REPUBLIQUE DU CAMEROUN
*****
Paix-Travail-Patrie
*****
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
*****
Université de Yaoundé I
*****
Institut Saint Jean

REPUBLIC OF CAMEROON
*****
Peace-Work-Fatherland
*****
MINISTRY OF HIGHER EDUCATION
*****
University of Yaounde I
*****
Institut Saint Jean

Cahier de conception

Conception et mise en œuvre d'un gestionnaire de fichiers multi-utilisateurs: cas de Pop!_OS

Rédigé et présenté par :

TEMATIO Zixfiline powera
PAYONG Brice valery

Année Académique

2025-2026

ETUDE DU PROJET

Nom du projet :	Conception et mise en œuvre d'un gestionnaire de fichiers multi-utilisateurs: cas de Pop!_OS
Version du document :	
Date de création :	14/01/2026
Dernière mise à jour :	23/01/2026
Responsable projet :	• PAYONG Brice Valery • TEMATIO ZEBAZE Zixfiline Powera
Validé par :	*M. NGUIMBUS Emmanuel*

SOMMAIRE

LISTE DES FIGURES

LISTES DES TABLEAUX

INTRODUCTION

Le cahier de conception constitue la suite logique du cahier d'analyse. Après l'identification et la formalisation des besoins fonctionnels et non fonctionnels, ce document a pour objectif de décrire la **solution technique retenue** pour répondre à ces besoins.

Il présente les choix d'architecture, les différentes composantes du système ainsi que leur organisation à travers des **diagrammes UML**, afin de fournir une vision claire et structurée du fonctionnement interne du gestionnaire de fichiers multi-utilisateurs sous Pop!_OS.

I. CONTEXTE DU PROJET ET OBJECTIFS

1) Contexte du projet

Dans un environnement Linux multi-utilisateurs tel que Pop!_OS, la gestion sécurisée des fichiers repose sur un ensemble de règles strictes liées aux permissions, aux utilisateurs et aux groupes. La conception d'un gestionnaire de fichiers nécessite donc une architecture logicielle capable de respecter ces contraintes tout en offrant une interface simple et efficace.

2) Objectif du cahier de conception

- L'objectif de ce document est de :
- Définir l'architecture globale du système
- Décrire les interactions entre les différents composants
- Modéliser le fonctionnement interne de l'application
- Servir de référence pour la phase d'implémentation

II. Architecture de la Solution

1. Type d'architecture retenu

L'application adopte une **architecture logique à deux couches (Two-Tier)**, déployée localement sur le système d'exploitation Pop!_OS.
Toutes les composantes du système sont exécutées sur une seule machine, au sein d'un même

programme, mais organisées en deux couches distinctes afin d'assurer une bonne séparation des responsabilités :

1. **Couche Présentation (Interface Utilisateur)**
2. **Couche Logique Métier intégrant l'accès au système de fichiers**

Cette organisation permet de distinguer clairement l'interaction avec l'utilisateur de la gestion des règles de sécurité et des opérations sur les fichiers.

3. Description des couches

3.1 Couche Présentation (Interface Utilisateur)

Rôle :
Assurer l'interaction entre l'utilisateur et l'application.

Fonctionnalités :

- Affichage de l'arborescence des fichiers et dossiers
- Saisie des commandes (création, suppression, copie, déplacement)
- Modification des permissions (lecture, écriture, exécution)
- Visualisation des propriétaires et groupes
- Affichage des messages d'erreur et de confirmation

Responsabilités :

- Capturer les actions de l'utilisateur
- Valider les entrées
- Transmettre les requêtes à la couche logique
- Présenter les résultats de façon claire

3.2 Couche Logique Métier

Rôle :
Constituer le cœur fonctionnel de l'application en appliquant les règles de sécurité et en contrôlant l'accès au système de fichiers.

Fonctionnalités principales :

- Gestion des utilisateurs (UID) et des groupes (GID)
- Vérification des permissions Unix (rwx, propriétaire, groupe, autres)
- Contrôle d'accès avant toute opération
- Application des politiques de sécurité
- Orchestration des opérations sur les fichiers et dossiers

Accès au système :
Cette couche assure également l'interfaçage direct avec le noyau Linux via :

- Appels système POSIX (stat, chmod, chown, open, unlink, etc.)
- Lecture des métadonnées des fichiers
- Interaction avec les fichiers système (/etc/passwd, /etc/group)

4. Justification du choix architectural

Tableau 1: Justification du choix architectural

Critère	Justification
Simplicité	Architecture adaptée à une application locale
Performance	Accès direct au système sans couche intermédiaire
Sécurité	Application stricte du modèle Unix multi-utilisateurs
Maintenabilité	Séparation claire entre interface et logique
Évolutivité	Possibilité future d'évolution vers une architecture 3-tiers

