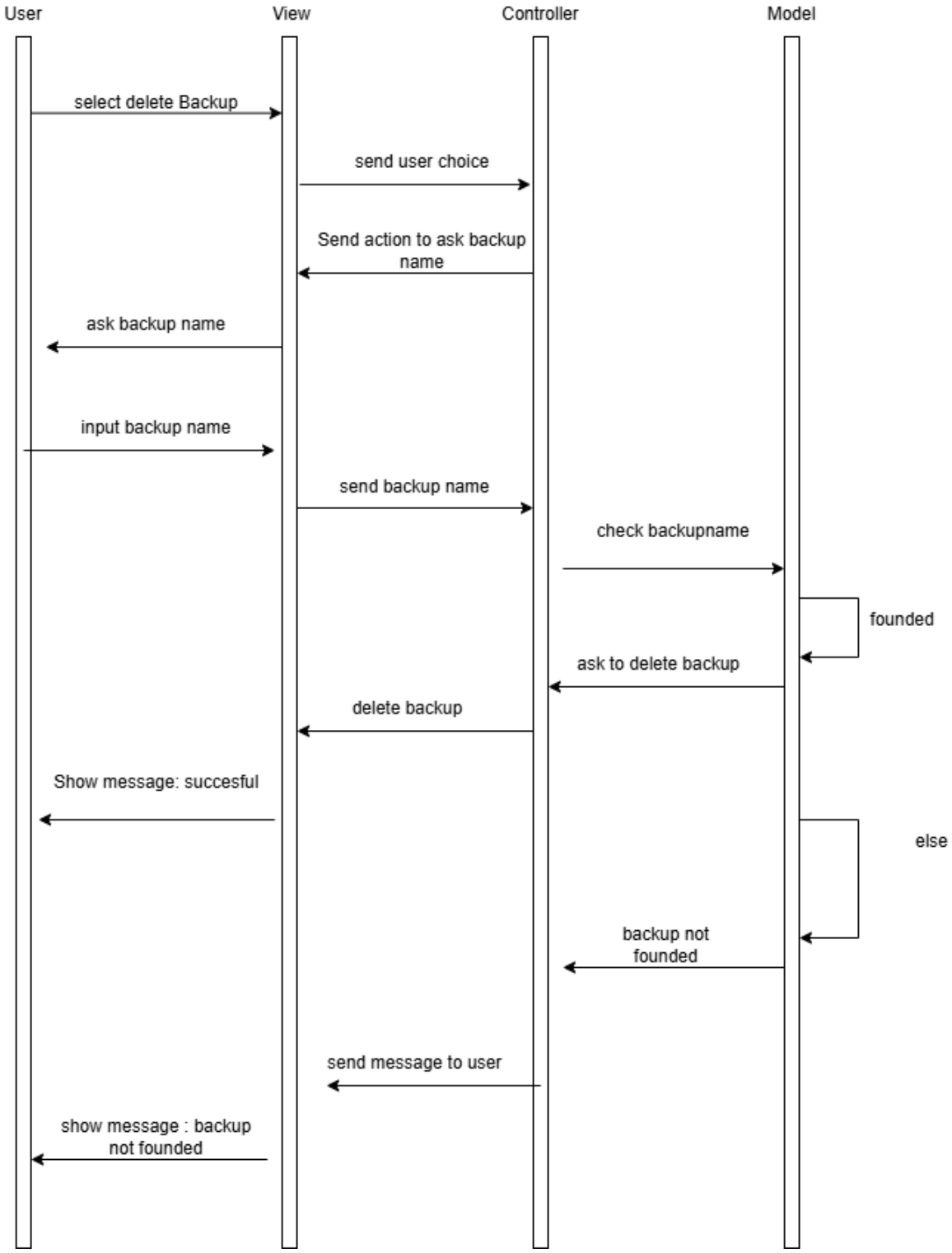
1. L'utilisateur sélectionne l'option **"delete Backup"**.
2. La **vue** transmet ce choix au **contrôleur**.
3. Le contrôleur envoie une action à la vue pour demander le **nom de la sauvegarde** à supprimer.
4. L'utilisateur saisit ce nom.
5. La vue envoie le nom saisi au contrôleur.
6. Le contrôleur transmet cette information au **modèle**, qui vérifie si une sauvegarde avec ce nom existe.
7. Si la sauvegarde est trouvée :
    - Le modèle procède à la suppression.
    - Un message de succès est retourné et affiché à l'utilisateur.
8. Si aucune sauvegarde ne correspond au nom fourni :
    - Le modèle retourne un message d'erreur.
    - Ce message est transmis via le contrôleur et affiché à l'utilisateur.