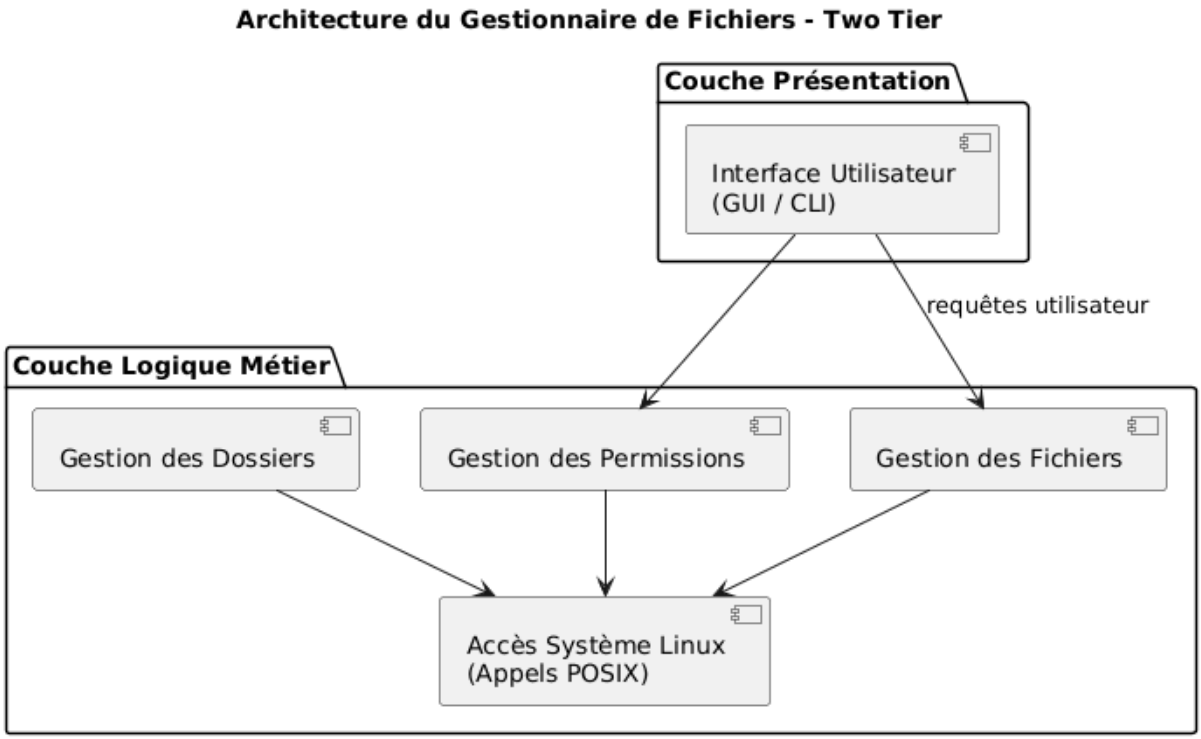

Figure 1: Architecture de la solution

III. DIAGRAMME DE CLASSE

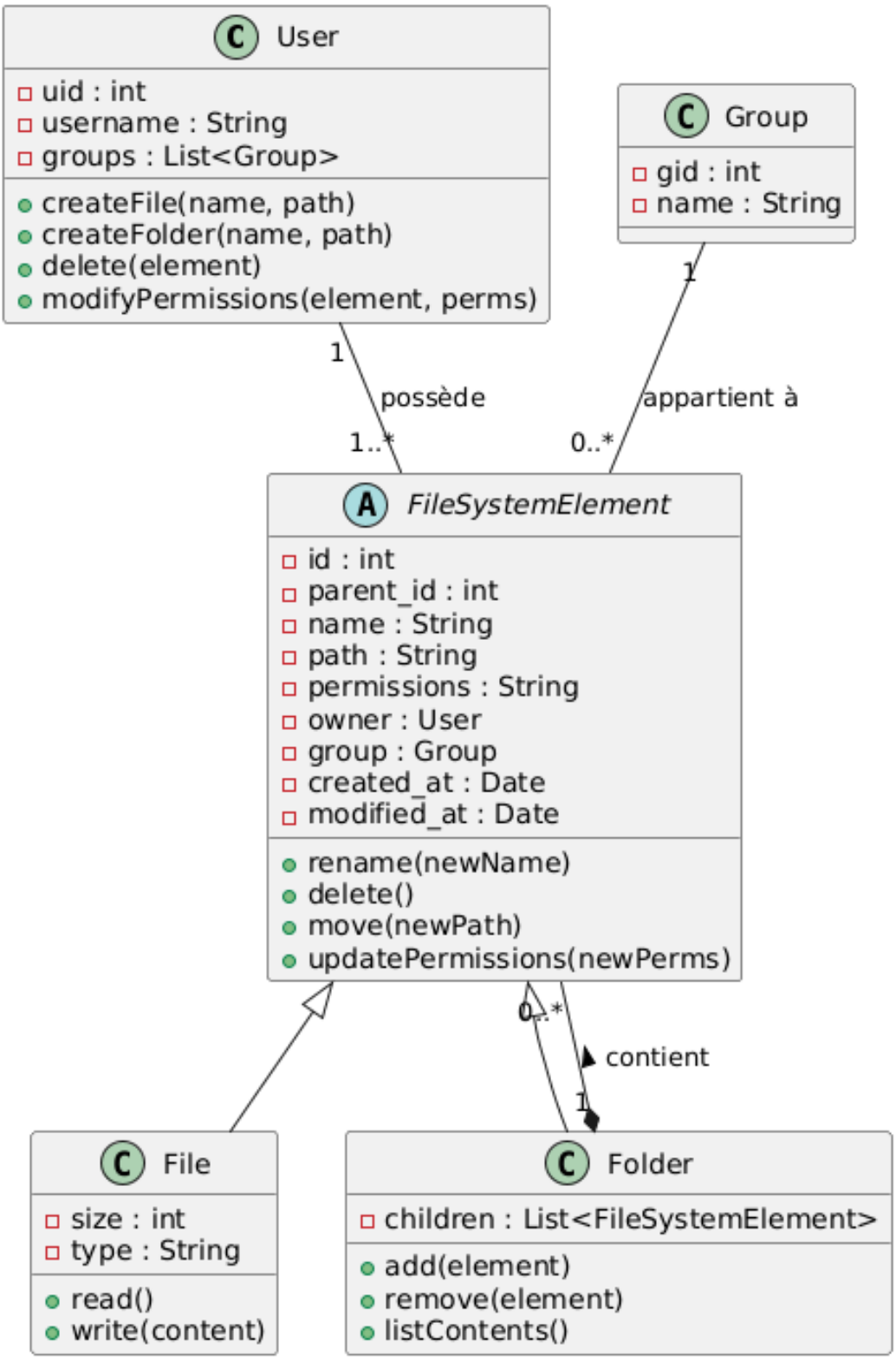

Figure 2: Diagramme de classe

IV.

Description du Diagramme de Classe : Système de Gestion de Fichiers

Le diagramme de classe présente l'organisation structurelle de l'application, modélisant les relations entre les utilisateurs, les groupes système et les éléments du système de fichiers (fichiers et dossiers).