Conditions traitées

- Nom valide → sauvegarde supprimée avec message : **"successful"**.
- Nom introuvable → message d'erreur : **"backup not found"**.

Objectif

Ce diagramme permet de visualiser les interactions et vérifications nécessaires avant de supprimer une sauvegarde. Il met en valeur la rigueur du système dans la gestion des erreurs et la traçabilité des actions.



2.4.4 Diagramme Séquence **Load Backup** :

Ce diagramme de séquence illustre le processus de **chargement des sauvegardes existantes** dans le système EasySave, permettant à l'utilisateur d'accéder à la liste des sauvegardes déjà créées.

Participants

- **User** : L'utilisateur qui souhaite consulter les sauvegardes.
- **View** : Interface d'interaction avec l'utilisateur.
- **Controller** : Gère les échanges entre l'interface et la logique métier.
- **Model** : Contient les données métiers et effectue les vérifications.

Scénario principal

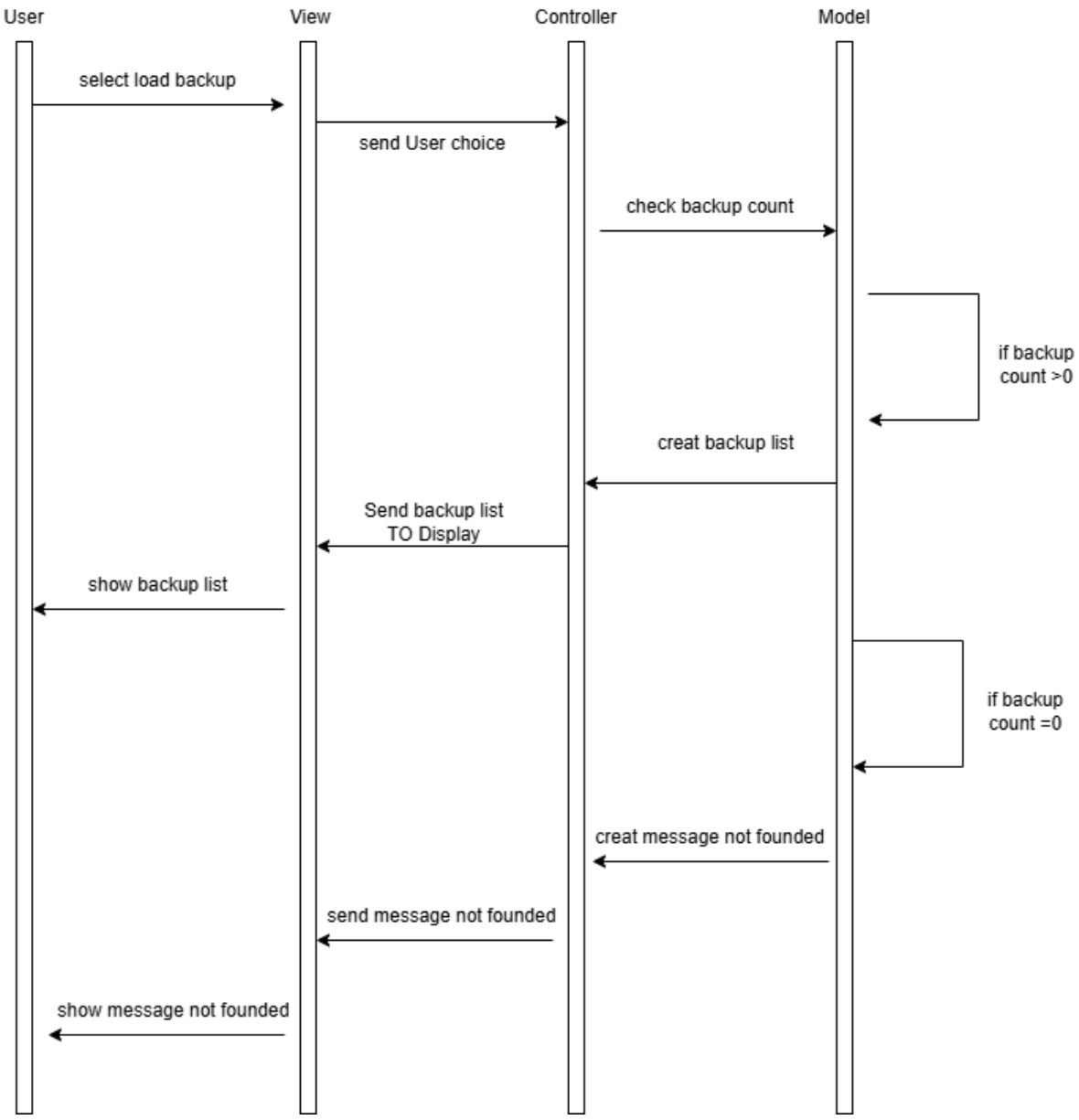
1. L'utilisateur sélectionne l'option **"load backup"** depuis l'interface.
2. La **vue** transmet le choix au **contrôleur**.
3. Le contrôleur appelle la fonction de **vérification du nombre de sauvegardes** via le **modèle**.
4. Deux cas de figure se présentent :
    - Si au moins une sauvegarde est présente :
        - Le modèle génère la **liste des sauvegardes**.
        - Cette liste est retournée au contrôleur, puis envoyée à la vue.
        - La vue l'affiche à l'utilisateur.
    - Si aucune sauvegarde n'existe :
        - Le modèle génère un **message d'erreur** ("aucune sauvegarde trouvée").
        - Ce message est transmis à la vue et affiché à l'utilisateur.