1. Les Entités Principales

- **User (Utilisateur) :** Représente un utilisateur du système Pop!_OS. Chaque utilisateur est identifié par un uid (User ID) et possède un nom d'utilisateur ainsi qu'une liste de groupes auxquels il appartient. Il est l'initiateur des actions de création, suppression et modification des permissions.
- **Group (Groupe) :** Représente un groupe de sécurité Linux identifié par un gid (Group ID). Il est essentiel pour définir les droits d'accès partagés sur les fichiers et dossiers.
- **FileSystemElement (Classe Abstraite) :** Il s'agit de la classe mère qui définit les propriétés communes à tous les objets stockés sur le disque. Elle contient des attributs critiques comme le name, le path (chemin absolu), les permissions (format rwx), ainsi que les dates de création et de modification. Elle porte également les identifiants de relation id et parent_id.
- **File (Fichier) :** Spécialisation de FileSystemElement. Elle ajoute les attributs de taille (size) et de type. Ses méthodes permettent les opérations de lecture et d'écriture de contenu.
- **Folder (Dossier) :** Spécialisation capable de contenir d'autres éléments. Elle gère la liste de ses enfants et possède des méthodes pour lister, ajouter ou retirer des éléments du répertoire.

2. Relations et Règles de Gestion

- **Possession (Multiplicité 1..*) :** Un utilisateur possède **au moins un ou plusieurs** FileSystemElement. Cette relation (1 vers 1..*) est une règle de gestion stricte garantissant que chaque utilisateur a un espace personnel (Home directory) au minimum. Un élément appartient à un seul propriétaire (owner).
- **Composition (Hiérarchie Folder) :** Il existe une relation de **composition** entre Folder et FileSystemElement. Un dossier est le parent direct de zéro à plusieurs éléments. Le losange plein indique que la durée de vie des éléments dépend de leur dossier parent : si un dossier est supprimé, ses composants internes le sont également.
- **Appartenance au Groupe :** Chaque FileSystemElement est obligatoirement rattaché à un Group unique (multiplicité 1). Cela permet d'appliquer les règles de permissions de groupe du noyau Linux.

- **Héritage :** File et Folder héritent de la classe abstraite, ce qui permet de traiter n'importe quel élément du système de manière générique pour des opérations comme le renommage ou le déplacement.

3. Traçabilité et Navigation

Grâce à l'attribut parent_id présent dans la classe abstraite, le système peut reconstituer l'arborescence complète à partir de n'importe quel fichier. Cette structure facilite la navigation ascendante (vers le parent) et descendante (vers les enfants) dans l'interface graphique de l'application.

V. DIAGRAMME DE SEQUENCES

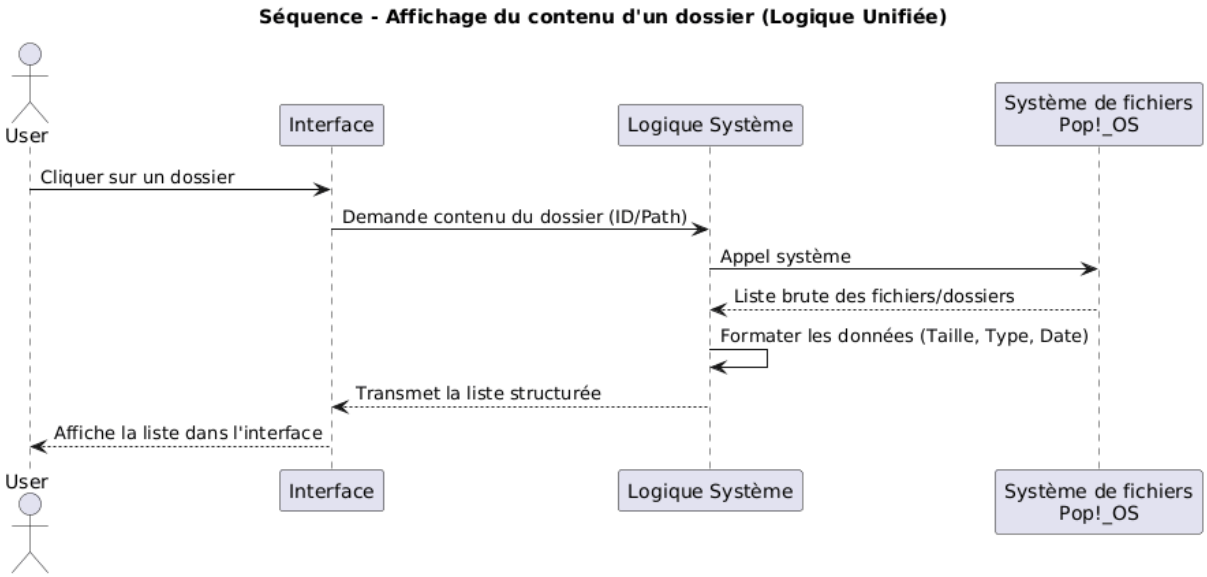

Figure 3:Diagramme de Séquence : Affichage du contenu d'un dossier

Diagramme de Séquence : Affichage du contenu d'un dossier

Ce diagramme illustre le flux de lecture permettant à l'utilisateur de naviguer dans l'arborescence et de visualiser les fichiers présents dans un répertoire.

1. Requête de Navigation

- **Action Utilisateur** : L'utilisateur clique sur un dossier dans l'**Interface**.
- **Transmission** : L'interface envoie une demande de récupération de contenu à la **Logique Système**, en précisant l'identifiant ou le chemin (path) du dossier.

2. Interaction avec le Noyau Pop!_OS

- **Appel Système** : La **Logique Système** effectue un appel direct au **Système de fichiers Pop!_OS** (via des fonctions comme os.listdir) pour obtenir la liste brute des éléments présents sur le disque.
- **Extraction des données** : Le système de fichiers retourne la liste des fichiers et sous-dossiers.

3. Traitement et Rendu

- **Formatage** : La **Logique Système** traite les informations brutes pour les transformer en données structurées, incluant la taille, le type de fichier et les dates de modification.
- **Affichage** : La liste formatée est transmise à l'**Interface**, qui rafraîchit la vue pour présenter le contenu du dossier de manière lisible à l'utilisateur.

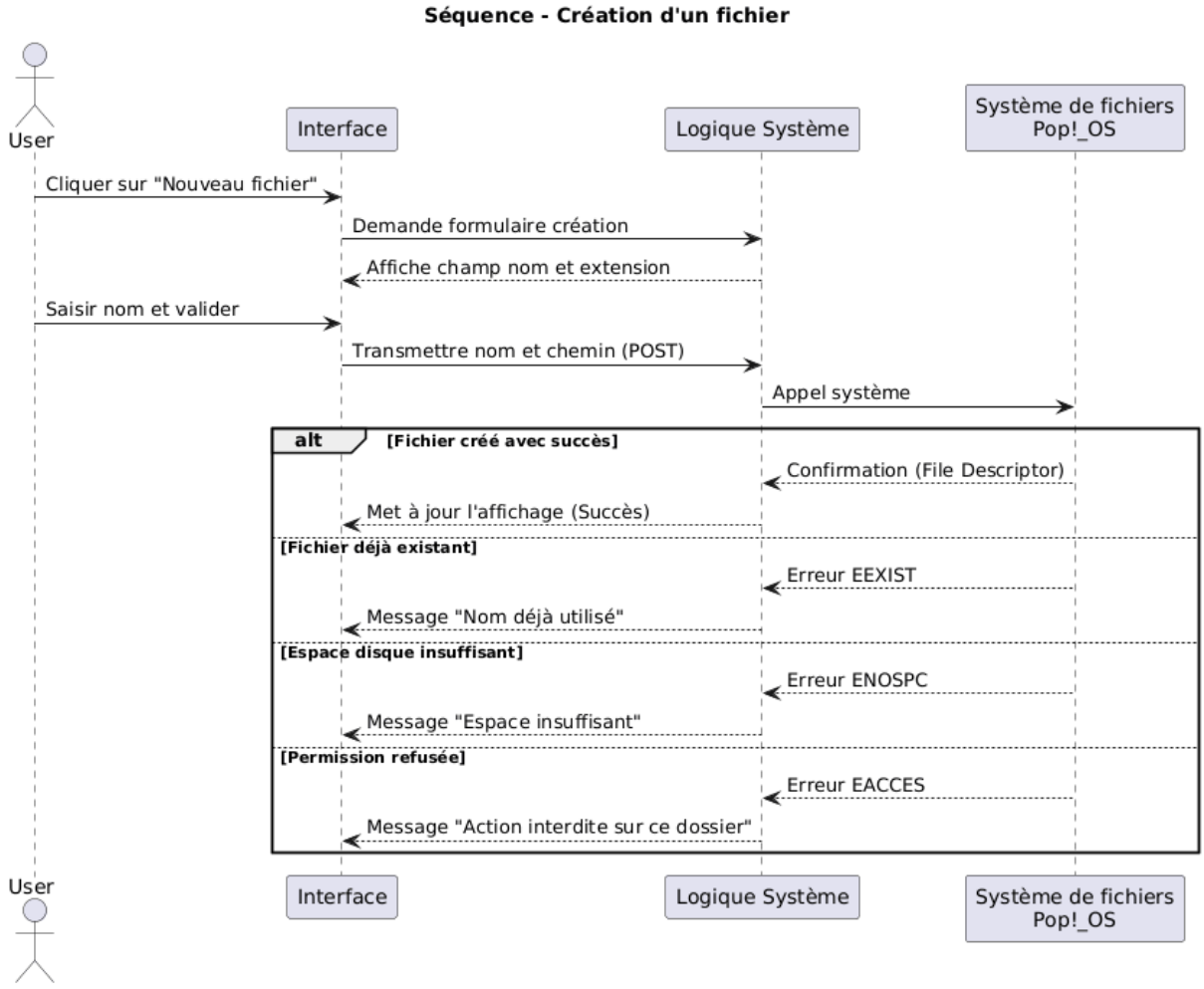

Figure 4:Diagramme de Séquence : Création d'un fichier

Diagramme de Séquence : Création d'un fichier

Ce diagramme détaille le flux de travail permettant à un utilisateur de générer un nouvel élément sur le disque, en passant par les étapes de validation et de retour d'état du système.

1. Interaction Initiale

- **Demande de création** : L'utilisateur clique sur le bouton "Nouveau fichier" dans l'**Interface Graphique**.
- **Saisie des données** : L'interface affiche un formulaire où l'utilisateur saisit le nom du fichier et son extension. Ces informations, accompagnées du chemin de destination, sont transmises à la **Logique Système**.

2. Traitement et Appel Système

- **Validation Métier** : La **Logique Système** vérifie la validité du nom (absence de caractères interdits) et s'assure que l'utilisateur possède les droits d'écriture dans le dossier parent.
- **Exécution sur Pop!_OS** : L'application sollicite le **Système de fichiers** via un appel système (typiquement open avec le flag de création).

3. Scénarios de Sortie (Gestion des erreurs)

- **Succès** : Le système de fichiers confirme la création. La **Logique Système** enregistre l'élément dans la base de données (avec ses métadonnées initiales) et l'**Interface** rafraîchit la vue pour afficher le nouveau fichier.
- **Erreurs fréquentes** :
    - **Fichier déjà existant (EEXIST)** : Si un fichier porte déjà ce nom, la logique métier intercepte l'erreur et affiche un message "Nom déjà utilisé".
    - **Espace disque insuffisant (ENOSPC)** : En cas de saturation du stockage, l'utilisateur est immédiatement averti par une notification dédiée.
    - **Permission refusée (EACCES)** : Si l'emplacement est protégé par le noyau Linux, le système bloque l'action et informe l'utilisateur de ses droits insuffisants.