Conditions traitées

- Nombre de sauvegardes > 0 → affichage de la liste.
- Nombre de sauvegardes = 0 → message **"not found"** affiché.

Objectif

Ce diagramme souligne la simplicité du processus de consultation des sauvegardes, tout en garantissant une gestion claire des cas d'absence de données. Il illustre la logique de vérification appliquée à chaque interaction de lecture dans le système.



2.4.5 Diagramme Séquence **Execute Backup :**

Ce diagramme de séquence décrit le déroulement complet du processus d'**exécution d'une tâche de sauvegarde** dans EasySave. Il montre les interactions nécessaires entre l'utilisateur, l'interface (vue), le contrôleur, et la logique métier pour valider et exécuter une opération de sauvegarde.

Participants

- **User** : L'utilisateur déclenchant l'opération.
- **View** : Interface graphique ou console affichant les informations et collectant les choix.
- **Controller** : Fait le lien entre les interactions utilisateur et les services métiers.
- **Model** : Gère la logique de validation, d'exécution de la sauvegarde et la journalisation.

Scénario principal

1. L'utilisateur sélectionne **"execute backup"** dans le menu.
2. La **vue** envoie ce choix au **contrôleur**.
3. Le **contrôleur** demande au modèle de **charger les sauvegardes disponibles**.
4. La vue affiche à l'utilisateur la liste des identifiants de sauvegardes existants.
5. L'utilisateur saisit le **type d'exécution** souhaité (exécuter toutes, une seule, ou plusieurs sauvegardes).
6. La vue envoie ces informations au contrôleur, qui les transmet au modèle.
7. Le **modèle** vérifie la validité des entrées (ID de sauvegarde, type...).
8. Si les informations sont correctes :
    - La sauvegarde est exécutée.
    - Une ligne est ajoutée dans le fichier **LOG** (journalisation).
    - Un message de confirmation est retourné à l'utilisateur.
9. Si les informations sont incorrectes (ID inexistant, type invalide, etc.) :