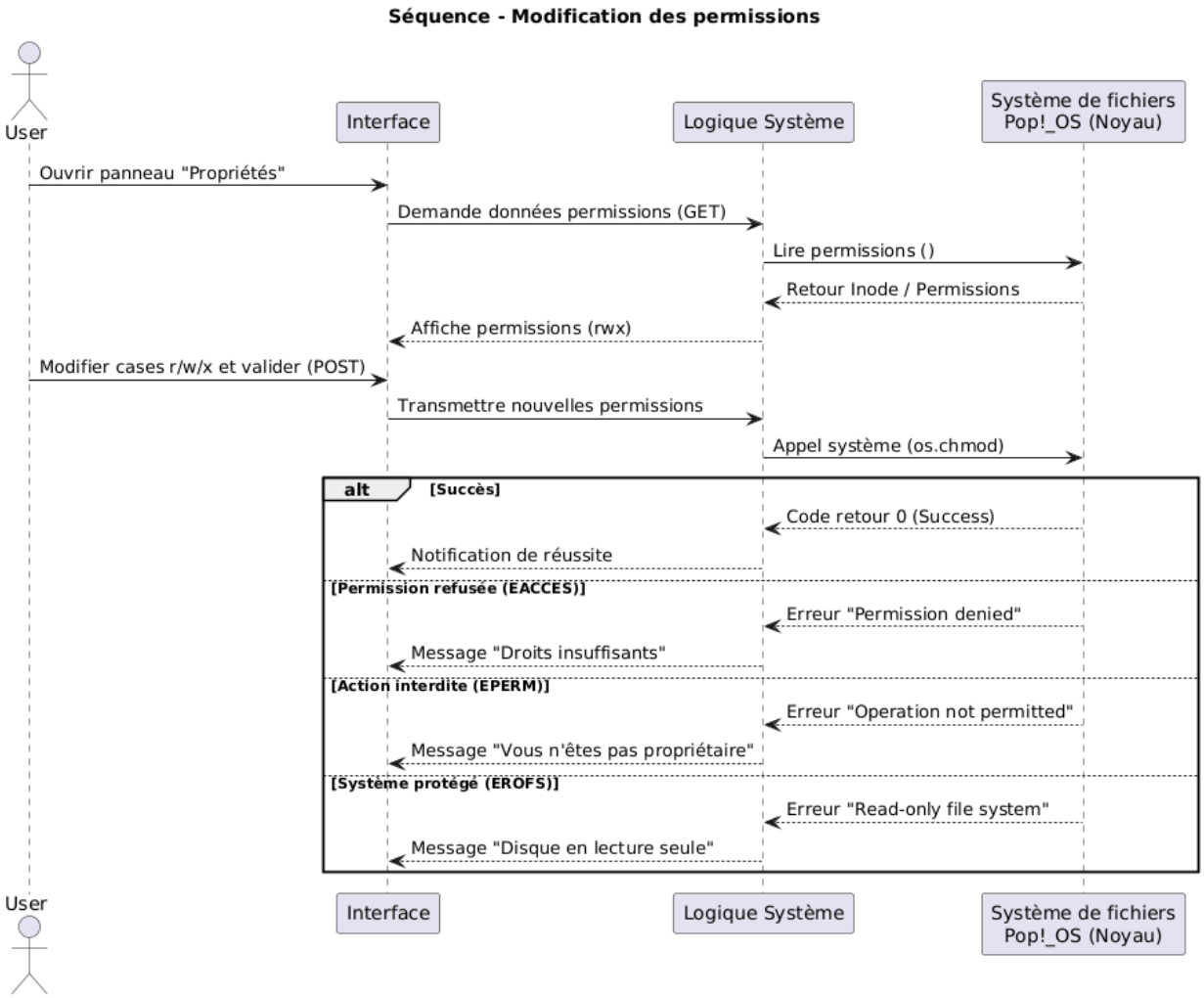

Figure 5:Diagramme de Séquence : Modification des Permissions

Diagramme de Séquence : Modification des Permissions

Ce diagramme illustre comment l'application permet à un utilisateur de modifier les droits d'accès (rwx) d'un élément, traduisant une intention graphique en une commande de sécurité système.

1. Interface et Sélection

- **Action Utilisateur** : L'utilisateur sélectionne un fichier ou un dossier et clique sur "Modifier les permissions".
- **Récupération de l'état** : L'**Interface** demande à la **Logique Système** les permissions actuelles. Celle-ci interroge le **Système de fichiers Pop!_OS** pour afficher les droits existants (Lecture, Écriture, Exécution) pour le Propriétaire, le Groupe et les Autres.

2. Validation et Appel Système

- **Saisie** : L'utilisateur coche ou décoche les permissions souhaitées dans l'interface et valide.

- **Vérification de Propriété** : La **Logique Système** vérifie que l'utilisateur est bien le propriétaire de l'élément (owner) ou possède les privilèges nécessaires avant de poursuivre.
- **Exécution** : L'application effectue l'appel système os.chmod auprès du noyau Linux de Pop!_OS pour appliquer le nouveau mode numérique ou symbolique.

3. Confirmation et Retour

- **Scénario de Succès** : Le système de fichiers confirme le changement. La **Logique Système** met à jour l'attribut permissions dans la base de données SQLite pour maintenir la synchronisation. L'interface affiche une notification "Permissions mises à jour".
- **Gestion des Erreurs** :
    - **Permission refusée (EACCES)** : Si l'utilisateur tente de modifier un fichier système protégé, le noyau Linux renvoie une erreur que l'interface traduit par "Action interdite : droits système insuffisants".

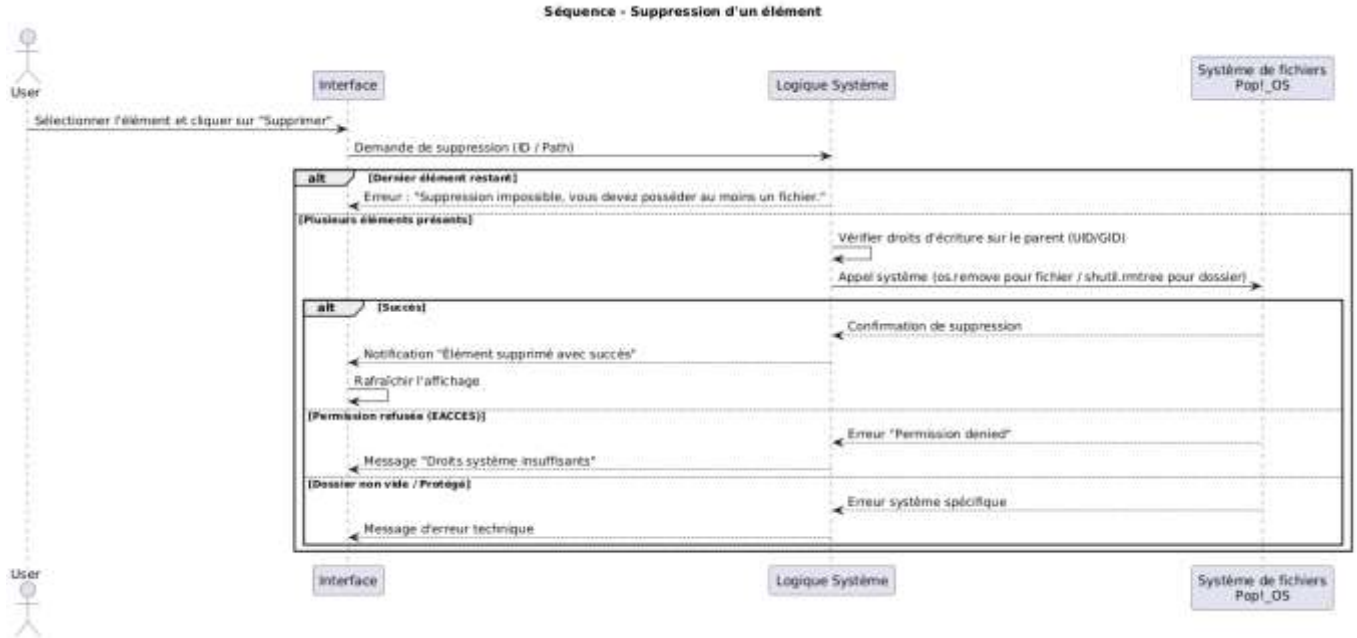

Figure 6:Diagramme de Séquence : Suppression d'un élément

Diagramme de Séquence : Suppression d'un élément

Ce diagramme détaille la procédure de retrait définitif d'un fichier ou d'un dossier, en mettant l'accent sur la sécurité des données et l'intégrité des règles de l'application.

1. Demande et Validation Métier

- **Action Utilisateur** : L'utilisateur sélectionne l'élément et clique sur "Supprimer".

- **Contrôle de la Multiplicité (1..*)** : Avant toute action, la **Logique Système** vérifie si l'élément est le dernier possédé par l'utilisateur. Si c'est le cas, la suppression est bloquée pour garantir que chaque utilisateur possède au moins un fichier (généralement son répertoire racine).

2. Exécution Système

- **Vérification des Droits** : La logique métier vérifie les permissions d'écriture sur le répertoire parent via les identifiants UID/GID.
- **Appel Système** : L'application transmet l'ordre au **Système de fichiers Pop!_OS** en utilisant les fonctions natives (os.remove pour un fichier ou shutil.rmtree pour un dossier).

3. Finalisation et Retours d'Erreurs

- **Confirmation** : Une fois la suppression physique confirmée par le noyau Linux, l'interface graphique est rafraîchie pour retirer l'élément de la vue utilisateur.
- **Gestion des Exceptions** :
    - **Permission refusée** : Si l'utilisateur n'a pas les droits Linux suffisants, un message "Action interdite" est renvoyé.
    - **Erreur d'intégrité** : Si l'élément est le dernier de l'utilisateur, un message d'avertissement spécifique s'affiche pour empêcher la suppression.

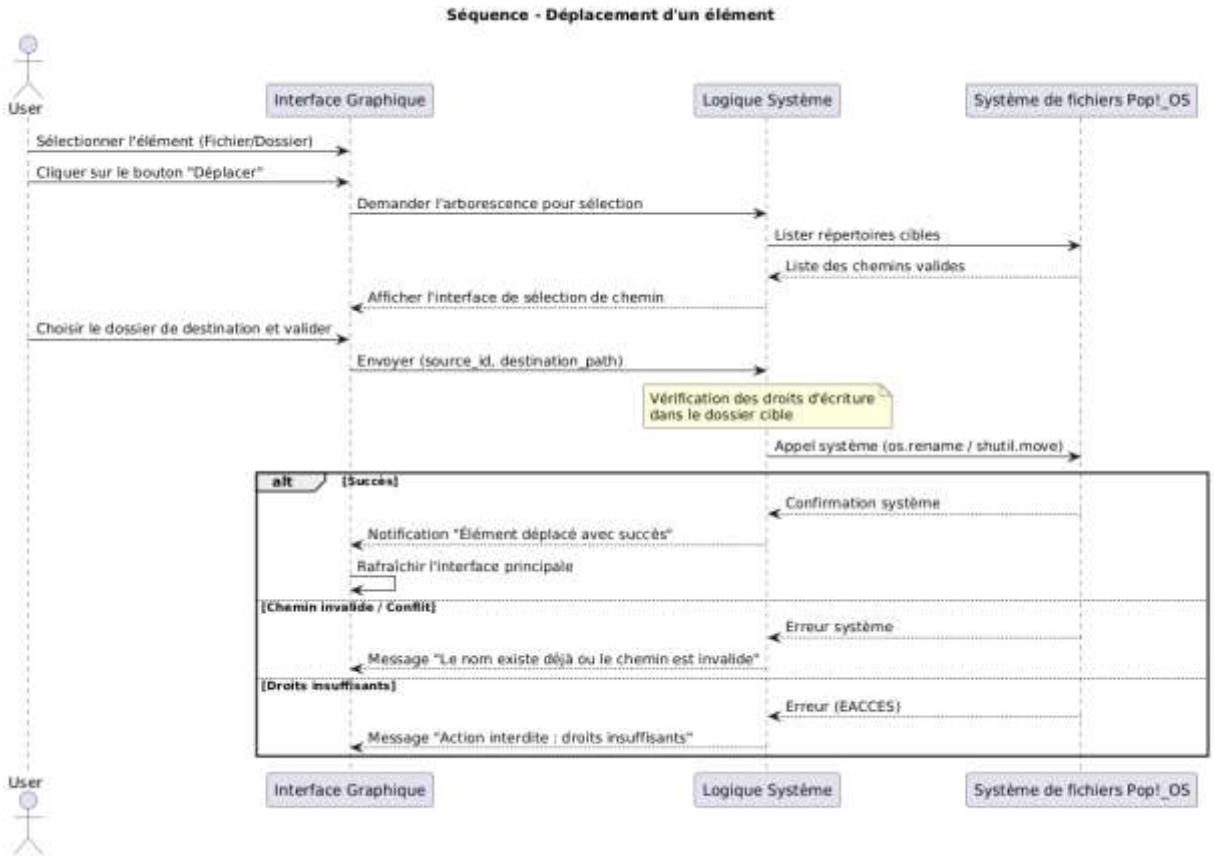

Figure 7:Diagramme de Séquence : Déplacement d'un élément

Diagramme de Séquence : Déplacement d'un élément

Ce diagramme décrit la procédure permettant de transférer un fichier ou un dossier d'un emplacement à un autre, impliquant une interaction entre l'utilisateur et l'arborescence système.