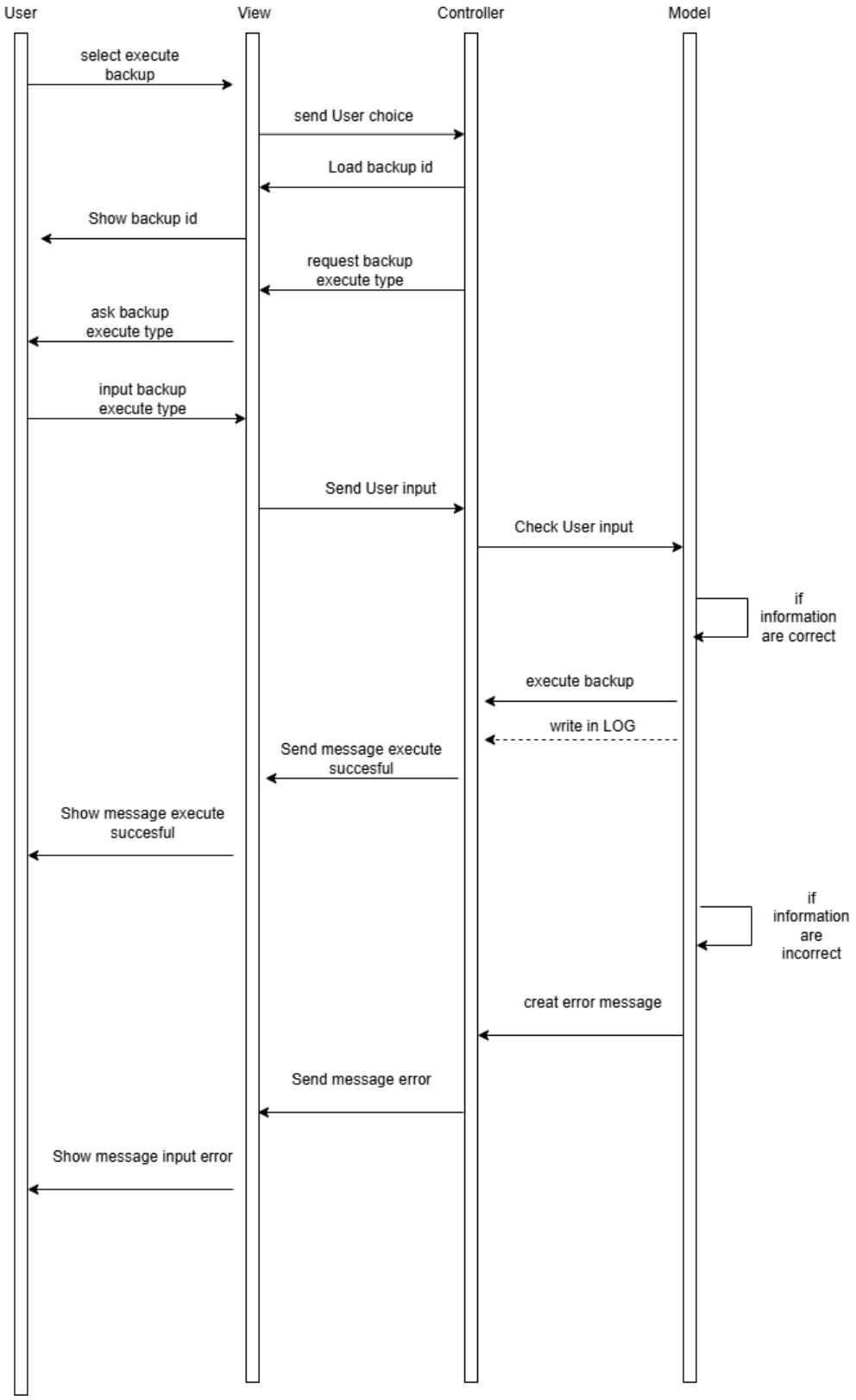
- Le modèle génère un **message d'erreur**.
- Celui-ci est transmis à la vue et affiché à l'utilisateur.

Conditions traitées

- Informations valides → sauvegarde lancée + message de succès.
- Informations invalides → message d'erreur : **"input error"**.