1. Initialisation et Navigation

- **Sélection de l'objet** : L'utilisateur sélectionne l'élément à déplacer et clique sur le bouton "Déplacer".
- **Interface de sélection** : L'**Interface Graphique** sollicite la **Logique Système** pour afficher l'arborescence des dossiers cibles disponibles.
- **Choix de destination** : L'utilisateur navigue dans cette interface, choisit le dossier de destination et valide son choix.

2. Validation et Exécution (Logique Système)

- **Contrôle de sécurité** : La **Logique Système** vérifie que l'utilisateur possède les droits d'écriture nécessaires dans le répertoire de destination.
- **Appel Système** : L'application exécute la commande de déplacement (os.rename ou shutil.move) auprès du **Système de fichiers Pop!_OS**. Cette opération est quasi

instantanée car elle ne déplace pas les données physiquement, mais modifie simplement leur adresse dans l'index du disque.

3. Mise à jour et Notifications

- **Succès de l'opération** :
  - Une fois le déplacement confirmé par le noyau Linux, la **Logique Système** met à jour le parent_id et le nouveau path de l'élément dans la base de données interne.
  - L'**Interface** affiche une notification de réussite et rafraîchit l'arborescence pour refléter le changement.
- **Gestion des erreurs** :
  - **Conflit de nom** : Si un élément identique existe déjà à destination, le système interrompt l'action et informe l'utilisateur.
  - **Droits insuffisants (EACCES)** : Si l'accès est refusé par Pop!_OS, un message d'erreur spécifique est affiché.

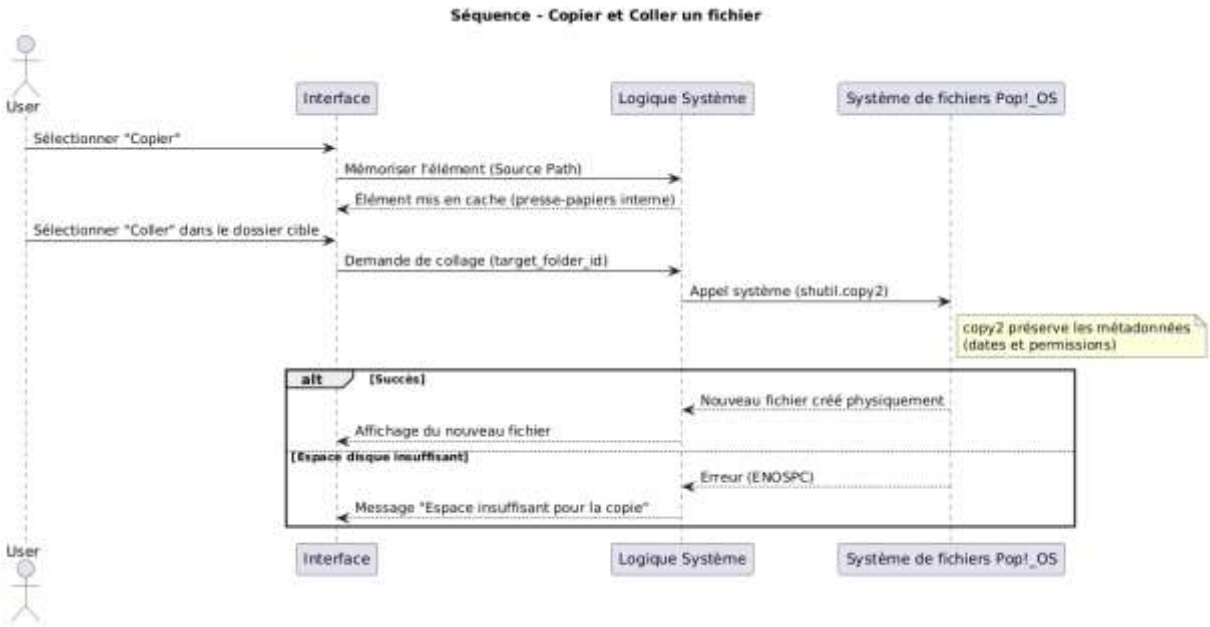

Figure 8:Diagramme de Séquence : Copier et Coller

Diagramme de Séquence : Copier et Coller

Ce diagramme détaille le processus de duplication d'un élément (fichier ou dossier) au sein de l'application, en mettant l'accent sur la création d'une nouvelle entité physique et logique.

1. Phase de Copie (Mise en cache)

- **Action Utilisateur** : L'utilisateur sélectionne un élément et clique sur "Copier".
- **Mémorisation** : L'**Interface** transmet le chemin source à la **Logique Système**.
- **Cache Interne** : L'application ne réalise aucune action sur le disque à ce stade ; elle stocke simplement la référence (chemin source) dans un "presse-papiers" interne à la session.

2. Phase de Collage (Exécution physique)

- **Action Utilisateur** : L'utilisateur navigue vers le dossier de destination et clique sur "Coller".
- **Validation** : La **Logique Système** vérifie la validité du chemin cible et les droits d'écriture de l'utilisateur.
- **Appel Système** : L'application commande au **Système de fichiers Pop!_OS** d'exécuter la copie physique (via une fonction type shutil.copy2 en Python). Cette commande est privilégiée car elle préserve les métadonnées (permissions et dates) de l'original.