Objectif

Ce diagramme assure une compréhension claire du processus d'exécution d'une tâche de sauvegarde avec journalisation, tout en intégrant la gestion d'erreur. Il met en avant la coordination entre validation métier et retour utilisateur.


User
View
Controller
Model
select execute backup
send User choice
Load backup id
Show backup id
request backup execute type
ask backup execute type
input backup execute type
Send User input
Check User input
if information are correct
execute backup
write in LOG
Send message execute succesful
Show message execute succesful
if information are incorrect
creat error message
Send message error
Show message input error

3.GitHub :

GitHub est une plateforme en ligne de gestion de code source, basée sur le système de contrôle de version Git. Elle permet à des développeurs du monde entier de collaborer sur des projets logiciels, de manière centralisée, transparente et traçable.

3.1Fonctionnalités principales :

- **Hébergement de dépôts Git** (publics ou privés),
- **Suivi des modifications sur le code** (historique des commits),
- **Collaboration via les branches, pull requests et merges,**
- **Gestion des bugs, des tâches et des discussions** avec les issues et les projets.

3.2Avantages de GitHub :

- **Travail collaboratif facilité** : plusieurs développeurs peuvent travailler sur les mêmes fichiers sans conflit.
- **Traçabilité des changements** : chaque modification est datée, identifiée, et réversible.
- **Sauvegarde sécurisée** : le code est conservé en ligne et synchronisé localement.
- **Outils intégrés** : intégration continue, déploiement, gestion de tickets, documentation, etc.
- **Communauté et open source** : des millions de projets accessibles et partageables.

GitHub s'est imposé comme **l'outil standard de collaboration et de versioning dans le développement logiciel**, aussi bien dans les projets open source que dans les environnements professionnels.

3.3Pourquoi avons-nous choisi GitHub pour notre projet EasySave ?

Dans le cadre du développement du logiciel **EasySave**, GitHub a été retenu comme **plateforme de travail collaboratif** pour les raisons suivantes :

1. Travail d'équipe à distance facilité

Chaque membre de l'équipe a pu accéder au code source à tout moment, effectuer des modifications en local, puis les synchroniser avec le reste de l'équipe via des **pull requests**.

2. Structuration du développement

Grâce à la **gestion des branches**, nous avons pu séparer les fonctionnalités en cours de développement (ex. : ajout de la fonction d'exécution, création du fichier log...) sans perturber la version stable du projet.

3. Traçabilité et gestion des erreurs

GitHub nous a permis de suivre précisément qui a modifié quoi, à quel moment, et pourquoi, ce qui est essentiel pour la **maintenance** et la **compréhension du code**.

4. Centralisation et sauvegarde

Le dépôt GitHub a servi de **référence centrale** et de **sauvegarde** en ligne, réduisant le risque de perte de données ou de conflit de version.

GitHub s'est donc révélé être un outil stratégique dans le bon déroulement de notre projet, en alliant **collaboration, organisation et efficacité**.

3.4Comment avons-nous utilisé GitHub dans notre projet EasySave ?

Lors du projet EasySave, GitHub a été utilisé à chaque étape du développement :

- Initialisation :

- Création d'un dépôt privé nommé EasySave.
- Ajout d'un fichier README.md décrivant le projet.
- Ajout d'un fichier .gitignore pour ignorer les dossiers inutiles lors du push.
- Partage des accès avec les membres de l'équipe et le tuteur.

- Branches :

- Une branche principale : Final (code stable).
- Des branches secondaires lors de chaque push : UML, UMLL, master etc…

- Commits et push réguliers :

- Les changements étaient poussés quotidiennement pour éviter les conflits.

- Pull Requests & Merges :

- Lorsqu'une fonctionnalité était finalisée, une pull request était ouverte.
- Le code était relu puis fusionné dans Final après validation.

- Gestion des bugs et des tâches :

- Des issues ont été créées pour suivre les bugs détectés ou les fonctionnalités à implémenter.
- Certaines ont été liées aux pull requests.

3.5Procédure d'utilisation de GitHub :

1. Se connecter au dépôt :
git remote add origin https://github.com/nom-utilisateur/nom-projet.git

2. Ajouter/modifier un fichier :
git add .
ou
git add nomFichier.cs

3. Sauvegarder les modifications localement :
git commit -m "message clair de la modification"

4. Envoyer les modifications sur GitHub :
git push origin nom-de-la-branche

5. Récupérer les dernières modifications :
git pull origin nom-de-la-branche

6. Créer une nouvelle branche :
git checkout -b nom-branche

7. Créer une pull request (depuis GitHub) :
- Aller sur la page du dépôt,
- Cliquer sur "Pull requests" > "New pull request",
- Choisir la branche source et la branche cible (main),
- Ajouter une description claire et soumettre.

4.Code source de l'application

4.1Structure MVC du Projet EasySave

Le projet EasySave suit une architecture Modèle-Vue-Contrôleur (MVC), une approche claire et organisée qui sépare la logique métier, l'affichage et la gestion des interactions.

1. Modèle (Model)

Le Modèle représente les données et la logique métier de l'application.

Dans EasySave, il inclut :

- Backup : Décrit une tâche de sauvegarde (nom, source, cible, type).
- BackupService : Gère la logique des sauvegardes (création, suppression, exécution).
- IStateTracker et FileStateTracker : Suivent l'état des sauvegardes en temps réel.

Les modèles sont indépendants de l'interface utilisateur, ce qui permet une maintenance et une évolution simplifiées.

2. Vue (View)

La **Vue** (qui pourrait être une interface console ou graphique) affiche les informations à l'utilisateur via des méthodes définis, comme :

- Le Menu principal
- La liste des sauvegardes configurées.
- Les résultats des opérations (succès/échec).

Dans une application console, la vue se résume à des Console.WriteLine(), tandis qu'une interface graphique utiliserait des fenêtres ou des widgets.

3. Contrôleur (Controller)

Le Contrôleur fait le lien entre la Vue et le Modèle. Bien que non formalisé dans une classe dédiée ici, sa logique est intégrée dans :

- Controller => View : récupérer les choix utilisateur et envoyer le message généré par le modèle au view
- Controller => BackupService : envoyer les choix utilisateur au modèle puis récupérer le message généré par ce dérnier.

4.2Implémentation des Fonctionnalités Générales

1. **Gestion des Sauvegardes**
    - L'utilisateur peut créer, modifier, supprimer, Lister et lancer des sauvegardes.
    - Deux types de sauvegardes sont pris en charge : complète (copie intégrale) et différentielle (copie des fichiers modifiés depuis la dernière sauvegarde).

2. **Journalisation (Logs)**
    - La journalisation a été implémenté comme demander dans le cahier des charges, ou une DLL (Dynamic link library) nommé Easysave.Logging a été créer pour gérer la logique des fichiers LOG, puis on a appelé cette DLL dans notre modèle BackupService.
    - Chaque opération de copie est enregistrée dans un fichier JSON daté (YYYY-MM-DD.json), avec des détails comme la taille du fichier et le temps de transfert.

3. **Suivi en Temps Réel**
    - Le fichier state.json capture l'état des sauvegardes (actives, terminées, échouées) et la progression (fichiers restants, taille transférée).
4. **Validation et Sécurité**
    - Vérification des chemins d'accès avant toute opération.
    - Limitation du nombre de sauvegardes (5 maximum).

4.3Avantages de l'Architecture MVC

- Modularité : Le code est découpé en composants réutilisables (ex. FileLogger peut être remplacé sans toucher au reste).
- Maintenance facilitée : Les modifications de l'interface (console → GUI) n'affectent pas la logique métier.
- Testabilité : Les modèles et services peuvent être testés unitairement.

Cette structure permet à EasySave d'être scalable (ajout facile de nouvelles fonctionnalités) et robuste (gestion centralisée des erreurs).

5.Conclusion

Le projet EasySave représente une solution robuste et modulaire pour la gestion de sauvegardes, alliant simplicité d'utilisation et puissance fonctionnelle. Grâce à son architecture MVC (Modèle-Vue-Contrôleur), le code est clairement structuré, séparant la logique métier, l'affichage et la gestion des interactions, ce qui facilite la maintenance et les futures évolutions.