3. Finalisation et Gestion des erreurs

- **Scénario de Succès** :
    - Le noyau confirme la création du nouveau fichier.
    - La Logique Système génère une nouvelle entrée dans la base de données avec un nouvel id et le nouveau parent_id.
    - L'Interface affiche immédiatement le nouvel élément dans le dossier cible.
- **Gestion des Échecs** :
    - **Espace disque insuffisant (ENOSPC)** : Si le disque est plein, l'erreur est interceptée et l'utilisateur est notifié.
    - **Conflit de nom** : Si un fichier identique existe déjà, la logique système peut proposer de renommer automatiquement la copie (ex: "fichier_copie.txt").

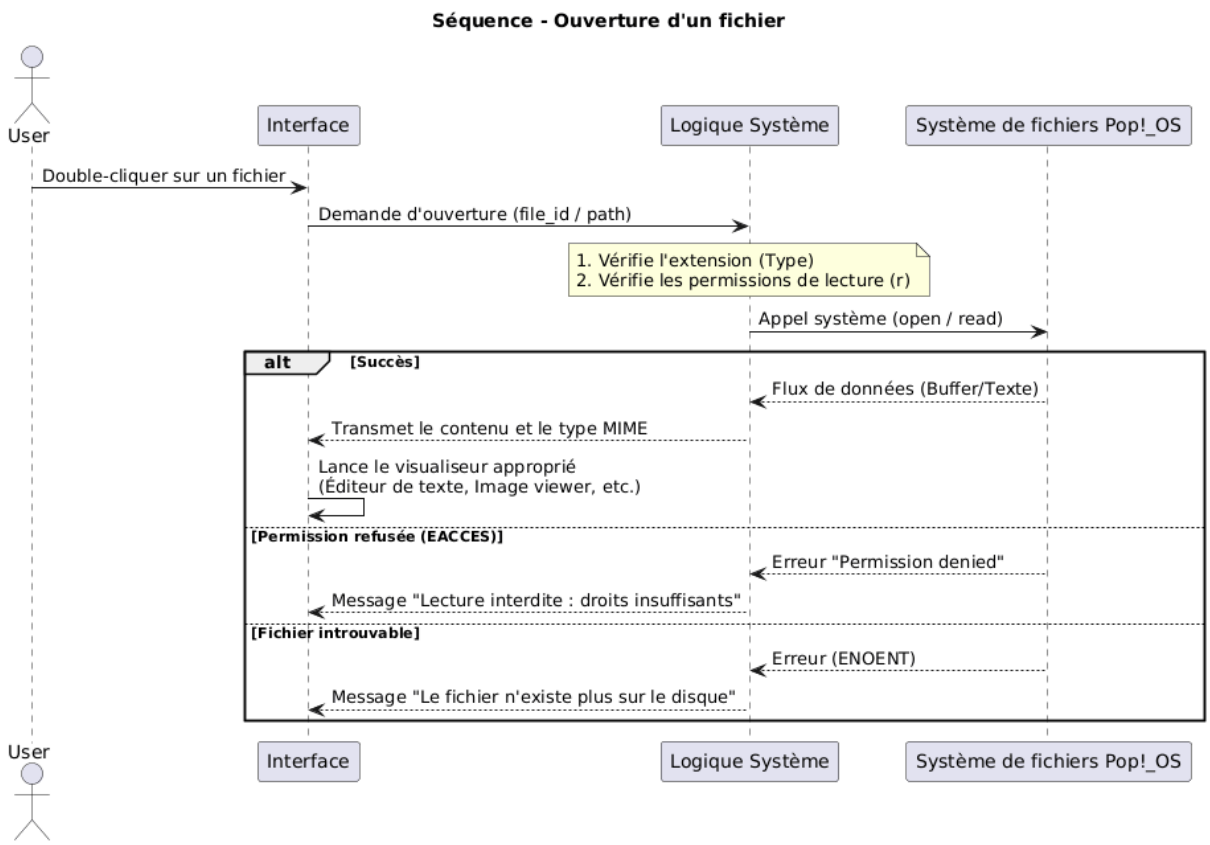

Figure 9:Diagramme de Séquence : Ouverture d'un fichier

Diagramme de Séquence : Ouverture d'un fichier

Ce diagramme illustre le flux d'exécution et les interactions nécessaires pour accéder au contenu d'un fichier stocké sur le système **Pop!_OS**.

1. Déclenchement et Requête

- **Action Utilisateur** : Le processus débute lorsque l'utilisateur effectue un double-clic sur un fichier spécifique dans l'**Interface Graphique**.
- **Transmission** : L'interface envoie une requête d'ouverture à la **Logique Système**, contenant l'identifiant (file_id) ou le chemin (path) du fichier cible.

2. Traitement et Validation (Logique Système)

- **Vérification métier** : Avant toute action physique, la Logique Système analyse le type de fichier (extension) et vérifie si l'utilisateur possède les **droits de lecture (r)** requis.
- **Appel Système** : Une fois validée, la logique métier effectue l'appel système correspondant (open / read) auprès du **Noyau Linux (Pop!_OS)**.

3. Gestion des Flux et des Erreurs

- **Scénario Nominal (Succès)** :
    - Le système de fichiers retourne le flux de données.
    - La Logique Système transmet ce contenu ainsi que le type MIME à l'Interface.
    - L'Interface lance le visualiseur approprié (éditeur de texte, visionneuse d'images, etc.).
- **Scénarios Alternatifs (Gestion des exceptions)** :
    - **Droits insuffisants (EACCES)** : Si le système de fichiers refuse l'accès, une notification d'erreur est renvoyée à l'utilisateur.
    - **Fichier introuvable (ENOENT)** : Si le fichier a été déplacé ou supprimé en dehors de l'application, un message d'erreur spécifique informe l'utilisateur que l'élément n'existe plus sur le disque.

CONCLUSION

Ce cahier de conception a permis de définir l’architecture et les modèles techniques nécessaires à la réalisation du gestionnaire de fichiers multi-utilisateurs. Les diagrammes UML présentés constituent une base solide pour la phase d’implémentation, garantissant une application cohérente, sécurisée et conforme aux objectifs définis.